

Министерство науки и высшего образования Российской Федерации
ФГАОУ ВО «Северо-Восточный федеральный университет имени
М.К.Аммосова»

Институт математики и информатики
Кафедра информационных технологий

Выпускная квалификационная работа

02.03.02 «Фундаментальная информатика и информационные технологии»

Выполнил: студент гр. ФИИТ 16
ИМИ СВФУ

Ефимов Максим Германович

Научный руководитель: доцент
кафедры ИТ Павлов Александр
Викторович

(оценка руководителя)

(подпись руководителя)

Якутск, 2020

СОДЕРЖАНИЕ

Введение	4
Глава 1. Теория	5
Анализ существующих решений.....	5
Algorithmia	5
DeOldify	5
MyHeritage	6
Colourise.sg.....	6
Сравнение результатов	6
Вывод	7
Основные понятия	7
Глубокое обучение	7
Свёрточные нейронные сети	7
Generative adversarial network (GAN)	8
Цветовое пространство LAB	10
Feature Loss	10
U-Net	11
Глава 2. Разработка решения	13
Архитектура модели	13
Тренировочные данные	13
Feature Loss	14
Дискриминатор	14
Генератор.....	15
GAN	18
Цикл тренировки	18
Конвертация в CoreML	21
Разработка приложения	22
Настройка.....	22
Функция обработки	22
Обработка результата	23
Примеры	25
Заключение.....	27
Список литературы	28

Приложение А. Исходный код модели	30
Приложение Б. Исходный код приложения	43
ViewController.swift.....	43
SaveViewController.swift.....	47
CGImagePropertyOrientation.swift	49

ВВЕДЕНИЕ

Актуальность: Первая черно-белая фотография была сделана 1827 году [1], и с тех до распространения цветных камер были сняты миллионы черно-белых фотографий и видео. Также большинство фотографий и документов, таких как книги, газеты, манги и т. п. ради дешевизны по сей день печатаются в черно-белом варианте.

На данный момент черно-белые изображения в основном раскашиваются вручную с помощью программы Photoshop. Это занимает от нескольких часов до нескольких дней и требует специальных умений [2]. Широко известен опыт колоризации черно-белых кинофильмов, отдельные компании утверждают, что используют автоматизацию на основе нейронных сетей, позволяющая выделять области для раскрашивания разными цветами автоматически, что позволяет снизить расходы на колоризацию. Известно, что раскраска каждой минуты сериала 17 мгновений весны в 2009 году обошлась в 3000 долларов за минуту [3].

Цель исследования: Разработка приложения под iOS для раскраски черно-белых изображений методами машинного обучения.

Объект исследования: Методы машинного обучения.

Предмет исследования: Применение нейронных сетей для колоризации изображений.

Задачи исследования:

1. Изучить алгоритмы глубокого обучения
2. Сделать модель для раскраски черно-белых изображений
3. Сконвертировать модель в CoreML
4. На основе модели разработать приложение для iOS

ГЛАВА 1. ТЕОРИЯ

Анализ существующих решений

Algorithmia

Алгоритм разработанный Ричардом Чжаном в 2016 году. Модель представляет из себя feed-forward CNN и тренирована на более чем миллионе фотографий ImageNet. Реализовано на устаревшем на данный момент фреймворке caffe.

Модель использует цветовую модель CIELab. В качестве входа используется L канал, а в качестве выхода каналы a и b. Минус такого подхода в том что сложно использовать предобученные модели, потому что они, как правило, обучаются с использованием цветовой модель RGB. [4]

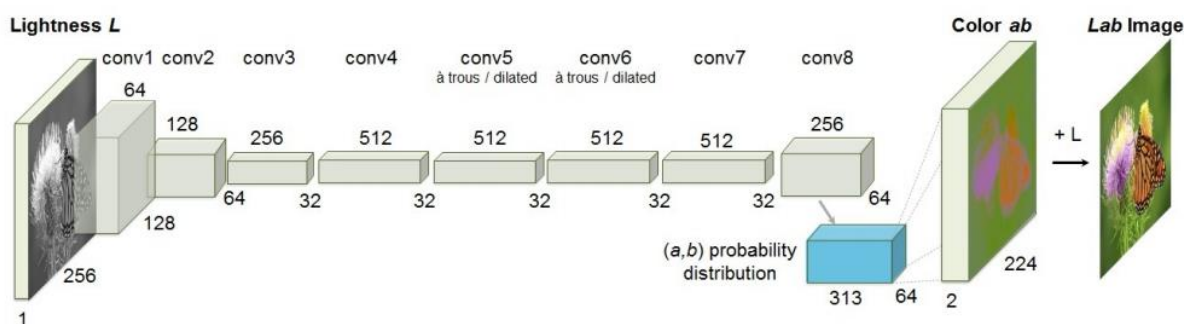


Рисунок 1 Архитектура модели Algorithmia

DeOldify

Модель глубокого обучения с использованием архитектуры GAN. Генеративная модель – это предобученная на imagenet модель U-Net. Функция потерь состоит из двух частей. Первая это Perceptual Loss (Feature Loss) основанная на VGG16. Вторая это функция потерь дискриминационной модели. Реализована с использованием фреймворка Fast.AI.

Модель является весьма требовательным к ресурсам поэтому его нельзя запустить на телефоне без сильной оптимизации. Плюс к этому у фреймворка Fast.AI слабая поддержка конвертации в CoreML. [5]

MyHeritage

Платная версия DeOldify, которая по заверениям разработчиков работает лучше [6]. Можно бесплатно обработать до 10 фотографий, далее нужно оформить подписку за 10\$ в год.

Colourise.sg

Бесплатный сайт с закрытым исходным кодом вдохновленный DeOldify. Разработан во время хакатона технологическим отделом правительства Сингапура. [7]

Сравнение результатов

Будем сравнивать качество существующих решений на нескольких фотографиях с сайта Unsplash. Для этого возьмем несколько цветных фотографий, сделаем их черно белыми и обработаем через модели.

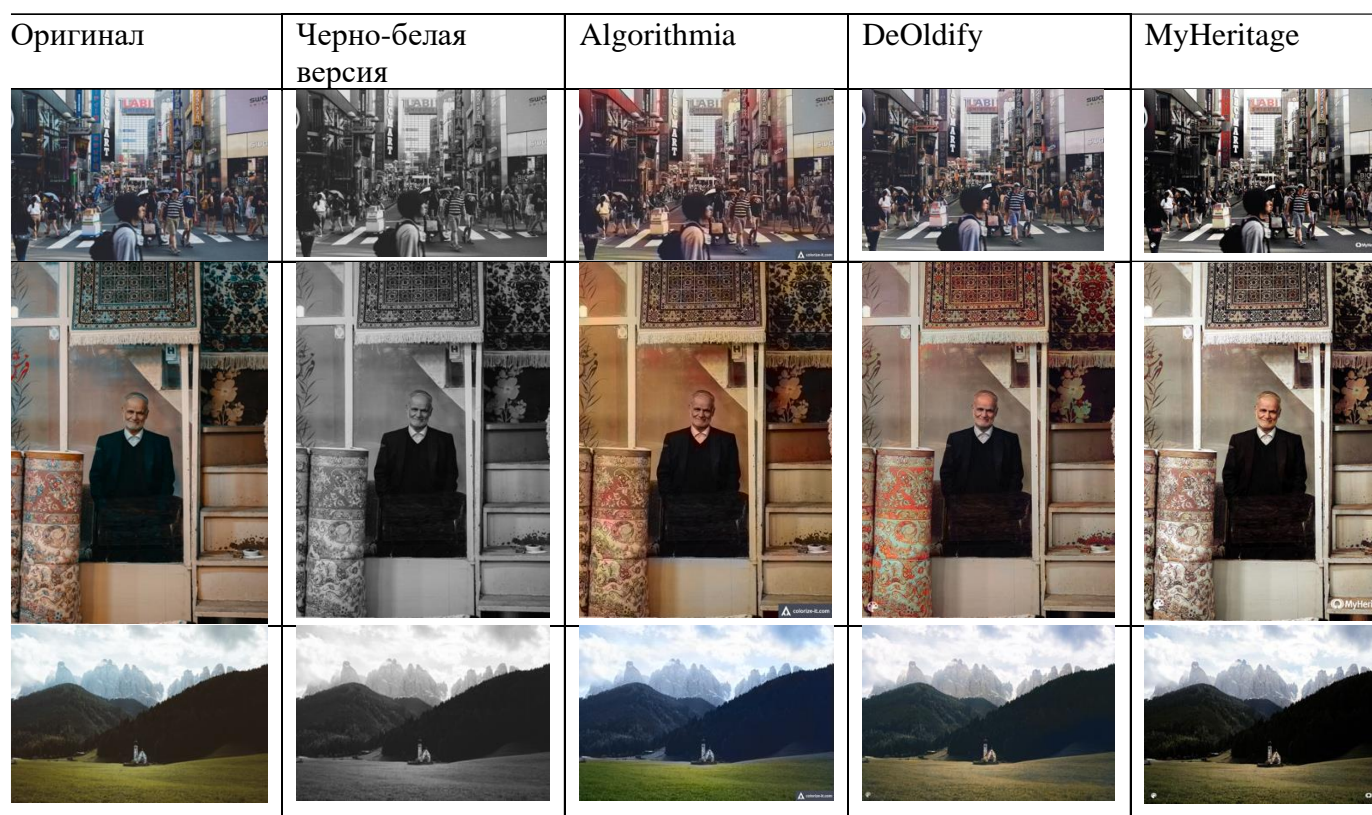


Рисунок 2 Сравнение аналогов

Вывод

Учитывая что MyHeritage и Colourise являются ПО с закрытым исходным кодом, а Algorithmia работает не очень хорошо, самым интересным для исследования является DeOldify. Учитывая сложности конвертации модели DeOldify было решено переписать его оптимизированную под телефоны версию с использованием популярного фреймворка Keras, потому что у него хорошая поддержка конвертации в CoreML.

