МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Разработка информационных систем

Доклад по теме:

«Построение нейронных сетей для раскраски черно-белых изображений»

Студент: Хахин М.С.

Группа: М8О-404Б-18

Преподаватель: Виноградов В.И.

Дата: 02.12.2021

1. Постановка задачи

Задача: обучить нейронную сеть переносить стиль с одного изображения на второе

Решение задачи: изучить теорию по нейронным сетям, изучить библиотекуTensorflow и слои из библиотеки Keras и правильно применить знания

2. Теория

**Свёрточная нейронная сеть** (CNN)— специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 годуи нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения. Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв и субдискретизирующих слоёв (слоёв подвыборки). Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

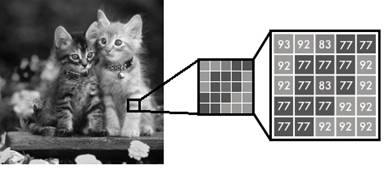
Как с помощью СНС делается раскраска изображений (из градаций серого в цветное)?



По-английски, это называется:

Colorization

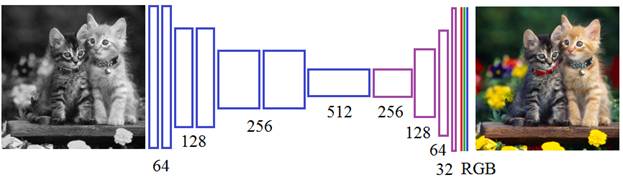
Начнем с того, чем отличается на уровне пикселей цветное изображение от черно-белого (градации серого для простоты буду называть черно-белым). Черно-белое представлено одной компонентой яркости со значениями пикселей в диапазоне от 0 до 255:



Полноцветные, как правило, описываются тремя компонентами RGB:



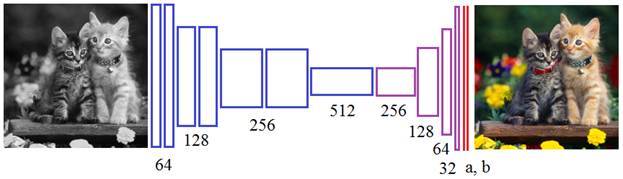
Следовательно, чтобы СНС раскрасила черно-белое изображение, на ее выходе должно формироваться три цветовых канала:



Но это не лучшее решение. Полноцветные изображения в данной задаче удобнее представлять в другом цветовом пространстве, например:

Lab: (Light – слой яркости (градации серого); a, b – цветовые слои)

Причем, пиксели слоя L меняются в диапазоне от 0 до 100, а пиксели слоев a и b – в диапазоне от -128 до 127. В этом случае СНС достаточно сгенерировать только для канала: a и b, вместо трех RGB (третий канал у нас уже есть – это исходное изображение в градациях серого):



Кроме того, 94% рецепторов человеческого глаза настроены на восприятие яркости (компоненты L – градации серого) и только 6% - на цветовые составляющие. Поэтому, если нейросеть немного «напутает» в цветах – это не так сильно скажется на визуальном восприятии картинки в целом. А вот путаница в яркостной составляющей – это уже критично. Но мы ее и не будем формировать, а возьмем уже готовую. Это еще одно преимущество цветового пространства Lab в рамках данной задачи.

3. Решение задачи

Для преобразования изображения из RGB в Lab в Python воспользуемся пакетом skimage и импортируем следующие методы:

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** skimage.color **import** rgb2lab, lab2rgb

* rgb2lab – преобразовывает изображение из RGB в Lab;
* lab2rgb – преобразовывает изображение из Lab в RGB.

В качестве тестовой реализации мы можем взять любое полноцветное изображение, из него получить яркостную компоненту L (в градациях серого) и две цветовые компоненты a и b. На вход сети подадим изображение в градациях серого, а на выходе потребуем цветовые составляющие a и b:

Для загрузки изображения в коллаборатории google импортируем следующие модули:

**from** google.colab **import** files

**from** io **import** BytesIO

**from** PIL **import** Image

И, затем, выполним строчки:

upl = files.upload()

names = list(upl.keys())

img = Image.open(BytesIO(upl[names[0]]))

Загруженное изображение будет в RGB-формате. Для преобразования в пространство Lab определим следующую функцию:

**def** processed\_image(img):

  image = img.resize( (256, 256), Image.BILINEAR)

  image = np.array(image, dtype=float)

  size = image.shape

  lab = rgb2lab(1.0/255\*image)

  X, Y = lab[:,:,0], lab[:,:,1:]

  Y /= 128    *# нормируем выходные значение в диапазон от -1 до 1*

  X = X.reshape(1, size[0], size[1], 1)

  Y = Y.reshape(1, size[0], size[1], 2)

**return** X, Y, size

Мы здесь сначала изменяем размер изображения до 256х256 пикселей и преобразовываем его в массив numpy. Затем, делаем преобразование в пространство Lab (обратите внимание, на вход функции нужно передавать изображение с компонентами RGB и вещественными значениями пикселей от 0 до 1, поэтому мы здесь добавляем нормирующий множитель 1/255). Далее, выделяем яркостную компоненту X и две цветовые в Y. Цвета будут использоваться как требуемые выходные значения НС, поэтому мы их нормируем до диапазона [-1; 1]. Затем, формируем нужный формат размерностей для входных и выходных данных НС.

