Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Обработка больших объемов информации

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ 3

по предмету «Интеллектуальный анализ информации»

Студент
гр. 758601
Лимонтов А. С.

Проверил
Ивашенко В. П.

Постановка задачи **2**

Используемые библиотеки **2**

Keras 2

OpenCV 2

Этапы реализации программы **2**

Исходные данные 2

Обработка изображения 3

Класс автоенкодера 4

Тестирование модели 5

Выводы **8**

Литература **9**

Постановка задачи

Реализация линейной рециркуляционной сети как модели автодекодера для задачи понижения размерности данных с использованием библиотеки Keras.

Используемые библиотеки

Keras

Как было продемонстрировано в предыдущей лабораторной работе, Keras является хорошим инструментом для использования в задачах Deep Learning. С помощью него легко управлять архитектурой сети, определять прогресс на этапах обучения. Для этой библиотеки выполним необходимые импорты:

OpenCV

OpenCV позволяет легко выполнять обработку изображений (проще реализовывать преобразования над пикселами). Больше деталей можно найти далее в исходном коде.

Этапы реализации программы

Исходные данные

Импорты используемых библиотек:

```
import numpy as np
import random
from matplotlib.pyplot import imshow
from functools import partial
import matplotlib.pyplot as plt

from keras.layers import Input, Dense
from keras.models import Model
from keras.callbacks import Callback
from keras.losses import mean_squared_error
from keras import backend as K
from keras import optimizers

import cv2

%matplotlib inline
```

Задание глобальных параметров:

- `SEED`: для случайных генераторов с целью воспроизводимости
- `MODE`: режим работы с картинкой RGB/L - цветной/черно-белый
- `S`: количество аргументов на пиксель (3** - для цветного изображения, **1 - для черно-белого)
- `IMG_PATH`: путь к обрабатываемому изображению

```
SEED = 342323425
np.random.seed(SEED)
MODE = 'L'
S = 1
CROP = 16
SIZE = 256
IMG_PATH = './img.jpg'
```

Метод загрузки картинки и преобразования в необходимый режим и размер:

```
def load_image(path):
    img = cv2.imread(path, 0)
    img = cv2.resize(img, (SIZE, SIZE))
    return img
```

Обработка изображения

Методы преобразования картинки, генераторы нарезаемых блоков. Картинка равномерно нарезается на блоки последовательно слева-направо и сверху-вниз на размеры nxm. При декодировании следует помнить о том, что в результате должны получиться

- целый значения
- числа в пределах [0..255][0..255]

Поэтому делается `np.clip(np.int_(C_MAX * (area + 1.0) / 2.0), 0, 255`:

```
def transform_image(img):
```

```
    C_MAX = 255.0
           return 2.0 * img.reshape([SIZE, SIZE // CROP,
CROP]).transpose(1, 0, 2).reshape((SIZE // CROP) ** 2, CROP, CROP)
/ C_MAX - 1.0
```

Класс автоенкодера

Класс отслеживания статуса выполнения обучения. В нем переопределен лишь метод `on_epoch_end`, который вызывается после прогона очередной эпохи. В нем определяется ошибка с помощью `np.linalg.norm`, а также обновляются общие результаты запуска модели:

```
class TrackStatus(Callback):

    def __init__(self, encoder, img):
        self.encoder = encoder
        self.img = img

    def on_epoch_end(self, epoch, logs={}):
        y = self.encoder.run_compression(self.img)
        current_error = np.linalg.norm(self.img - y)
        print('Epoch: {} Error: {}'.format(epoch, current_error))
        self.encoder.results['min_err'] =
np.min([self.encoder.results.get('min_err', 100000000000.0),
current_error])
        self.encoder.results['epoch'] = epoch + 1

        if current_error < self.encoder.min_error:
            self.model.stop_training = True
```