Основные понятия

Глубокое обучение

Глубокое обучение — семейства методов машинного обучения, основанных на имитации работы человеческого мозга в процессе обработки данных и создания паттернов. Использует многослойную систему нелинейных фильтров для извлечения признаков с преобразованиями. Каждый последующий слой получает на входе выходные данные предыдущего слоя [8]

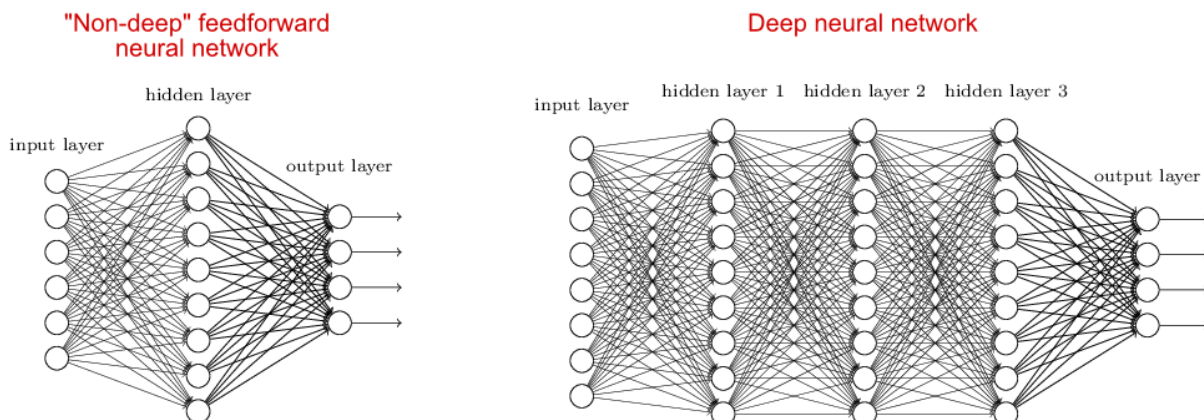


Рисунок 3 Глубокие и неглубокие нейронные сети.

Свёрточные нейронные сети

Сверточная нейронная сеть — специальная архитектура нейронных сетей, предложенная Яном Лекуном, изначально нацеленная на эффективное распознавание изображений.

Две основные концепции в свёрточных нейронных сетях:

- свёртка

- операция подвыборки (pooling, max pooling)

Свёртка — процесс применения фильтра («ядра») к изображению.

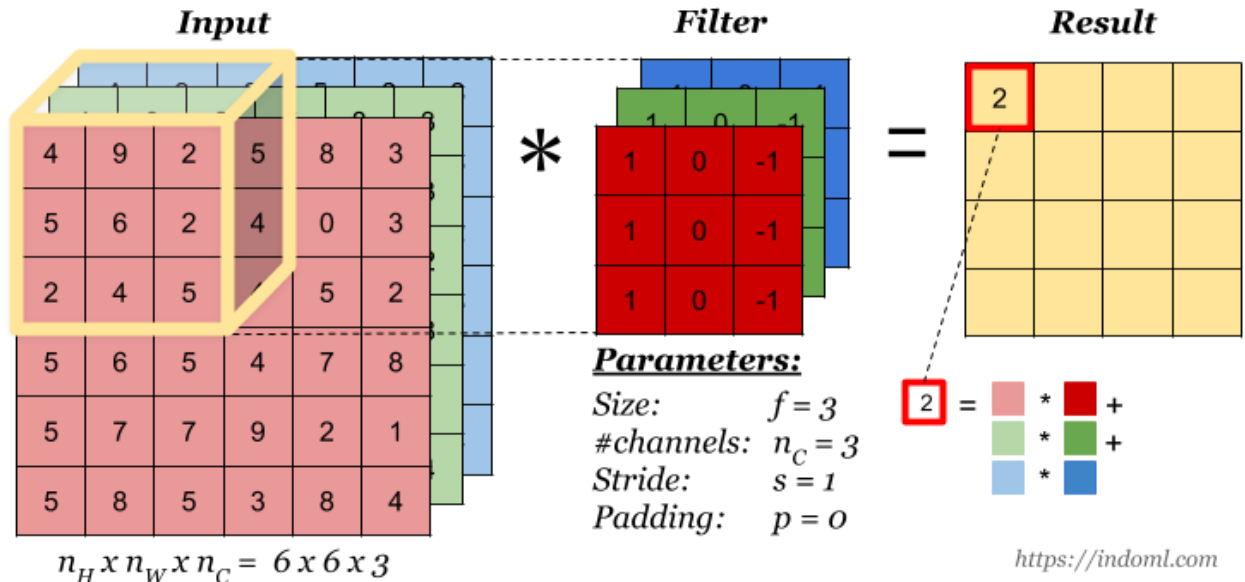


Рисунок 4 Пример свертки

Операция подвыборки по максимальному значению — процесс уменьшения размеров изображения через объединение группы пикселей в единое максимальное значение из этой группы.

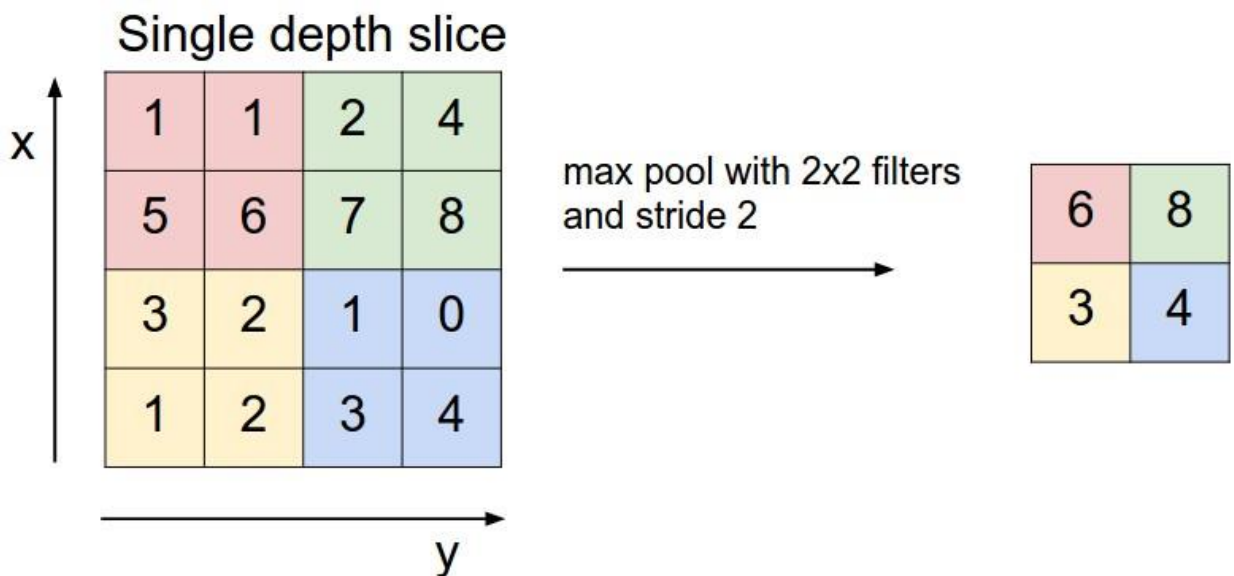


Рисунок 5 Пример пулинга по максимуму

Generative adversarial network (GAN)

Генеративно-состязательная сеть — это класс моделей машинного обучения без учителя, в котором используется две нейронных сети, одна из

которых генерирует новые объекты (генеративная модель), а другая старается отличить правильные объекты от неправильных (дискриминативная модель). Алгоритм был разработан Яном Гудфеллоу и исследователями из университета Монреаля в 2014 году [9].

Применение

GAN используется в генерации изображений, музыки, голоса и текстов, которые человеку сложно отличить от настоящих объектов окружающего мира. Благодаря этому алгоритм нашел применение в методике Deepfake синтеза новых изображений и видео на основе существующих [10].

В последнее время системы GANs стали использоваться для подготовки кадров фильмов или мультипликации. Кроме того, GAN может использоваться для улучшения качества нечётких или частично испорченных фотографий.

