ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
СПбГУТ))) Июнь 2020 Том 8 № 2 ISSN 2307-1303 http://www.ijitt.ru
TELECOM IT
June 2020 Vol. 8 Iss. 2 ISSN 2307-1303 http://www.ijitt.ru

ПРИМЕНЕНИЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИЗА ФИЗИЧЕСКОЙ АКТИВНОСТИ ПОЛЬЗОВАТЕЛЯ СМАРТФОНА

И. Ю. Зеличенок[1*], Р. Я. Пирмагомедов[2]

[1]Санкт-Петербургский государственный университет телекоммуникаций им. проф. М. А. Бонч-Бруевича, Санкт-Петербург, 193232, Российская Федерация
[2]Университет Тампере, Тампере, 33104, Финляндия
[*]Адрес для переписки: zelichenok.igor@gmail.com

Аннотация—Рост количества данных, генерируемых носимыми электронными устройствами, вынуждает разработчиков пересмотреть традиционные методы анализа данных. Благодаря этому автоматические системы на основе машинного обучения стали намного более востребованными. Алгоритмы подбора контента для конкретного пользователя, распознавание лиц, болезней по фотографиям – это лишь малая часть сфер, в которых можно применить машинное обучение. **Предмет исследования.** В статье разрабатывается приложение для распознавания движений при помощи методов машинного обучения. **Метод.** Для построения модели использовались следующие программные библиотеки языка Python – Pandas, Matplotlib, TensorFlow, NumPy, SkiKit Learn. Также, для сбора данных со смартфона, использовалась программа Physics Toolbox Suite. **Основные результаты.** Разработана модель, способная распознавать заданный набор движений с точностью до 95 %. Подробно описаны все этапы разработки программы, приведен исходный код. **Практическая значимость.** Предложенную модель можно использовать для распознавания активности на умных устройствах, или для систем наружного видеонаблюдения. Кроме того, на основе данной статьи читатели могут разработать аналогичные модели самостоятельно.

Ключевые слова—Машинное обучение, распознавание физической активности, мобильные приложения, информационная безопасность.

Информация о статье
УДК 004.77
Язык статьи – русский.
Поступила в редакцию 08.06.20, принята к печати 15.06.20.

Ссылка для цитирования: Зеличенок И. Ю., Пирмагомедов Р. Я. Применение методов машинного обучения для анализа физической активности пользователя смартфона // Информационные технологии и телекоммуникации. 2020. Том 8. № 2. С. 92–108. DOI 10.31854/2307-1303-2020-8-2-92-108.

TUTORIAL ON USING MACHINE LEARNING FOR ACTIVITY RECOGNITION VIA A SMARTPHONE

I. Zelichenok[1*], R. Pirmagomedov[2]

[1]The Bonch-Bruevich Saint-Petersburg State University of Telecommunications,
St. Petersburg, 193232, Russian Federation
[2]Tampere University, Tampere, 33104, Finland
[*]Corresponding author: zelichenok.igor@gmail.com

Abstract—This article provides a tutorial for developing a simple machine learning application in Python. More specifically, the paper considers daily activity recognition using sensors of a smartphone. For development, we used TensorFlow, Skikit learn, NumPy, Pandas, and Matplotlib. The paper explains in detail the main steps of the application development, including data collection and preprocessing, design of the neural network, learning process, and use of a trained model. The overall accuracy of the developed application when recognizing the activity is about 95 %. This paper can be useful for students and specialists who want to start work on machine learning.
Keywords—machine learning, daily activity recognition, mobile applications information security.

Article info
Article in Russian.
Received 08.06.20, accepted 15.06.20.

For citation: Zelichenok I., Pirmagomedov R.: Tutorial on Using Machine Learning for Activity Recognition Via a Smartphone // Telecom IT. 2020. Vol. 8. Iss. 2. pp. 92–108 (in Russian). DOI 10.31854/2307-1303-2020-8-2-92-108.