Класс автоенкодера. С помощью функций из Keras `Input`, `Dense` создаются слои в модели, количество нейронов в них задается соответствующими параметрами. Функции активации линейные. В качестве оптимизатора был выбран `Adam`, поскольку он показал себя лучше остальных (сходимость быстрее), а в качестве функции для оптимизации была выбрана `mean_squared_error` из Keras. Функция run_compression выполняет преобразование картинки (то есть вызывает `model.predict()`), а затем приводит полученные значения обратно в пиксельные значения для отображения получившейся картинки.

```
class Autoencoder(object):

    def __init__(self, input_layers=N, lr=1e-3, dynamic_lr=True,
z=0.9, max_epochs=1000, min_error=1000.0):
        self.input_layers = input_layers
        self.mid_layers = int(z * input_layers)
        self.initializer = glorot_uniform(self.input_layers,
self.mid_layers)
        self.lr = lr
```

```
        self.dynamic_lr = dynamic_lr
        self.max_epochs = max_epochs
        self.min_error = min_error
        self.results = {}
        self.build()

    def build(self):
        input_img = Input(shape=[self.input_layers])
        encoded = Dense(self.mid_layers, activation='linear',
kernel_initializer='glorot_uniform')(input_img)
        decoded = Dense(self.input_layers, activation='linear',
kernel_initializer='glorot_uniform')(encoded)

        self.autoencoder = Model(input_img, decoded)
        self.optimizer = optimizers.Adam(lr=self.lr)
        self.autoencoder.compile(optimizer=self.optimizer,
loss=mean_squared_error)

    def __call__(self, img, pbar=None):
        X = self.map_X(img)
        callbacks = [TrackStatus(self, img)]
        history = self.autoencoder.fit(X, X,
epochs=self.max_epochs,
                                       batch_size=128,
                                       shuffle=True,
validation_split=0.2, verbose=-1,
                                       callbacks=callbacks)
        return history

    def map_X(self, img):
        tr_img = transform_image(img)
        return np.asarray(list(map(lambda x:
np.expand_dims(x.flatten(), 0),
tr_img))).reshape((self.input_layers, -1)).T

    def run_compression(self, img):
        X = self.map_X(img)
        y = self.autoencoder.predict(X)
        y = (y.T.reshape(-1, CROP, CROP).reshape(SIZE // CROP,
SIZE, CROP).transpose(1, 0, 2).reshape(SIZE, SIZE) + 1.0) * 255 /
2
        return np.clip(y, 0, 255)
```

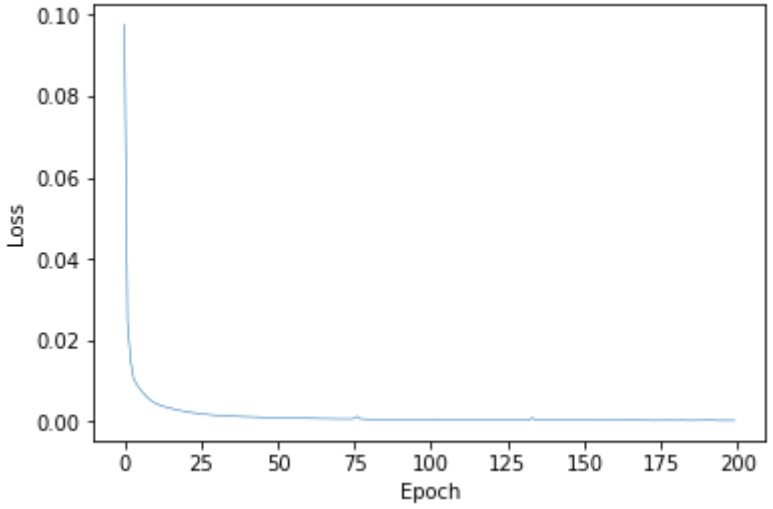
Тестирование модели

Функция для запуска выбранной модели и построения графика loss и полученной картинки:

```
def try_run(model, img=load_image(IMG_PATH), viz=True):
    history = model(img)
    err = history.history['loss']

    if viz:
                    plt.plot(range(len(err)),  err,  linewidth=0.5,
markersize=3)
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.show()

        res = model.run_compression(img)
        fig, ax = plt.subplots(1, 2, figsize=(16, 16))
        ax[0].imshow(img, cmap='gray')
        ax[0].set_xlabel('original')
        ax[1].imshow(res, cmap='gray')
        ax[1].set_xlabel('modified')
        plt.show()

    return model.results
```