Как работает GAN

Дискриминативная модель – это стандартная сверточная сеть, которая классифицирует поданные на вход объекты. Генеративная модель – это обратная сверточная сеть, которая на основе случайного шума создает объект

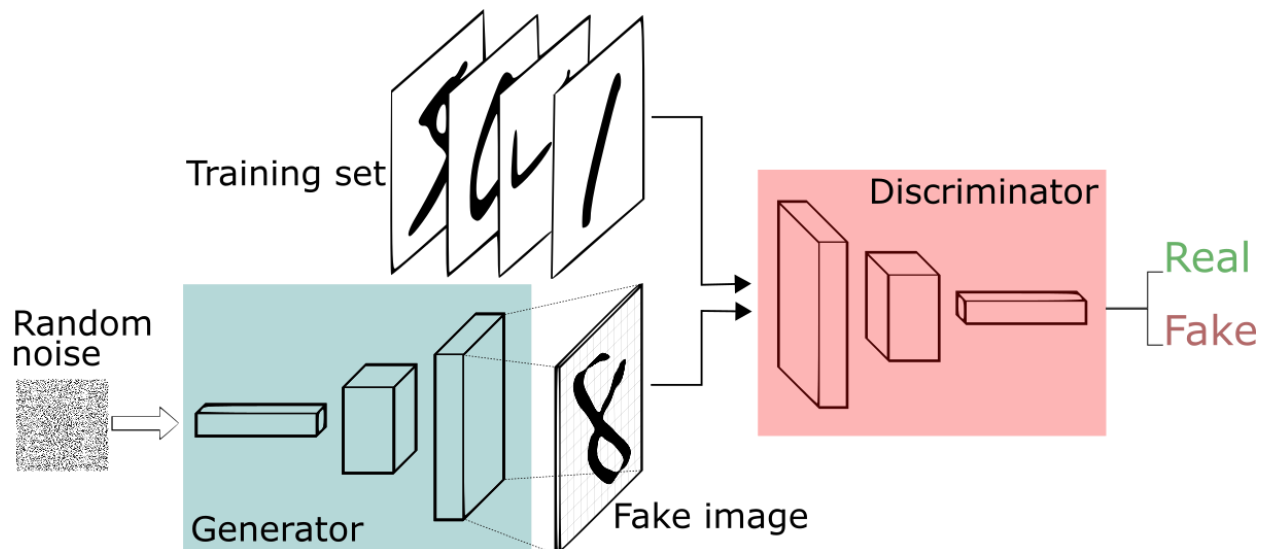


Рисунок 6 Архитектура GAN

Цветовое пространство LAB

LAB – это цветовое пространство разработанное международной комиссией по освещению (CIE) в 1976 году. Оно кодирует цвет тремя числами, L – это яркость серого, a – зеленый-красный и b – синий-желтый оттенки.

LAB был разработан после теории противоположных цветов, согласно которой два цвета не могут быть зелеными и красными в одно время или желтыми и синими в одно время.

Он удобен тем, что в черно-белых изображения присутствует лишь компонента L , то есть в модели компонента L является входом, по которому мы предсказываем a и b . [11]

Feature Loss

Feature Loss – эта техника при которой вместо попиксельного сравнения картинок сравниваются высокоуровневые характеристики картинок. Для этого используется предобученная модель классификации из которой извлекается слой характеристик и генеративная модель старается восстановить эти характеристики. Характеристики представлены числами, минимизируется разница между ними, обычно используется среднеквадратическая ошибка. Предобученная модель во время обучения замораживается. [12]

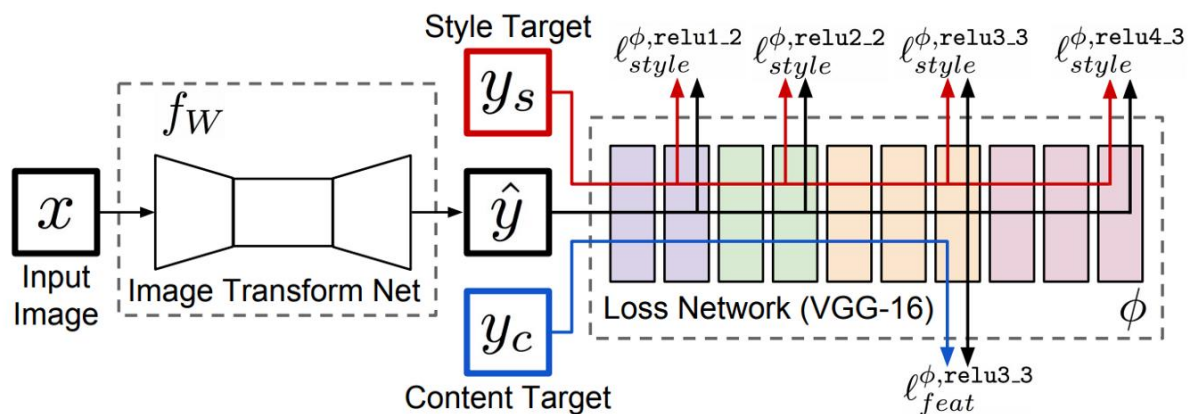


Fig. 2. System overview. We train an *image transformation network* to transform input images into output images. We use a *loss network* pretrained for image classification to define *perceptual loss functions* that measure perceptual differences in content and style between images. The loss network remains fixed during the training process.

U-Net

U-Net — это свёрточная нейронная сеть, которая была создана в 2015 году для сегментации биомедицинских изображений в отделении Computer Science Фрайбургского университета.

Архитектура сети

Сеть содержит сжимающий путь (слева) и расширяющий путь (справа), поэтому архитектура похожа на букву U, что и отражено в названии. На каждом шаге мы удваиваем количество каналов признаков.

Сжимающий путь похож на типичную свёрточную сеть, он содержит два подряд свёрточных слоя 3×3 , после которых идет слой ReLU и пулинг с функцией максимума 2×2 с шагом 2.

Каждый шаг расширяющего пути содержит слой, обратный пулингу, который расширяет карту признаков, после которого следует свертка 2×2 , которая уменьшает количество каналов признаков. После идет конкатенация с соответствующим образом обрезанной картой признаков из сжимающего пути и две свертки 3×3 , после каждой из которой идет ReLU. Обрезка нужна из-за того, что мы теряем пограничные пиксели в каждой свёртке. На последнем слое свертка 1×1 используется для приведения каждого 64-компонентного вектора признаков до требуемого количества классов.

Всего сеть имеет 23 свёрточных слоя. [13]

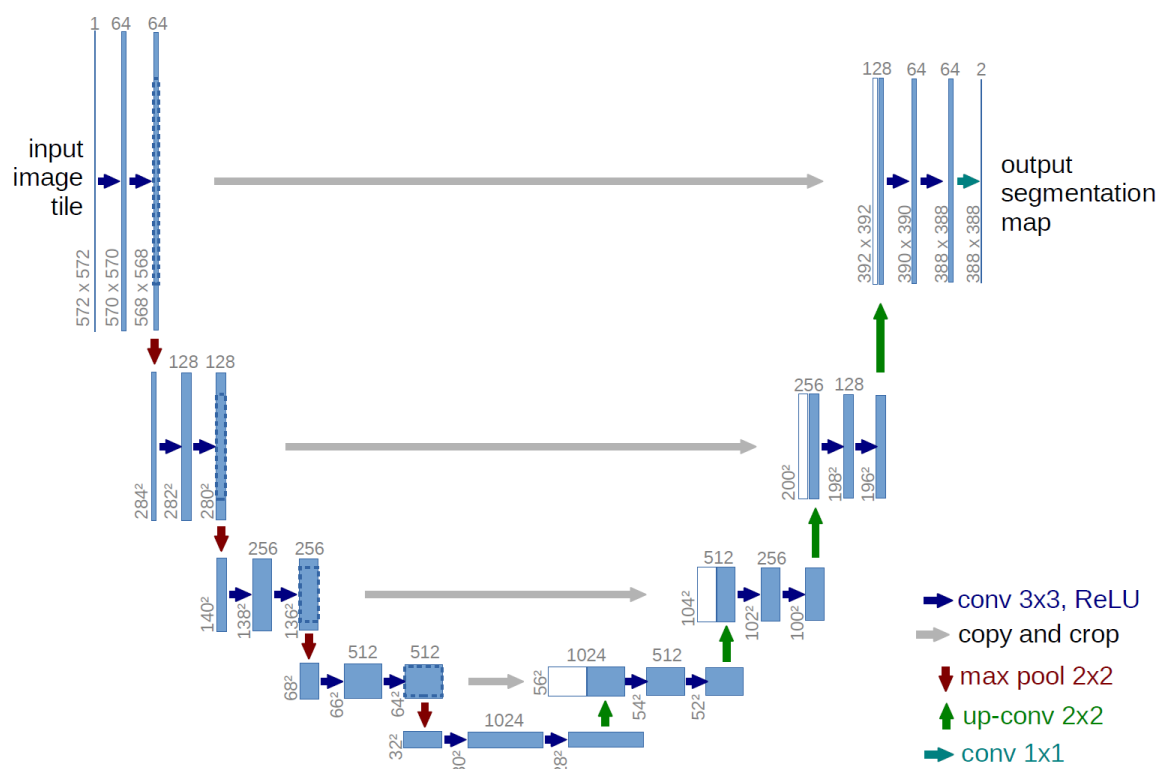


Рисунок 8 Архитектура U-Net

ГЛАВА 2. РАЗРАБОТКА РЕШЕНИЯ

Архитектура модели

Модель является глубокой нейронной сетью GAN где генеративная модель это U-Net. Функция потерь это комбинация Feature Loss и потери дискриминатора.