Введение

Анализ физической активности (англ. *activity recognition*) – одна из типовых задач, решаемых с использованием искусственных нейросетей. Искусственные нейронные сети способны анализировать данные о физической активности, собираемые при помощи устройств носимой электроники или смартфона. В данной статье мы подробно рассмотрим все этапы разработки такой сети, начиная от подготовки набора данных для обучения (*dataset*), заканчивая реализацией модели. Вся работа выполнялась в программной среде Jupiter Notebook с использованием специализированных библиотек Python.

Распознавание движений без использования методов машинного обучения было весьма трудоемкой задачей, для решения которой требовалось вручную задавать диапазоны колебаний акселерометра. Как следствие, приложения, основанные на физической активности, были весьма ограничены. Можно вспомнить, например, смартфоны от компании Samsung, которые распознавали движения глаз через фронтальные камеры. Эта технология работала плохо, движения распознавались неверно, и в итоге она не прижилась.

В данной статье разрабатывается приложение для мониторинга активности пользователя смартфона. Для обучения было решено записать следующие действия: хождение по лестнице (*Stairs*), посадка на стул (*Sit)*, доставание телефона

левой/правой рукой из кармана джинсов (*Lefthand/Righthand*), и доставание телефона из нагрудного кармана (*Pocket*). Такие модели достаточно универсальны и имеют большие возможности для реализации. Это может быть полезно, например, для фитнес-браслетов, или для системы наружного видеонаблюдения.

Статья состоит из 8 разделов, в каждом из которых рассмотрены различные этапы решения поставленной задачи. В разделе «Искусственные нейронные сети» мы коснемся теории, которая должна обеспечить читателю базовый понятийный аппарат, необходимый для понимания процедур, описанных в дальнейших разделах. В разделе «Подготовка данных» описаны аспекты сбора и подготовки данных для обучения нейронной сети. В следующем разделе, называемом «Статистический анализ данных» представлена визуализация «сырых» исходных данных, и будет проведен их предварительный анализ. В разделе «Предобработка данных» мы преобразуем наши данные в формат, подходящий для обучения модели. В разделе «Построение модели» мы построим модель и зададим параметры для нее. Далее, в разделе «Обучение и тестирование», мы обучим модель и подготовим ее для практического использования. Затем, в разделе «Численные данные» рассмотрим, насколько эффективно работает обученная модель используя для этого тестовые выборки. И, наконец, в разделе «Реализация» мы сделаем простейшую программу, в которой будет использоваться наша модель (обученная искусственная нейронная сеть) для выявления активностей.

Искусственные нейронные сети

В зависимости от решаемой задачи, методы машинного обучения подразделяются на пять основных классов:

1. Задача регрессии – к таким задачам относятся прогнозы, составляемые на основе некоторого набора начальных данных с различными признаками. Например, цены акций, выручка ресторана или инфляция.

2. Задача классификации – задачи, в которых нужен, как правило, один ответ, например: «Да» или «Нет». Определение объекта на изображении, задачи типизации, и другие.

3. Задача кластеризации – такие задачи требуют распределения исходного набора данных на группы. Например, кластеризация клиентов по уровню платежеспособности.

4. Задача уменьшения размерности – сведение большого числа признаков к меньшему (обычно 2–3) для удобства их последующей визуализации (например, сжатие данных).

5. Задача выявления аномалий – отделение аномалий от стандартных случаев. Схоже с задачами классификации, но, в отличие от оных, процесс направлен на выявление редких случаев (аномалий).

Наша задача относится к задаче классификации, так как имеется изначально определенный набор категорий, к которым необходимо соотнести действия, выполняемые пользователем. То есть, раздать метки неизвестному изначально набору данных, полученному при помощи акселерометра.

В нашем случае среди потока данных нет точного разделения, когда и какое действие происходит. Все люди разные, а, значит, время между «покоем» и «действием» не будет точным. Продолжительность колебаний будет разниться, а переход из одного действия к другому будет неявным.