Проверим на тестовом запуске:

```
model   =   Autoencoder(input_layers=N,   z=0.2,   max_epochs=200,
min_error=100.0)
try_run(model)
```



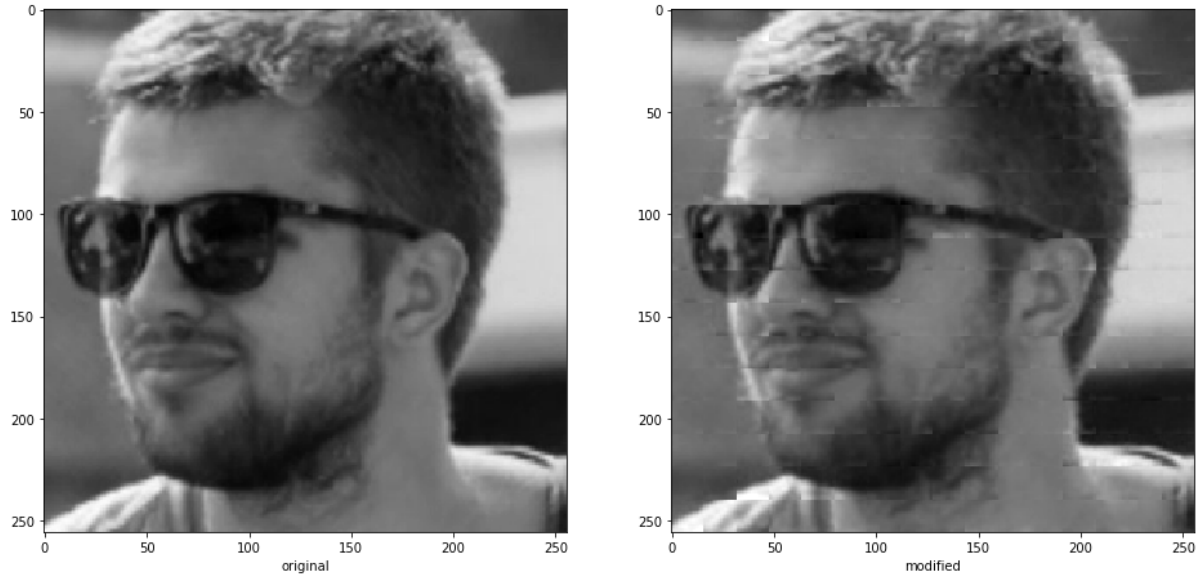


График зависимости количества эпох от коэффициента сжатия `Z` (с кодом):

```
N_z = 10
zs = np.linspace(0.05, 0.95, N_z)
MAX_EPOCHS = 200
TH_ERR = 2000.0

epochs = np.array([])
for z in zs[::-1]:
    print('running for z={}'.format(z))
                model    =    Autoencoder(input_layers=N,    z=z,
max_epochs=MAX_EPOCHS, min_error=TH_ERR)
    best_results = try_run(model)
    epochs = np.append(epochs, best_results['epoch'])
    print('Error:', best_results['min_err'])

plt.plot(zs[::-1], epochs)
plt.title('Epochs to achieve error {}'.format(TH_ERR))
plt.xlabel('Z')
plt.ylabel('Epochs')
plt.show()
```