Тренировочные данные

Тренировочные данные можно получить очень просто – любое цветное изображение может быть преобразовать в черно-белое и можно составить из них пару. Все фотографии нормализуются в диапазоне от -1 до 1.

```
class DataLoader():
    def __init__(self, dataset_name, img_res=(256, 256)):
        self.dataset_name = dataset_name
        self.img_res = img_res

    def load_data(self, batch_size=1, is_testing=False):
        path = glob('./datasets/%s/*' % (self.dataset_name))
        batch_images = np.random.choice(path, size=batch_size)

        imgs_hr = []
        imgs_lr = []
        for img_path in batch_images:
            img_hr, img_lr = self._load(img_path, self.img_res)

            imgs_hr.append(img_hr)
            imgs_lr.append(img_lr)

        # нормализация данных
        imgs_hr = np.array(imgs_hr) / 127.5 - 1
        imgs_lr = np.array(imgs_lr) / 127.5 - 1

        return imgs_hr, imgs_lr

    # returns pair (original photo, grayscale photo)
    def _load(self, path, size):
        return np.array(PIL.Image.open(path).resize(size)).astype(np.float), np.expand_dims(np.array(PIL.Image.open(path).resize(size).convert('L')).astype(np.float), axis=2)
```

Figure 1 Класс Загрузки данных

Feature Loss

В качестве модели для feature loss используется модель VGG19 предобученная на датасете imagenet. В качестве output-а используется не вероятности классов, а из него извлекаются слой характеристик. Модель во время обучения GAN не обучается. В качестве функции потерь используется среднеквадратичная ошибка.

```
def build_vgg():
    vgg = VGG19(weights='imagenet')
    vgg.outputs = [vgg.layers[9].output]
    img = Input(shape=hr_shape)
    img_features = vgg(img)
    return Model(img, img_features)
vgg = build_vgg()
vgg.trainable = False
vgg.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
```

Figure 2 Модель feature loss

Дискриминатор

Дискриминатор получает на вход картинку и старается угадать класс картинки: настоящая или сгенерированная. Функция активации на выходном слое сигмоида и указывает на вероятность класса где 0 это сгенерированная картинка 1 настоящая. Функция потерь среднеквадратичная ошибка.

```

# calculate output shape of D (PatchGAN)
patch = int(hr_height / 2**4)
disc_patch = (patch, patch, 1)
# Number of filters in the first layer of G and D
gf = 64
df = 64

def build_discriminator():
    def d_block(layer_input, filters, strides=1, bn=True):
        """Discriminator layer"""
        d = Conv2D(filters, kernel_size=3, strides=strides, padding='same')(layer_in-
put)
        d = LeakyReLU(alpha=0.2)(d)
        if bn:
            d = BatchNormalization(momentum=0.8)(d)
        return d

    # Input img
    d0 = Input(shape=hr_shape)

    d1 = d_block(d0, df, bn=False)
    d2 = d_block(d1, df, strides=2)
    d3 = d_block(d2, df*2)
    d4 = d_block(d3, df*2, strides=2)
    d5 = d_block(d4, df*4)
    d6 = d_block(d5, df*4, strides=2)
    d7 = d_block(d6, df*8)
    d8 = d_block(d7, df*8, strides=2)

    d9 = Dense(df*16)(d8)
    d10 = LeakyReLU(alpha=0.2)(d9)
    validity = Dense(1, activation='sigmoid')(d10)

    return Model(d0, validity)

discriminator = build_discriminator()
discriminator.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])

```

Figure 3 Дискриминатор

Генератор

Генератор это стандартный unet. Выходных слоев 3 соответственно на каждый канал RGB. Функция активации на выходе тангенс потому что мы нормализовали наши данные в диапазоне от -1 до 1.

```

def unet(pretrained_weights = None, input_size = (256, 256, 3)):
    inputs = Input(input_size)

```

```

conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(inputs)
conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(pool1)
conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(pool2)
conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(pool3)
conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv4)
drop4 = Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializ
er = 'he_normal')(pool4)
conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializ
er = 'he_normal')(conv5)
drop5 = Dropout(0.5)(conv5)

up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(drop5))
merge6 = concatenate([drop4,up6], axis = 3)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(merge6)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv6)

up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(conv6))
merge7 = concatenate([conv3,up7], axis = 3)

```



```

conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(merge7)
conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv7)

up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(conv7))
merge8 = concatenate([conv2,up8], axis = 3)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(merge8)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv8)

up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv8))
merge9 = concatenate([conv1,up9], axis = 3)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(merge9)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv9)
conv9 = Conv2D(3, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv9)
conv10 = Conv2D(3, 1, activation = 'tanh')(conv9)

model = Model(input = inputs, output = conv10)

#model.summary()

if(pretrained_weights):
    model.load_weights(pretrained_weights)

return model
generator = unet(input_size=lr_shape)
generator.compile(optimizer = Adam(lr = 1e-
4), loss = 'mse', metrics = ['accuracy'])
generator.summary()

```

GAN

GAN состоит из трех моделей: генератора, дискриминатора и VGG для feature loss. Сначала черно-белая картинка подается на вход генератору, а результат генератора имеет два пути:

- В первом случае она подается на вход дискриминатору который предсказывает класс картинки. Функция потерь здесь бинарная кросс энтропия.

- Второй путь это feature loss, из результата извлекаются характеристики. Функция потерь здесь среднеквадратичная ошибка.

Притом вес у функции потери feature loss больше, то есть модель будет учитывать его сильнее. Дискриминатор во время обучения GAN не обучается.

```
# High res. and low res. images
img_lr = Input(shape=lr_shape)
# generate high res. version from low res.
fake_hr = generator(img_lr)
# extract image features of the generated img
fake_features = vgg(fake_hr)
# for the combined model we will only train the generator
discriminator.trainable = False
# Discriminator determines validity of generated high res. images
validity = discriminator(fake_hr)
combined = Model([img_lr], [validity, fake_features])
combined.compile(loss=['binary_crossentropy', 'mse'], loss_weights=[1e-3, 1], optimizer=optimizer)
```

Figure 4GAN

Цикл тренировки

Первым делом загружаются пары черно-белых и цветных фотографий. Потом генератор генерирует цветную версию из черно-белой версии. Оригинальная цветная и сгенерированная цветная подаются на обучение дискриминатору. Далее обучается GAN, ему на вход подается черно-белые фотографии. У GAN два выхода, первый это выход дискриминатора, второй это характеристики. Первый выход сравнивается с единицами, потому что

генератору нужно генерировать правдоподобные картинки, второй выход сравнивается с характеристиками оригинальной цветной фотографии.

```
! rm -rf ./images
epochs=2000
batch_size=1
sample_interval=50
start_time = datetime.datetime.now()
for epoch in range(epochs+1):
    # -----
    # Train Discriminator
    # -----

    # Sample images and their conditioning counterparts
    imgs_hr, imgs_lr = data_loader.load_data(batch_size)

    # From low res. image generate high res. version
    fake_hr = generator.predict(imgs_lr)

    valid = np.ones((batch_size,) + disc_patch)
    fake = np.zeros((batch_size,) + disc_patch)

    # Train the discriminators (original images = real / generated = Fake)
    d_loss_real = discriminator.train_on_batch(imgs_hr, valid)
    d_loss_fake = discriminator.train_on_batch(fake_hr, fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # -----
    # Train Generator
    # -----

    # Sample images and their conditioning counterparts
    imgs_hr, imgs_lr = data_loader.load_data(batch_size)

    # The generators want the discriminators to label the generated images as real
    valid = np.ones((batch_size,) + disc_patch)

    # Extract ground truth image features using pre-trained VGG19 model
    image_features = vgg.predict(imgs_hr)

    # Train the generators
    g_loss = combined.train_on_batch([imgs_lr], [valid, image_features])

    elapsed_time = datetime.datetime.now() - start_time
    # Plot the progress
    print ("%d time: %s" % (epoch, elapsed_time))

    # If at save interval => save generated image samples
    if epoch % sample_interval == 0:
        sample_images(epoch)
```