Анализ данных построчно будет неэффективным, так как координат одной точки в пространстве не будет хватать для определения действия. Необходимо анализировать данные «блоками», иными словами, мы разделим данные, получаемые с датчиков на части длинной в фиксированное количество записей (за некоторый временной интервал), которые будут связаны между собой.

Для анализа блоков данных, подойдут рекуррентные нейронные сети. Их особенность в том, что те способны обрабатывать серии событий во времени. Эту их особенность сети мы и будем использовать в этой работе. Для разрабатываемой модели была выбрана архитектура LSTM (*Long short-term memory*), являющаяся одной из разновидностей рекуррентных нейронных сетей. Как следует из литературы данная архитектура отлично подходит для задач классификации, обработки и прогнозирования временных рядов в случаях, когда важные события разделены временными лагами с неопределённой продолжительностью и границами [1, 4].

Подготовка данных

Для сбора данных мы используем программу на Android, под названием Physics Toolbox Suite. Она позволяет записывать данные с датчиков смартфона и сохранять их в формате csv. На рис. 1 показан интерфейс самой программы, а в таблице 1 пример данных, которые она считывает.

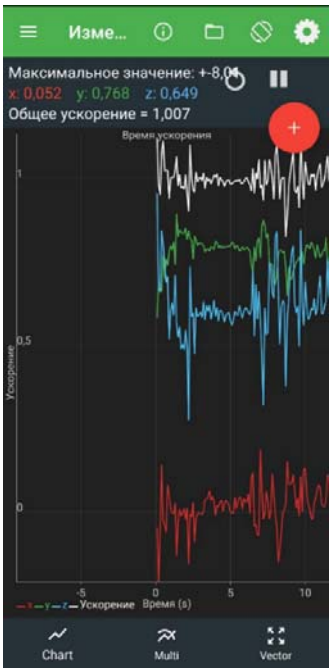

Рис. 1. Интерфейс Physics Toolbox Suite

Таблица 1.

Пример данных, снимаемых Physics Toolbox Suite

Time	gFx	gFy	gFz	TgF
0.003	0.0176	0.4417	0.8121	0.925
0.004	0.0137	0.4456	0.8179	0.932
0.004	0.0098	0.4495	0.8228	0.938
0.004	0.0059	0.4515	0.8267	0.942

Для дальнейшей работы нужно провести предварительную обработку исходных данных и привести их к виду, указанному в таблице 2 (добавились ярлыки и идентификаторы пользователей). Были отсечены ненужные данные. Из старых сохранилось только время и координаты XYZ. В наборе данных для обучения 6 колонок и 226 137 строк.

Таблице 2.

Формат исходных данных после предварительной обработки

User	Activity	Timestamp	Xaxis	Yaxis	Zaxix
1	Lefthand	0.003	0.0176	0.4417	0.8121
1	Lefthand	0.004	0.0137	0.4456	0.8179
1	Lefthand	0.004	0.0098	0.4495	0.8228
1	Lefthand	0.004	0.0059	0.4515	0.8267

Статистический анализ данных

Для начала импортируем набор данных для обучения в Python.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
import seaborn as sns
from pylab import rcParams
from sklearn import metrics

%matplotlib inline

sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42
columns = ['user','activity','timestamp', 'xaxis', 'yaxis', 'zaxis']
df = pd.read_csv('./data.csv', header = None, names = columns, sep = ';')
df.user = df.user.astype(float)
df.head()
```

Таблица 3.

Вывод данных в Jupiter Notebook

	User	Activity	Timestamp	Xaxis	Yaxis	Zaxix
0	1	Lefthand	0.003	0.0176	0.4417	0.8121
1	1	Lefthand	0.004	0.0137	0.4456	0.8179
2	1	Lefthand	0.004	0.0098	0.4495	0.8228
3	1	Lefthand	0.004	0.0059	0.4515	0.8267
4	1	Lefthand	0.005	0.0020	0.4534	0.8326

Далее рассмотрим количество записей по активностям и пользователям (рис. 2, см. ниже).