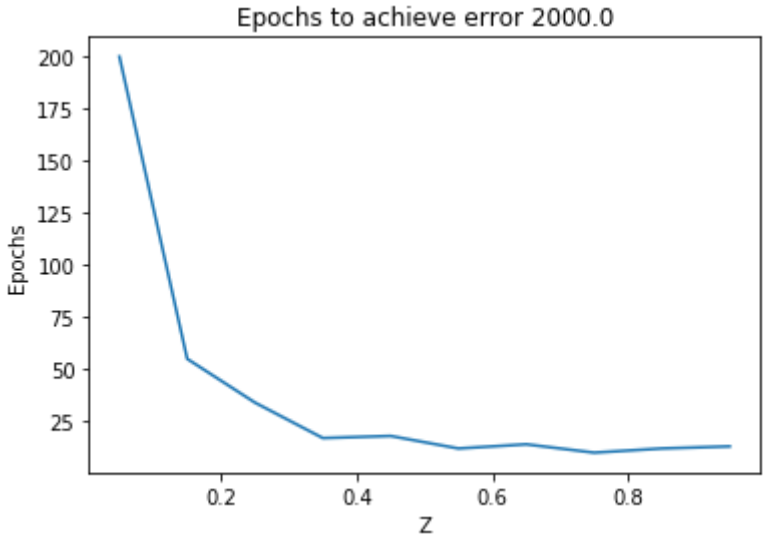
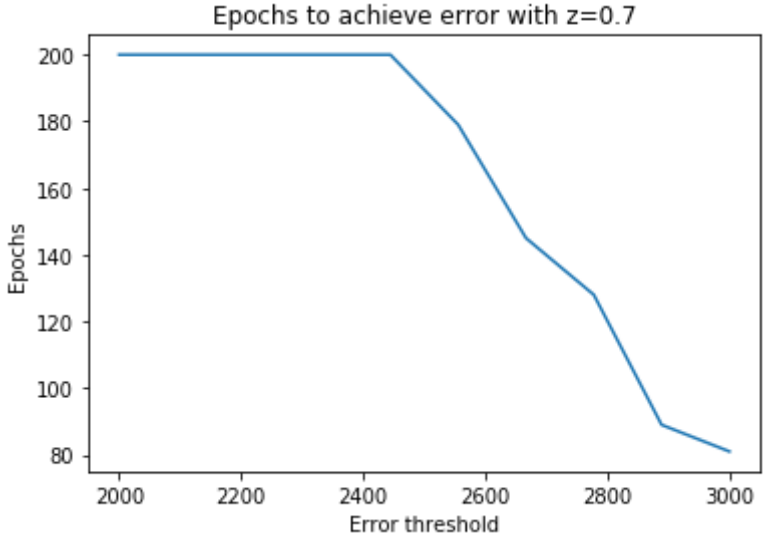


График зависимости количества эпох от заданного порогового значения ошибки:



Выводы

Цели, поставленные на эту лабораторную работу достигнуты, поскольку удалось модифицировать исходный код лабораторной работы 1 таким образом, чтобы получить автоенкодер с применением библиотеки для глубокого обучения Keras. Keras действительно облегчает работу с построением сети, поскольку для ее создания потребовалось лишь пару строк кода. Кроме того, процесс обучения также упрощен благодаря механизму callbacks, в которых можно выполнять те или иные действия по достижении некоторых результатов (например, останавливать обучение, когда достигнута ошибка, заданная пользователем, либо сохранять промежуточные результаты). Дополнительный выбор оптимизаторов и их настроек позволяет подобрать необходимые параметры для каждый задачи.

Литература

1. How does Keras compare to other Deep Learning frameworks like Tensor Flow, Theano, or Torch? [Электронный ресурс] Режим доступа: https://www.quora.com/How-does-Keras-compare-to-other-Deep-Learning-frameworks-like-Tensor-Flow-Theano-or-Torch. Дата доступа: 01.12.2018
2. Keras Intro [Электронный ресурс] Режим доступа: https://keras.io/. Дата доступа: 01.12.2018
3. Keras [Электронный ресурс] Режим доступа: https://en.wikipedia.org/wiki/Keras. Дата доступа: 01.12.2018
4. Tensorflow on Keras [Электронный ресурс] Режим доступа: https://www.tensorflow.org/guide/keras. Дата доступа: 01.12.2018
5. 7 Steps to Mastering Deep Learning with Keras [Электронный ресурс] Режим доступа: https://www.kdnuggets.com/2017/10/seven-steps-deep-learning-keras.html. Дата доступа: 01.12.2018