Примеры

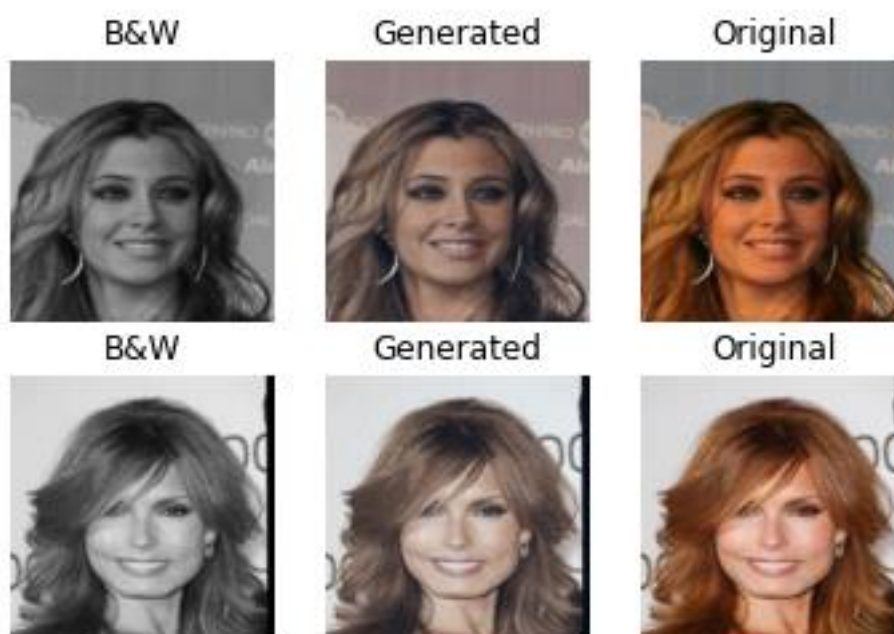


Рисунок 9 Пример 1



Рисунок 10 Пример 2



Рисунок 11 Пример 3

Конвертация в CoreML

Запускать модель на телефоне вместо обработки картинок на сервере имеет много преимуществ. Основные из них:

- **Бесплатные вычисления.** Аренда сервера, интернет трафик стоят денег, это особенно критично для бесплатных приложений.
- **Нет потенциальных проблем с безопасностью.** При хранении и пересылке пользовательских данных всегда есть вероятность кражи, утери. При обработке данных на телефоне у Вас гарантированно нет такой проблемы.
- **Неограниченная масштабируемость.** При большом количестве пользователей сервер может не справиться с нагрузкой.

CoreML – это фреймворк разработанный компанией Apple для интеграции моделей машинного обучения в телефоны операционной системы iOS.

Core ML Community Tools – это библиотека от сообщества для конвертации, редактирование и валидации моделей машинного обучения таких как TensorFlow, Keras, Caffe, scikit-learn и других.

```
generator.save('generator.h5')
! pip install coremltools
! pip install keras==2.2.4
! pip install tensorflow==1.14.0
import coremltools
coreml_model = coremltools.converters.keras.convert('generator.h5', add_custom_layers=True)

# Saving the Core ML model to a file.
coreml_model.save('colorizer.mlmodel')
```

Рисунок 12 Конвертация в CoreML

Разработка приложения

Приложение написано на языке Swift в среде разработки Xcode. В приложение можно загрузить любое черно-белое изображение из библиотеки и получить его раскрашенную с помощью модели версию, которую потом можно сохранить. Для запуска модели и предобработки изображений используется фреймворк Vision.

Настройка

CoreML модель которая была получена в результате конвертации перетаскивается внутрь проекта Xcode. Классы необходимые для его работы генерируются автоматически CoreML.

```
let model = try! VNCoreMLModel(for: Colorizer512().model)
```

Рисунок 13 Инициализация модели

Функция обработки

Чтобы настроить запрос Vision с использованием модели, создайте экземпляр класса VNCoreMLRequest. В его обработчике завершения вызывается функция обработки результата.

Объекту класса `VNImageRequestHandler` передается изображение и вызывается метод `perform` с ранее созданным объектом `request`. Этот метод выполняется синхронно, поэтому оно запускается в фоновом потоке чтобы приложение не блокировалось во время обработки.

```
func process(_ image: UIImage) {
    activityIndicator.startAnimating()
    startButton.setTitle("", for: .normal)
    startButton.isEnabled = false
    let request = VNCoreMLRequest(model: model) { (request, error) in
        self.processResult(for: request, error: error)
    }
    request.imageCropAndScaleOption = .scaleFit

    DispatchQueue.global(qos: .userInitiated).async {
        let handler = VNImageRequestHandler(cgImage: image.cgImage!)
        do {
            try handler.perform([request])
        } catch {
            print("Failed to perform classification.\n\(error.localizedDescription)")
        }
    }
}
```

Обработка результата

Обработчик завершения запроса `Vision` указывает, был ли запрос успешным или привел к ошибке. В случае успеха результат содержит свойство типа `CVPixelBuffer`, который может быть сконвертирован в `UIImage`.

```

func processResult(for request: VNRequest, error: Error?) {
    DispatchQueue.main.async {
        guard let results = request.results as? [VNPixelBufferObservation] else {
            print("error", error ?? "")
            return
        }
        print(results[0].pixelBuffer)
        let ciImage = CIImage(cvPixelBuffer: results[0].pixelBuffer)
        let cgImage = self.convertCIImageToCGImage(inputImage: ciImage!)
        var uiImage = UIImage(cgImage: cgImage)
        let scale = uiImage.size.height / max(self.originalImage!.size.height, self.originalImage!.size.width)
        uiImage = self.cropToBounds(image: uiImage, width: self.originalImage!.size.width * scale, height: self.originalImage!.size.height * scale)
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let newViewController = storyboard.instantiateViewController(withIdentifier: "SaveViewController") as! SaveViewController
        newViewController.image = uiImage

        self.show(newViewController, sender: self)

        self.activityIndicator.stopAnimating()
        self.startButton.setTitle("Let's start", for: .normal)
        self.startButton.isEnabled = true
    }
}

```


Примеры

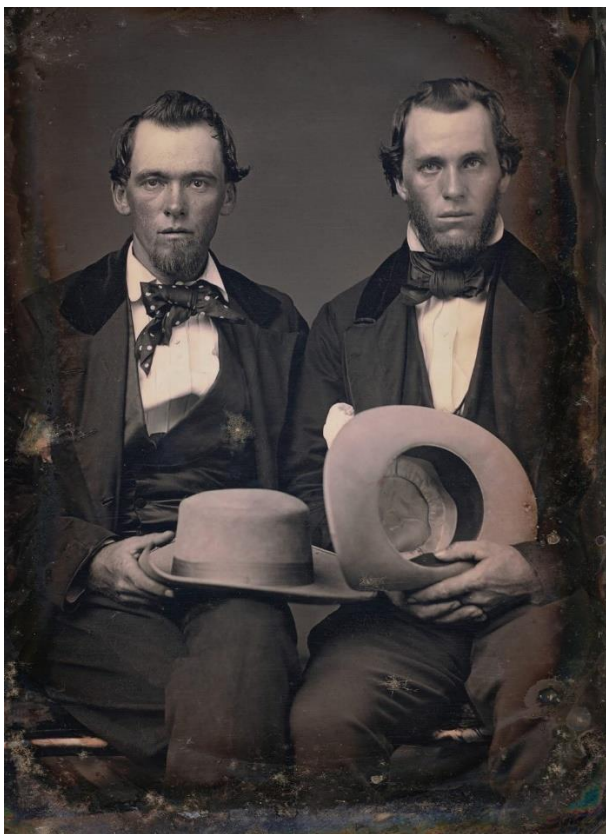


Рисунок 14 Пример. Оригинал

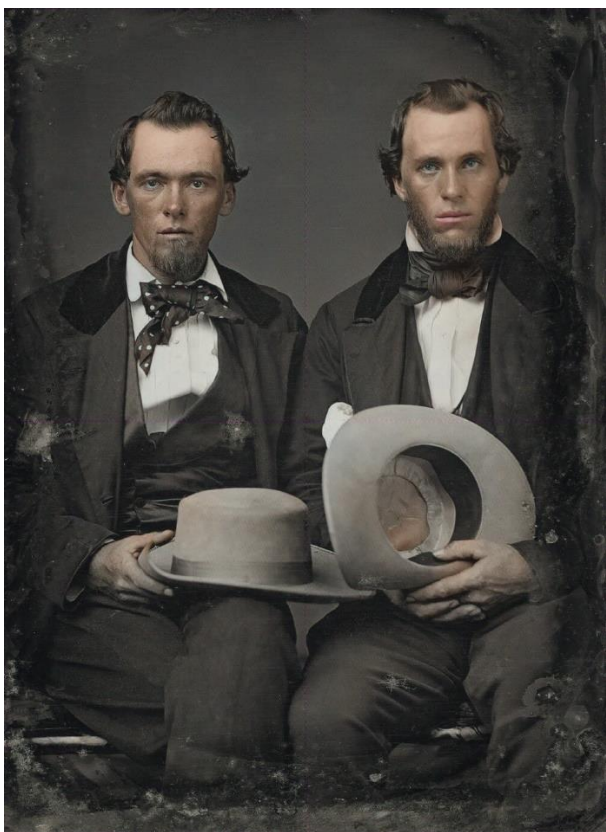


Рисунок 15 Пример. Сгенерированная

ЗАКЛЮЧЕНИЕ

В данной работе было разработано приложение для раскрашивания черно-белых изображений.