```
df['activity'].value_counts().plot(kind='bar', title='Training examples by activity type');
```

Так как мы работаем с непрерывным потоком данных, нам следует визуализировать их для определения длительности блока данных при их сегментировании.

На графиках будет представлена зависимость положения датчиков в пространстве от времени [2].

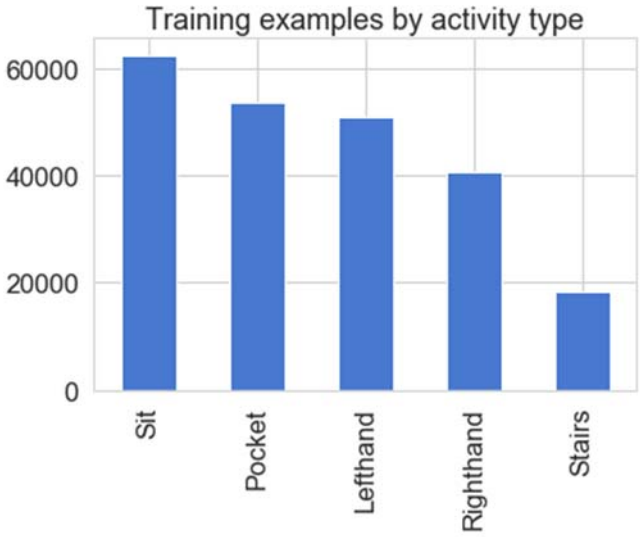

Рис. 2. Количество записей в наборе данных по активностям