Вызовем эту функцию и получим следующий набор данных:

X, Y, size = processed\_image(img)

Теперь у нас есть что подавать на сеть и что требовать на ее выходах. Поэтому дальше нам нужно построить модель СНС, представленной ранее на рисунке. С помощью Keras это можно сделать так:

**from** keras.layers **import** Conv2D, UpSampling2D, InputLayer

**from** keras.models **import** Sequential

model = Sequential()

model.add(InputLayer(input\_shape=(None, None, 1)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same', strides=2))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same', strides=2))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same', strides=2))

model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))

model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))

model.add(UpSampling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

model.add(UpSampling2D((2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))

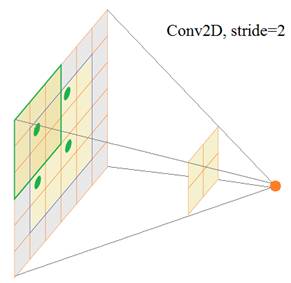
model.add(Conv2D(2, (3, 3), activation='tanh', padding='same'))

model.add(UpSampling2D((2, 2)))

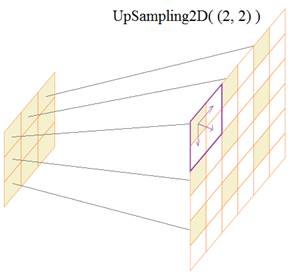
Чтобы не изобретать велосипед, я взял эту модель из статьи, посвященной теме раскраски изображений с помощью глубоких нейронных сетей:

<https://github.com/baldassarreFe/deep-koalarization>

Обратите внимание на структуру слоев. Первый слой – это обычный сверточный слой, состоящий из 64 фильтров и ядрами 3х3 пиксела. Следующая свертка имеет те же параметры, но шаг смещения фильтров равен двум пикселям по каждой координате. Почему здесь масштабирование признаков делается с помощью увеличения шага, а не методом MaxPooling который использовался при классификации и стилизации изображений? Дело в том, что слой MaxPooling хорошо концентрирует значимую информацию об особенностях изображения, но несколько искажает взаимное расположение пикселей на плоскости. В задачах колоризации такое искажение представления изображения недопустимо. Поэтому и используется сверточный слой с шагом 2.



Так продолжается движение в глубину, пока не встретится слой UpSampling2D((2, 2)). Параметр (2, 2) задает увеличение размера каждого элемента карты признаков. В Keras это работает следующим образом:



Каждый элемент карты признаков увеличивается до указанного размера (2, 2), причем при масштабировании каждой ячейки (синяя рамка) значение просто копируется в соседние, заполняя все свое пространство. И так каждый элемент. В результате получается увеличенное грубое представление карт признаков на каждом канале.

На последнем выходном слое имеем два канала (для двух цветовых компонент a и b), размеры которых совпадают с размерами исходного (входного) изображения. В качестве функции активации выбираем гиперболический тангенс, чтобы цветовые составляющие имели диапазон [-1; 1]. Так сеть будет формировать цвета.

Давайте для примера обучим эту НС на одном изображении, то есть, потребуем, чтобы она выдавала строго определенные выходные цветовые компоненты:

model.compile(optimizer='adam', loss='mse')

model.fit(x=X, y=Y, batch\_size=1, epochs=50)

Мы здесь делаем оптимизацию по Adam, критерий качества – минимум среднего квадрата рассогласования. На вход этой сети будем подавать черно-белое изображение, а на выходе требовать заданные для него цветовые составляющие.

После обучения прогоним через сеть изображение в градациях серого:

output = model.predict(X)

и посмотрим на результат:

output \*= 128

min\_vals, max\_vals = -128, 127

ab = np.clip(output[0], min\_vals, max\_vals)

cur = np.zeros((size[0], size[1], 3))

cur[:,:,0] = np.clip(X[0][:,:,0], 0, 100)

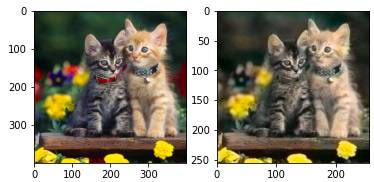
cur[:,:,1:] = ab

plt.subplot(1, 2, 1)

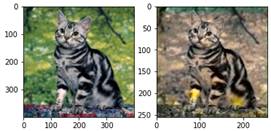
plt.imshow(img)

plt.subplot(1, 2, 2)

plt.imshow(lab2rgb(cur))