Были сложности с конвертацией модели DeOldify в CoreML, так как у фреймворке Fast.ai слабая поддержка конвертации в CoreML, поэтому модель была написана с нуля на фреймворке Keras.

Модель имеет архитектуру GAN где генератор это U-Net. Была использована техника Feature Loss.

Модель сконвертирована в CoreML с помощью библиотеки coreml-tools.

Приложение написано на языке Swift и работает на iOS 13 или выше. Для предобработки используется фреймворк Vision.

Приложение было опубликовано в магазине приложений App Store. Адрес приложения <https://apps.apple.com/ru/app/colorizer-ai-colorization/id1518362632?l=en>.

Исходный код проекта опубликован на платформе GitHub по адресу <https://github.com/Hapsidra/colorizer>

СПИСОК ЛИТЕРАТУРЫ

- [1] Wikipedia, «Хронология фотографии», [В Интернете]. Available: https://en.wikipedia.org/wiki/Timeline_of_photography_technology. [Дата обращения: 1 2020].
- [2] Vox, «Как одержимые художники раскрашивают старые фотографии», 2018. [В Интернете]. Available: https://www.youtube.com/watch?v=vubuBrcAwY&feature=emb_logo.
- [3] Время электроники, «Как раскрашивали «Семнадцать мгновений весны»,» 2009. [В Интернете]. Available: <http://www.russianelectronics.ru/leader-r/news/russianmarket/doc/43526/>. [Дата обращения: 2020].
- [4] R. Zhang, «Colorful Image Colorization», 2016. [В Интернете]. Available: <https://arxiv.org/pdf/1603.08511.pdf>.
- [5] J. Antic, «Github», [В Интернете]. Available: <https://github.com/jantic/DeOldify>. [Дата обращения: 15 май 2020].
- [6] MyHeritage, «Придайте цвет Вашей семейной истории», [В Интернете]. Available: <https://www.myheritage.com/incolor?lang=RU>. [Дата обращения: 22 март 2020].
- [7] P. Lim, «Bringing black and white photos to life using Colourise.sg — a deep learning colouriser trained with old Singaporean photos», 3 Февраль 2019. [В Интернете]. Available: <https://blog.data.gov.sg/bringing-black-and-white-photos-to-life-using-colourise-sg-435ae5cc5036>. [Дата обращения: 22 Март 2020].
- [8] Университет ИТМО, «Глубокое обучение», [В Интернете]. Available: http://neerc.ifmo.ru/wiki/index.php?title=%D0%93%D0%BB%D1%83%D0%B1%D0%BE%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5. [Дата обращения: 15 май 2020].

- [9] J. Romero, «A Beginner's Guide to GANs,» [В Интернете]. Available: <https://pathmind.com/wiki/generative-adversarial-network-gan>. [Дата обращения: 2020].
- [10] «Deepfake,» [В Интернете]. Available: <https://en.wikipedia.org/wiki/Deepfake>. [Дата обращения: 2020].
- [11] M. Rovai, «Colorizing Old B&W Photos and Videos With the Help of AI,» 22 3 2019. [В Интернете]. Available: <https://towardsdatascience.com/colorizing-old-b-w-photos-and-videos-with-the-help-of-ai-76ba086f15ec>. [Дата обращения: 24 1 2020].
- [12] A. A. a. L. F.-F. Justin Johnson, «Perceptual Losses for Real-Time Style Transfer,» [В Интернете]. Available: <https://cs.stanford.edu/people/jcjohns/papers/eccv16/JohnsonECCV16.pdf>. [Дата обращения: 15 Май 2020].
- [13] P. F. a. T. B. Olaf Ronneberger, «U-Net: Convolutional Networks for Biomedical,» [В Интернете]. Available: <https://arxiv.org/pdf/1505.04597.pdf>. [Дата обращения: 15 Май 2020].
- [14] J. Antic, «DeOldify,» 2018. [В Интернете]. Available: <https://habr.com/ru/post/428818/>. [Дата обращения: 23 01 2020].

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД МОДЕЛИ

```
# -*- coding: utf-8 -*-
```

```
"""keras_colorizer.ipynb
```

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/drive/1i6N0SoT9ErcicImh39FhawFeFrFTlJr6>

```
# Keras colorizer of CelebA using Generative Adversarial Networks.
```

```
The dataset can be downloaded from: https://www.dropbox.com/sh/8oqt9vytwxb3s4r/AA\_DIKlZ8PR9zr6Y20qbkunrba/Img/img\_align\_celeba.zip?dl=0
```

```
## Instrustion on running the script:
```

1. Download the dataset from the provided link
 2. Save the folder 'img_align_celeba' to 'datasets/'
- ```
"""
```

```
! python
```

```
tf.__version__
```

```
from keras import backend as K
```

```
K.tensorflow_backend._get_available_gpus()
```

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
! mkdir datasets
```

```
! mkdir originals
```

```
! unzip -q "/content/drive/My Drive/img_align_celeba.zip" -d datasets
```

```
! unzip -q "/content/drive/My Drive/coco_val2017.zip" -d datasets
```

```
! mv datasets/val2017/* originals
```

```
! find /content/datasets/img_align_celeba -type f -exec sh -
c 'mv "$@" "$@"' originals/ {} +
```

```
! pip install segmentation-models
```

```
import scipy
import tensorflow as tf
#from keras.datasets import mnist
from keras.layers import Input, Dense, Reshape, Flatten, Dropout, Concatenate
from keras.layers import BatchNormalization, Activation, ZeroPadding2D, Add, MaxP
ooling2D
from keras.layers.advanced_activations import PReLU, LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.applications import VGG19
from keras.models import Sequential, Model
import keras
#from tensorflow.keras.optimizers import Adam
import datetime
import matplotlib.pyplot as plt
import sys
import numpy as np
import os
from glob import glob
import keras.backend as K
#import scipy.misc
import PIL
from PIL import Image
import os
#import skimage.io as io
#import skimage.transform as trans
from keras.models import *
from keras.layers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras.applications.vgg19 import preprocess_input as vgg19_preprocess_input
from keras.optimizers import Adam
#import segmentation_models as sm

! pip install --upgrade pip
```

```
! python -m pip install --upgrade pip
```

```
! pip install --upgrade tensorflow
```

```
! pip install --upgrade keras
```

```
#import keras
```

```
import tensorflow as tf
```

```
#keras.__version__
```

```
tf.__version__
```

```
tf.core
```

```
! pip uninstall tensorflow
```

```
import keras
```

```
config = tf.ConfigProto(device_count = {'GPU': 1 , 'CPU': 3})
```

```
sess = tf.Session(config=config)
```

```
keras.backend.set_session(sess)
```

```
preprocess_input = sm.get_preprocessing('resnet34')
```

```
preprocess_input
```

```
class DataLoader():
```

```
 def __init__(self, img_res=(256, 256)):
```

```
 self.img_res = img_res
```

```
 def load_data(self, batch_size=1, is_testing=False):
```

```
 path = glob('./originals/*')
```

```
 batch_images = np.random.choice(path, size=batch_size)
```

```
 imgs_hr = []
```

```
 imgs_lr = []
```

```
 for img_path in batch_images:
```

```
 img_hr, img_lr = self._load(img_path, self.img_res)
```

```
 imgs_hr.append(img_hr)
```

```
 imgs_lr.append(img_lr)
```



```

нормализация данных
imgs_hr = np.array(imgs_hr) / 127.5 - 1
imgs_lr = np.array(imgs_lr) / 127.5 - 1

return imgs_hr, imgs_lr

returns pair (original photo, grayscale photo)
def _load(self, path, size):
 img = Image.open(path).resize(size).convert('RGB')
 orig = np.array(img).astype(np.float)
 import random
 crap_width = random.randint(100, 512)
 crap_size = (crap_width, crap_width)
 gray = np.expand_dims(np.array(img.resize(crap_size).resize(size).convert('L')
)).astype(np.float), axis=2)
 # np.expand_dims(np.array(Image.open(path).resize(size).convert('L')).astype(
np.float), axis=2)
 return orig, gray

hr_channels = 3
lr_channels = 1
width = 512
lr_shape = (width, width, lr_channels)
hr_shape = (width, width, hr_channels)

n_residual_blocks = 16
optimizer = Adam(0.0002, 0.5)

configure data loader
data_loader = DataLoader(img_res=(width, width))

def sample_images(epoch):
 os.makedirs('samples', exist_ok=True)
 r,c = 2,3
 imgs_hr, imgs_lr = data_loader.load_data(batch_size=2, is_testing=True)
 fake_hr = generator.predict(imgs_lr)
 imgs_lr = 0.5 * imgs_lr + 0.5
 fake_hr = 0.5 * fake_hr + 0.5

```