```
def plot_activity(activity, df):
    data = df[df['activity'] == activity][['xaxis', 'yaxis', 'zaxis']][:500]
    axis = data.plot(subplots=True, figsize=(16, 12),
                     title=activity)
    for ax in axis:
        ax.legend(loc='lower left', bbox_to_anchor=(1.0, 0.5))
```

На следующих графиках (рис. 3 ,4) выполняются схожие действия, при использовании смартфона левой и правой рукой. В них можно заметить, что наиболее сильные колебания возникают по оси X.

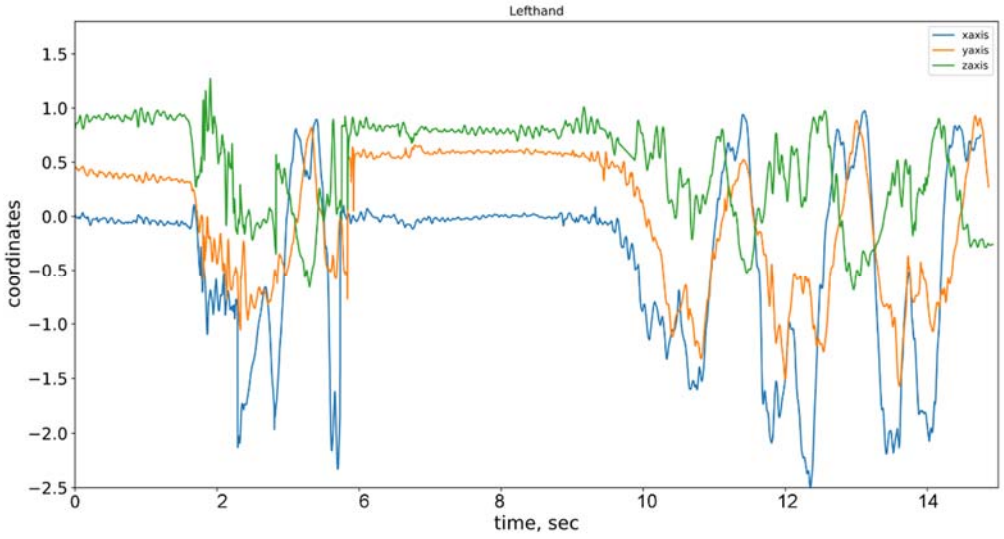

Рис. 3. Графики активностей Lefthand

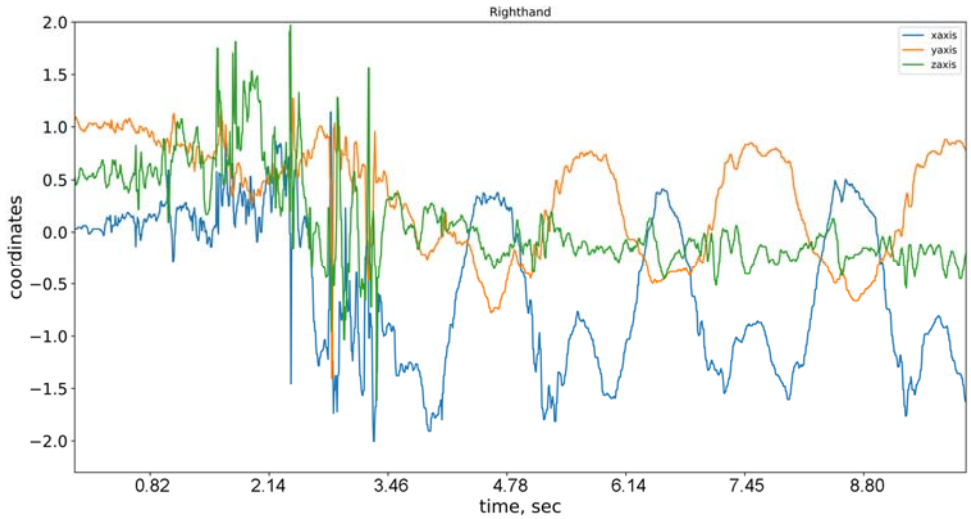

Рис. 4. Графики активностей Righthand

Также на обоих рисунках наблюдается изменение характера колебаний, по которому можно определить, когда именно устройство извлекалось из кармана, а затем его просто держали в руке.

Яркое различие по осям Y и Z можно объяснить тем фактом, что человек, достающий устройство – правша. Соответственно, правой рукой он действовал увереннее и быстрее.

Показания графика далее (рис. 5) ярко выражены наличием импульса координат X, Y и Z в момент, когда человек садится. В остальных же случаях данные колеблются равномерно, так как человек не предпринимает никаких активных действий.