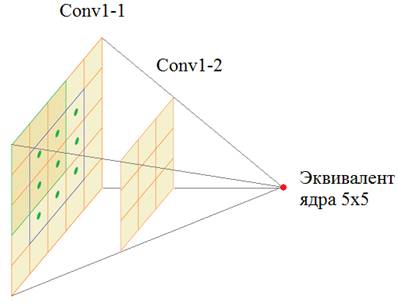


Как видите, это одно конкретное изображение она раскрасила вполне приемлемо, правда, мы именно на нем ее и обучали. Но этот простейший пример показывает, что такая операция, в принципе, возможна и общая идея вроде бы рабочая. Но если взять другое изображение, то эффект будет уже значительно хуже:

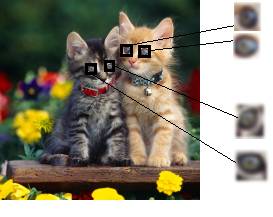


И это не удивительно. Наша обучающая выборка состояла всего из одного изображения. Этого явно недостаточно. Нужно хотя бы несколько тысяч. Будет ли такое обучение?

А мы зададимся вопросом: почему НС в принципе способна выполнять раскраску, как это работает? Давайте посмотрим еще раз на структуру НС. Здесь первые два слоя образуют свертки с ядром 3х3. Это приближенно заменяет свертку фильтра с ядром 5х5:



Получается, что каждый признак парных слоев связан с областью 5х5 отсчетов. И если в эту область часто попадают округлые очертания объекта темного объекта (например, глаза котов), то сеть связывает такой признак с темным цветом:



Это, особенно хорошо проявляется на глубоких слоях, где формируются более сложные объекты: глаза, лапы, цветок, трава и т.п. В весовых коэффициентах, как бы, сохраняется опыт и «знания» о соответствии элементов черно-белого изображения этим же элементам, но в цвете. Примерно так можно воспринимать работу НС по раскраске изображений.

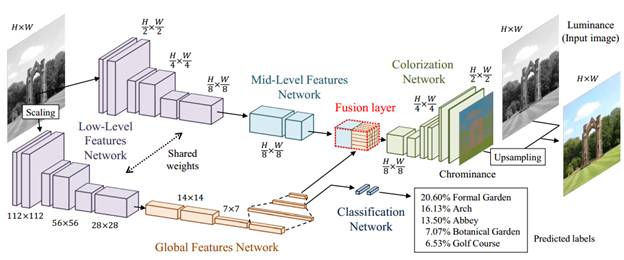
Приведенный алгоритм колоризации изображений может неплохо работать на однотипных данных, например, котов на фоне природы, или лица людей, или фотографий морских пляжей и так далее. Но если все это смешать, то результаты станут заметно хуже.

В 2016-м году ряд японских исследователей:

Хатоши Иизука, Эдгар Симо-Серра и Хироши Ишикава

<http://hi.cs.waseda.ac.jp/~iizuka/projects/colorization/data/colorization_sig2016.pdf>

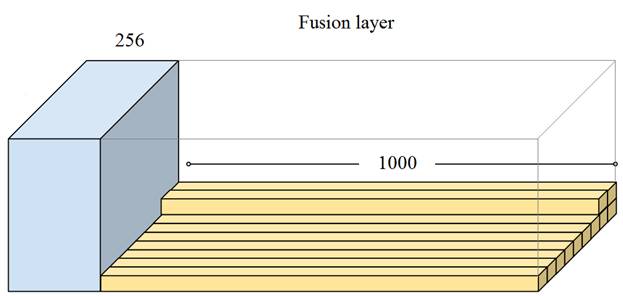
предложили интересную концепцию по улучшению раскраски изображений. Они к уже существующей НС параллельно добавили еще одну, которая выполняет обычную классификацию, то есть, определяет: к какому классу относится раскрашиваемое изображение. Как вы понимаете, если наша НС будет дополнительно «знать» о типе входных данных (изображение котов, пляжа, руин, лиц, машин и т.п.), то она сможет сохранить специализацию по раскраске и при этом работать с любыми данными.



Для определения характера изображения ученые предложили воспользоваться одной из известных и уже обученных СНС, например, знакомой нам VGG19, которая на выходе дает 1000 различных классов. Или же, можно взять более продвинутую сеть

Inception-ResNet-v2

которая также имеет 1000 выходных классов. На рисунке, который я взял из статьи, показано как добавляется классифицирующая сеть к раскрашивающей сети. Для этого японские ученые добавили еще один, так называемый слой слияния (Fusion layer). Что это за слой? В центре первой сверточной сети (с наименьшими размерами карт признаков) создается дополнительный сверточный слой с 256 каналами и таким же размером карт признаков, что и в предыдущем слое. Затем, признаки дополняются вектором классификации от второй СНС, по следующей схеме:



Благодаря этому, каждый признак в дальнейшем будет ассоциирован с установленной тематикой входного изображения и качество раскраски значительно увеличивается.

4. Вывод

В данной работе я продемонстрировал работу СНС и разобрался с задачей, которая была поставлена. С появлением СНС в области «нейронные сети» произошли глобальные изменения. СНС – это один из мощнейших инструментов для работы с изображений в сфере нейронных сетей. Они позволяют не только раскрашивать изображений, так же они могут увеличивать разрешение и качество изображений