```

imgs_hr = 0.5 * imgs_hr + 0.5
titles = ["B&W", "Generated", "Original"]
fig, axs = plt.subplots(r, c)
cnt = 0
for row in range(r):
 for col, image in enumerate([imgs_lr, fake_hr, imgs_hr]):
 if col == 0:
 axs[row, col].imshow(np.array(Image.fromarray((np.squeeze(image[row])
*255).astype(np.uint8)).convert('RGB'))))
 else:
 axs[row, col].imshow(image[row])
 axs[row, col].set_title(titles[col])
 axs[row, col].axis('off')
 cnt += 1
def zerofy(s: str):
 while len(s) < 6:
 s = '0' + s
 return s
fig.savefig('samples/'+zerofy(str(epoch)) + '.png')
plt.close()

```

"""We use a pre-trained VGG19 model to extract image features from the high resolution and the generated high resolution images and minimize the mse between them"""

```

def build_feature_loss(input_size):
 vgg = VGG19(weights='imagenet')
 vgg.outputs = [vgg.layers[9].output]
 img = Input(shape=input_size)
 img_features = vgg(img)
 model = Model(img, img_features)
 model.trainable = False
 model.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
 return model
feature_loss = build_feature_loss(hr_shape)

```

"""build and compile the discriminator"""

```

def build_discriminator(input_size):

```

```

Number of filters in the first layer of G and D
gf = 64
df = 64
def d_block(layer_input, filters, strides=1, bn=True):
 """Discriminator layer"""
 d = Conv2D(filters, kernel_size=3, strides=strides, padding='same')(layer_input)
 d = LeakyReLU(alpha=0.2)(d)
 if bn:
 d = BatchNormalization(momentum=0.8)(d)
 return d

Input img
d0 = Input(shape=input_size)

d1 = d_block(d0, df, bn=False)
d2 = d_block(d1, df, strides=2)
d3 = d_block(d2, df*2)
d4 = d_block(d3, df*2, strides=2)
d5 = d_block(d4, df*4)
d6 = d_block(d5, df*4, strides=2)
d7 = d_block(d6, df*8)
d8 = d_block(d7, df*8, strides=2)

d9 = Dense(df*16)(d8)
d10 = LeakyReLU(alpha=0.2)(d9)
validity = Dense(1, activation='sigmoid')(d10)

model = Model(d0, validity)
model.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])
return model

discriminator = build_discriminator(input_size=hr_shape)

"""Build the generator"""

def unet(pretrained_weights = None, input_size = (256, 256, 3)):
 inputs = Input(input_size)

```

```

conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(inputs)
conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(pool1)
conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(pool2)
conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(pool3)
conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv4)
drop4 = Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializ
er = 'he_normal')(pool4)
conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializ
er = 'he_normal')(conv5)
drop5 = Dropout(0.5)(conv5)

up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(drop5))
merge6 = concatenate([drop4,up6], axis = 3)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(merge6)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv6)

up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(conv6))
merge7 = concatenate([conv3,up7], axis = 3)

```

```

conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(merge7)
conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv7)

up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(UpSampling2D(size = (2,2))(conv7))
merge8 = concatenate([conv2,up8], axis = 3)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(merge8)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initialize
r = 'he_normal')(conv8)

up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv8))
merge9 = concatenate([conv1,up9], axis = 3)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(merge9)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv9)
conv9 = Conv2D(3, 3, activation = 'relu', padding = 'same', kernel_initializer
= 'he_normal')(conv9)
conv10 = Conv2D(3, 1, activation = 'tanh')(conv9)

model = Model(input = inputs, output = conv10)

#model.summary()

if(pretrained_weights):
 model.load_weights(pretrained_weights)
model.compile(optimizer = Adam(lr = 1e-
4), loss = 'mse', metrics = ['accuracy'])
return model

generator = unet(input_size=lr_shape)
#generator = sm.Unet('resnet34', input_shape=(width, width, 3), encoder_weights='
imagenet', classes=3, activation="tanh", encoder_freeze=True,decoder_filters=(102
4, 512,256, 128, 64))
#generator.compile(optimizer = Adam(lr = 1e-
4), loss = 'mse', metrics = ['accuracy'])

```

```
generator.summary()
```

```
def build_gan(input_size):
 # High res. and low res. images
 img_lr = Input(shape=lr_shape)
 # generate high res. version from low res.
 fake_hr = generator(img_lr)
 # extract image features of the generated img
 fake_features = feature_loss(fake_hr)
 # for the combined model we will only train the generator
 discriminator.trainable = False
 # Discriminator determines validity of generated high res. images
 validity = discriminator(fake_hr)
 combined = Model([img_lr], [validity, fake_features])
 combined.compile(loss=['binary_crossentropy', 'mse'], loss_weights=[1e-
3, 1], optimizer=optimizer)
 return combined
gan = build_gan(lr_shape)
```

```
"""## Train"""
```

```
#! rm -rf ./samples
epochs=100000
batch_size=1
sample_interval=50
start_time = datetime.datetime.now()
for epoch in range(epochs+1):
 # calculate output shape of D (PatchGAN)
 patch = int(width / 2**4)
 disc_patch = (patch, patch, 1)
 # -----
 # Train Discriminator
 # -----

 # Sample images and their conditioning counterparts
 imgs_hr, imgs_lr = data_loader.load_data(batch_size)

 # From low res. image generate high res. version
```

```

fake_hr = generator.predict(imgs_lr)

valid = np.ones((batch_size,) + disc_patch)
fake = np.zeros((batch_size,) + disc_patch)

Train the discriminators (original images = real / generated = Fake)
d_loss_real = discriminator.train_on_batch(imgs_hr, valid)
d_loss_fake = discriminator.train_on_batch(fake_hr, fake)
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

Train Generator

Sample images and their conditioning counterparts
imgs_hr, imgs_lr = data_loader.load_data(batch_size)

The generators want the discriminators to label the generated images as real
1
valid = np.ones((batch_size,) + disc_patch)

Extract ground truth image features using pre-trained VGG19 model
image_features = feature_loss.predict(imgs_hr)


Train the generators
g_loss = gan.train_on_batch([imgs_lr], [valid, image_features])

elapsed_time = datetime.datetime.now() - start_time
Plot the progress
print ("%d time: %s" % (epoch, elapsed_time))

If at save interval => save generated image samples
if epoch % sample_interval == 0:
 sample_images(epoch)
if epoch % 1000 == 0:
 save_generator = keras.Sequential([generator])
 save_generator.add(Lambda(lambda x: (x + 1) * 127.5))
 save_generator.save('generator512_2' + str(epoch) + '.h5')

```

```

img = Image.open('test9.jpg').resize((500, 500)).resize((512, 512)).resize((512,
512))
t = np.expand_dims(np.array(img.convert('L')).astype(np.float), axis=2)
a = np.array([t]) / 127.5 - 1
res = generator.predict(a)
res = 0.5 * res + 0.5
res = res[0]
Image.fromarray(np.uint8(res * 255))


```

```

img = Image.open('test7.jpg').resize((400, 400)).resize((512, 512))
img

```

```

import random

```

```

random.randint(100, 512)

```

```

! pip install --upgrade tensorflow==2.2

```

```

import tensorflow as tf
tf.__version__

```

```

generator.summary()

```

```

save_generator = keras.Sequential([generator])
save_generator.add(Lambda(lambda x: (x + 1) * 127.5))
save_generator.save('generator512_2.h5')

```

```

"""## Convert to CoreML"""

```

```

np.array(Image.open('test.jpg').convert('L')).max()

```

```

! pip install coremltools

```

```

! pip install keras==2.2.4

```

```

! pip install tensorflow==1.14.0

```

```

from coremltools.proto import NeuralNetwork_pb2

```



```

def convert_lambda(layer):
 params = NeuralNetwork_pb2.CustomLayerParams()

 # The name of the Swift or Obj-C class that implements this layer.
 params.className = "Lambda"

 # The description is shown in Xcode's mlmodel viewer.
 params.description = "Post process"

 return params

import coremltools
coreml_model = coremltools.converters.keras.convert('generator.h5', input_names='
input', output_names='output', image_input_names='input', add_custom_layers=True,
 image_scale=2/255, gray_bias=-1, red_bias=-
1, custom_conversion_functions={ "Lambda": convert_lambda }, use_float_arraytype=
True)

Saving the Core ML model to a file.
coreml_model.save('colorizer.mlmodel')

import coremltools
import coremltools.proto.FeatureTypes_pb2 as ft

spec = coremltools.utils.load_spec("colorizer.mlmodel")

output = spec.description.output[0]
width = 512
import coremltools.proto.FeatureTypes_pb2 as ft
output.type.imageType.colorSpace = ft.ImageFeatureType.RGB
output.type.imageType.height = width
output.type.imageType.width = width

coremltools.utils.save_spec(spec, "Colorizer.mlmodel")

#! pip uninstall keras
#! pip uninstall tensorflow
! pip install tensorflow

```

```
! pip install keras
```

## ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ

### ViewController.swift

```
//
// ViewController.swift
// colorizer
//
// Created by Максим Ефимов on 25.05.2020.
// Copyright © 2020 Максим Ефимов. All rights reserved.
//
import UIKit
import CoreML
import Vision

class ViewController: UIViewController, UIImagePickerControllerDelegate, UINavigationControllerDelegate {