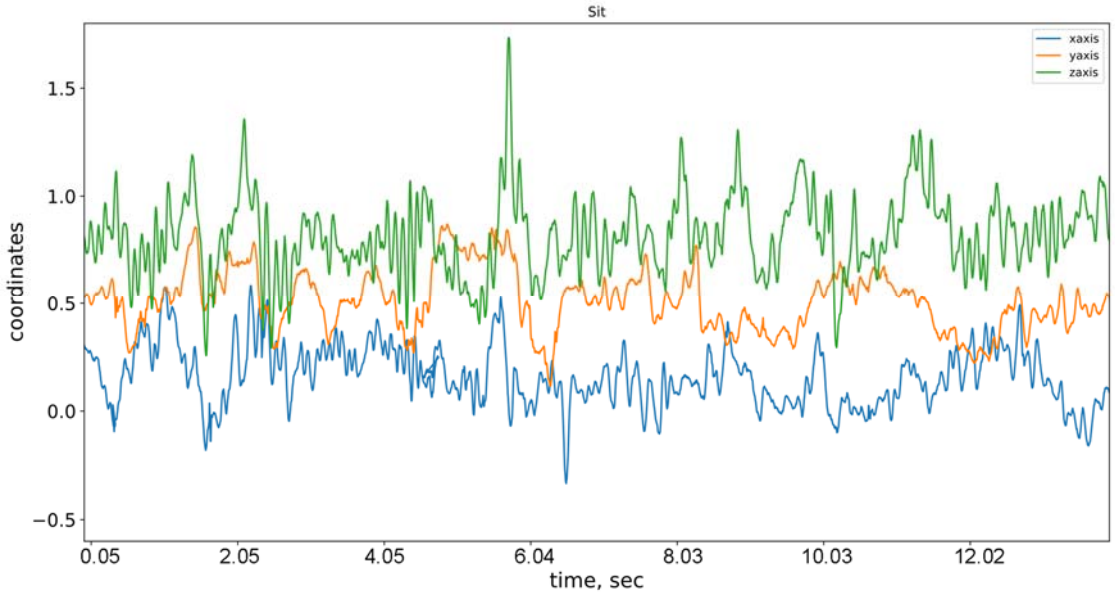

Рис. 5. Графики активностей Sit

На графике, показывающем активность Stairs (рис. 6) нам важно отследить не сколько колебания каждого графика в отдельности, сколько увидеть общую

тенденцию. Устройство начинает колебаться по всем осям координат с определенным временным периодом, что характерно для подъема или спуска по лестнице.

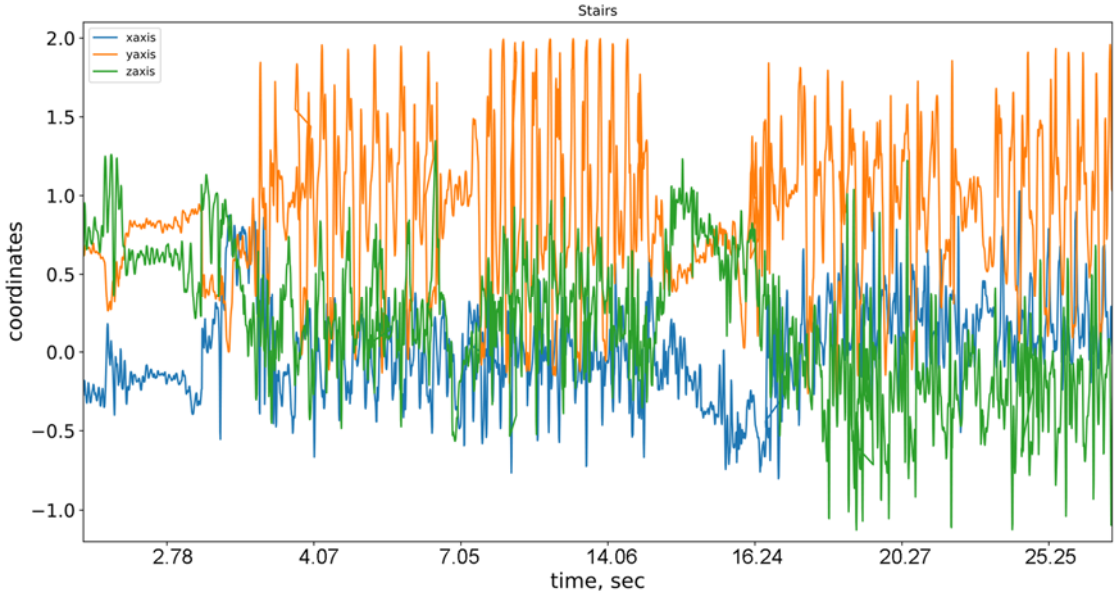

Рис. 6. Графики активностей Stairs

На графике активности Pocket (рис. 7) мы видим импульс по координатам Y и Z, затем их значения меняются местами, затем идет второй импульс, и они снова меняются местами. По этому графику можно представить, как телефон достают из кармана, держат в руке, а затем снова кладут в карман.

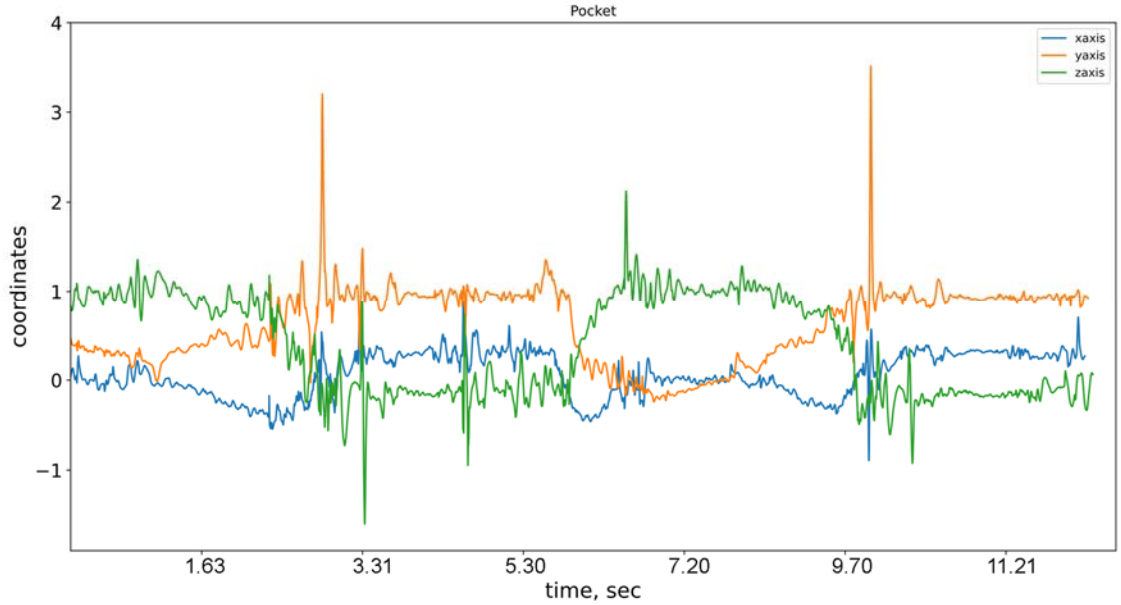

Рис. 7. Графики активностей Pocket

Если взять все периоды шумов и характерных для рассматриваемых действий показаний, можно выяснить, что в среднем, один информационный блок

занимает около двухсот записей в нашем наборе данных. Следовательно, на один блок, подающийся на вход модели, мы выдадим 200 значений из исходных данных.

Предобработка данных

Мы разделим данные на несколько блоков по 200 строк.