 @IBOutlet weak var promoImage: UIImageView!
 @IBOutlet weak var startButton: UIButton!
 let picker = UIImagePickerController()
 let model = try! VNCoreMLModel(for: Colorizer().model)
 @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
 var n: Int = 0
 var m: Int = 0

 override func viewDidLoad() {
 super.viewDidLoad()
 self.navigationController?.navigationBar.setValue(true, forKey: "hidesShadow")
 startButton.layer.masksToBounds = true
 startButton.layer.cornerRadius = 6
 picker.sourceType = .photoLibrary
 picker.delegate = self
 }

 override func viewDidLayoutSubviews() {
 super.viewDidLayoutSubviews()
 }
}
```

```

@IBAction func onStart(_ sender: UIButton) {
 self.present(picker, animated: true)
}

func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
 picker.dismiss(animated: true) {
 guard let image = info[.originalImage] as? UIImage else {
 fatalError("Expected a dictionary containing an image, but was provided the following: \(info)")
 }
 let img = image
 self.process(img)
 }
}

func process(_ image: UIImage) {
 activityIndicator.startAnimating()
 startButton.setTitle("", for: .normal)
 startButton.isEnabled = false
 let request = VNCoreMLRequest(model: model) { (request, error) in
 self.processResult(for: request, error: error, originalImage: image)
 }
 request.imageCropAndScaleOption = .scaleFill
 n = Int(max(1, round(image.size.height / 512)))
 m = Int(max(1, round(image.size.width / 512)))
 m = min(m, 2)
 n = min(n, 2)
 let resizedImage = image.resized(newSize: CGSize(width: 512 * m, height: 512 * n))
 DispatchQueue.global(qos: .userInitiated).async {
 let cgImage = resizedImage.cgImage!
 //CGImagePropertyOrientation(
 //let orientation = CGImagePropertyOrientation(image.imageOrientation)
)

 var imagesMatrix: [[CGImage]] = []
 for i in 0..

```

```

 let cropped = cgImage.cropping(to: CGRect(x: j * 512, y: i *
512, width: 512, height: 512))!
 images.append(cropped)
 }
 imagesMatrix.append(images)
}

let orientation = CGImagePropertyOrientation(image.imageOrientation)
for images in imagesMatrix {
 for img in images {
 let handler = VNImageRequestHandler(cgImage: img, orientation
: orientation)
 do {
 try handler.perform([request])
 } catch {
 print("Failed to perform classification.\n\(error.localiz
edDescription)")
 }
 }
}
}
}

var images: [UIImage] = []

func processResult(for request: VNRequest, error: Error?, originalImage: UIIm
age) {
 print(#function)
 DispatchQueue.main.async {

 guard let results = request.results as? [VNPixelBufferObservation] el
se {

 print("error", error ?? "")
 return
 }
 let ciImage = CIImage(cvPixelBuffer: results[0].pixelBuffer)
 let cgImage = self.convertCIImageToCGImage(inputImage: ciImage)!
 let uiImage = UIImage(cgImage: cgImage)
 self.images.append(uiImage)
 }
}

```

```

 if self.images.count == self.n * self.m {
 var result = self.merge()
 result = result.resized(newSize: originalImage.size)
 self.move(image: result, originalImage: originalImage)
 self.images = []
 }
 }
}

func move(image: UIImage, originalImage: UIImage) {
 let storyboard = UIStoryboard(name: "Main", bundle: nil)
 let newViewController = storyboard.instantiateViewController(withIdentifier: "SaveViewController") as! SaveViewController
 newViewController.image = image
 newViewController.originalImage = originalImage
 self.show(newViewController, sender: self)

 self.activityIndicator.stopAnimating()
 self.startButton.setTitle("Let's start", for: .normal)
 self.startButton.isEnabled = true
}

}

extension ViewController {
 func convertCIImageToCGImage(inputImage: CIImage) -> CGImage? {
 let context = CIContext(options: nil)
 if let cgImage = context.createCGImage(inputImage, from: inputImage.extent) {
 return cgImage
 }
 return nil
 }

 func merge() -> UIImage {
 // This is the rect that we've calculated out and this is what is actually used below

 // Actually do the resizing to the rect using the ImageContext stuff

```

```

 UIGraphicsBeginImageContextWithOptions(CGSize(width: 512 * m, height: 512
* n), false, 1.0)
 for (i, image) in images.enumerated() {
 let rect = CGRect(x: i % m * 512, y: i / m * 512, width: 512, height:
512)

 image.draw(in: rect)
 }
 let newImage = UIGraphicsGetImageFromCurrentImageContext()
 UIGraphicsEndImageContext()
 return newImage!
 }
}

extension UIImage {
 func resized(newSize: CGSize) -> UIImage {
 // This is the rect that we've calculated out and this is what is actually
used below
 let rect = CGRect(x: 0, y: 0, width: newSize.width, height: newSize.height)

 // Actually do the resizing to the rect using the ImageContext stuff
 UIGraphicsBeginImageContextWithOptions(newSize, false, 1.0)
 self.draw(in: rect)
 let newImage = UIGraphicsGetImageFromCurrentImageContext()
 UIGraphicsEndImageContext()
 return newImage!
 }
}

```

## SaveViewController.swift

```

//
// SaveViewController.swift
// colorizer
//
// Created by Максим Ефимов on 26.05.2020.
// Copyright © 2020 Максим Ефимов. All rights reserved.
//
import UIKit
import StoreKit

```

```

class SaveViewController: UIViewController {
 var image: UIImage!
 var originalImage: UIImage!
 @IBOutlet weak var imageView: UIImageView!
 @IBOutlet weak var saveButton: UIButton!
 override func viewDidLoad() {
 super.viewDidLoad()
 self.imageView.contentMode = .scaleAspectFit
 self.imageView.image = image
 self.navigationController?.navigationBar.setBackgroundImage(UIImage(), for: .default)
 self.navigationController?.navigationBar.shadowImage = UIImage()
 self.navigationController?.navigationBar.isTranslucent = true
 self.navigationController?.navigationBar.backgroundColor = .clear
 self.navigationController?.view.backgroundColor = .clear

 let backButton = UIBarButtonItem()
 backButton.title = ""
 self.navigationController?.navigationBar.topItem?.backBarButtonItem = backButton

 saveButton.layer.masksToBounds = true
 saveButton.layer.cornerRadius = 6
 imageView.isUserInteractionEnabled = true
 imageView.addGestureRecognizer(UITapGestureRecognizer(target: self, action: #selector(imageTap)))
 }

 @IBAction func saveButtonAction(_ sender: Any) {
 let imageShare = [image!]
 let activityViewController = UIActivityViewController(activityItems: imageShare, applicationActivities: [])
 activityViewController.completionWithItemsHandler = {(activityType: UIActivity.ActivityType?, completed: Bool, returnedItems: [Any]?, error: Error?) in
 if !completed {
 return
 }
 let ac = UIAlertController(title: "Saved!", message: "Your colored photo has been saved.", preferredStyle: .alert)

```



```

 ac.addAction(UIAlertAction(title: "OK", style: .default))
 self.present(ac, animated: true)
 }
 activityViewController.popoverPresentationController?.sourceView = saveButton
 self.present(activityViewController, animated: true)
 //UIImageWriteToSavedPhotosAlbum(image, self, #selector(image(_:didFinishSavingWithError:contextInfo:)), nil)
}

override func viewDidLoad(_ animated: Bool) {
 super.viewDidLoad(animated)
 if Int.random(in: 0..<4) == 2 {
 SKStoreReviewController.requestReview()
 }
}

@objc func imageTap() {
 if self.imageView.image == image {
 self.imageView.image = originalImage
 } else {
 self.imageView.image = image
 }
}
}

```

## CGImagePropertyOrientation.swift

```

/*
Copyright (c) 2017-2019 M.I. Hollemans
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

```

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

\*/

#if canImport(Uikit)

import UIKit

```
public extension CGImagePropertyOrientation {
 init(_ orientation: UIImage.Orientation) {
 switch orientation {
 case .up: self = .up
 case .upMirrored: self = .upMirrored
 case .down: self = .down
 case .downMirrored: self = .downMirrored
 case .left: self = .left
 case .leftMirrored: self = .leftMirrored
 case .right: self = .right
 case .rightMirrored: self = .rightMirrored
 @unknown default: self = .up
 }
 }
}
```

#if !os(tvOS)

```
public extension CGImagePropertyOrientation {
 init(_ orientation: UIDeviceOrientation) {
 switch orientation {
 case .portraitUpsideDown: self = .left
 case .landscapeLeft: self = .up
 case .landscapeRight: self = .down
 default: self = .right
 }
 }
}
```

```
#endif
```

```
extension UIImage.Orientation {
 init(_ cgOrientation: UIImage.Orientation) {
 switch cgOrientation {
 case .up: self = .up
 case .upMirrored: self = .upMirrored
 case .down: self = .down
 case .downMirrored: self = .downMirrored
 case .left: self = .left
 case .leftMirrored: self = .leftMirrored
 case .right: self = .right
 case .rightMirrored: self = .rightMirrored
 @unknown default: self = .up
 }
 }
}
```

```
#endif
```