```
N_TIME_STEPS = 200
N_FEATURES = 3
step = 20
segments = []
labels = []
for i in range(0, len(df) - N_TIME_STEPS, step):
    xs = df['xaxis'].values[i: i + N_TIME_STEPS]
    ys = df['yaxis'].values[i: i + N_TIME_STEPS]
    zs = df['zaxis'].values[i: i + N_TIME_STEPS]
    label = stats.mode(df['activity'][i: i + N_TIME_STEPS])[0][0]
    segments.append([xs, ys, zs])
    labels.append(label)

np.array(segments).shape

>>(11297, 3, 200)
```

Заметим, что размер нашего набора данных уменьшился после преобразования. Также, стоит заметить, что метка выдается всему блоку по самой распространенной активности в этом блоке.

Далее преобразуем форму нашего массива в 200 строк, каждая из которых содержит координаты X, Y, Z.

```
reshaped_segments = np.asarray(segments, dtype= np.float32).reshape(-1, N_TIME_STEPS, N_FEATURES)
labels = np.asarray(pd.get_dummies(labels), dtype = np.float32)

reshaped_segments.shape

>>(11297, 200, 3)
```

Здесь мы делим исходный набор данных на тренировочный и поверочный.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
      reshaped_segments, labels, test_size=0.2, random_state=RANDOM_SEED)
```

Построение модели

Далее начинается самое важное в нашей работе – это построение модели. Для начала инициализируем два LSTM слоя, по 64 нейрона в каждом (64 нейрона – это стандартное число, которое можно увеличивать или уменьшать для увеличения эффективности обучения). В случае если эффективность работы сети будет недостаточная, данные параметры могут быть изменены. На данный момент не существует четких рекомендаций по выбору количества слоев и нейронов при разработке нейронной сети, в связи с чем эта задача по большей части решается эвристически. Однако следует помнить, что слишком маленькое количество нейронов в скрытых слоях приводит к грубой аппроксимации сетью целевой функции, в то время как их чрезмерное количество приводит к эффекту переобучения [1, 3].

Далее зададим функцию оптимизации, функцию расчета потерь, количество слоев, тип модели, и количество нейронов [5]:

```
N_CLASSES = 5
N_HIDDEN_UNITS = 64
def create_LSTM_model(inputs):
    W = {
        'hidden': tf.Variable(tf.random_normal([N_FEATURES, N_HIDDEN_UNITS])),
        'output': tf.Variable(tf.random_normal([N_HIDDEN_UNITS, N_CLASSES]))
    }
    biases = {
        'hidden': tf.Variable(tf.random_normal([N_HIDDEN_UNITS], mean=1.0)),
        'output': tf.Variable(tf.random_normal([N_CLASSES]))
    }

    X = tf.transpose(inputs, [1, 0, 2])
    X = tf.reshape(X, [-1, N_FEATURES])
    hidden = tf.nn.relu(tf.matmul(X, W['hidden']) + biases['hidden'])
    hidden = tf.split(hidden, N_TIME_STEPS, 0)

    lstm_layers = [tf.contrib.rnn.BasicLSTMCell(N_HIDDEN_UNITS, forget_bias=1.0) for _ in range(2)]
    lstm_layers = tf.contrib.rnn.MultiRNNCell(lstm_layers)

    outputs, _ = tf.contrib.rnn.static_rnn(lstm_layers, hidden, dtype=tf.float32)

    lstm_last_output = outputs[-1]

    return tf.matmul(lstm_last_output, W['output']) + biases['output']
```

Зададим placeholder. Это простая переменная, благодаря которой мы сможем создавать нашу модель, не обращаясь к данным. Это нужно для удобства, чтобы потом одной строчкой приписать нужные данные во всем коде:

```
tf.reset_default_graph()

X = tf.placeholder(tf.float32, [None, N_TIME_STEPS, N_FEATURES], name="input")
Y = tf.placeholder(tf.float32, [None, N_CLASSES])

pred_Y = create_LSTM_model(X)

pred_softmax = tf.nn.softmax(pred_Y, name="y_")
```

Далее зададим параметры расчета потерь:

```
L2_LOSS = 0.0015

l2 = L2_LOSS * \
    sum(tf.nn.l2_loss(tf_var) for tf_var in tf.trainable_variables())

loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = pred_Y, labels = Y)) + l2
```

Затем оптимизатор:

```
LEARNING_RATE = 0.0025

optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE).minimize(loss)

correct_pred = tf.equal(tf.argmax(pred_softmax, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, dtype=tf.float32))
```

Обучение и тестирование

Далее мы переходим к обучению модели. Параметризуем основные характеристики обучения [6]:

```
N_EPOCHS = 50
BATCH_SIZE = 1024
saver = tf.train.Saver()

history = dict(train_loss=[],
               train_acc=[],
               test_loss=[],
               test_acc=[])

sess=tf.InteractiveSession()
sess.run(tf.global_variables_initializer())

train_count = len(X_train)
```

```
for i in range(1, N_EPOCHS + 1):
    for start, end in zip(range(0, train_count, BATCH_SIZE),
                          range(BATCH_SIZE, train_count + 1,BATCH_SIZE)):
        sess.run(optimizer, feed_dict={X: X_train[start:end],
                                       Y: y_train[start:end]})

    _, acc_train, loss_train = sess.run([pred_softmax, accuracy, loss], feed_dict={
                                        X: X_train, Y: y_train})

    _, acc_test, loss_test = sess.run([pred_softmax, accuracy, loss], feed_dict={
                                        X: X_test, Y: y_test})

    history['train_loss'].append(loss_train)
    history['train_acc'].append(acc_train)
    history['test_loss'].append(loss_test)
    history['test_acc'].append(acc_test)

    if i != 1 and i % 10 != 0:
        continue

    print(f'epoch: {i} test accuracy: {acc_test} loss: {loss_test}')

predictions, acc_final, loss_final = sess.run([pred_softmax, accuracy, loss], feed_dict={X: X_test, Y:
y_test})

print()
print(f'final results: accuracy: {acc_final} loss: {loss_final}')
```

Модель показала точность около 93 % (рис. 8), что можно считать довольно неплохим результатом.

```
epoch: 1 test accuracy: 0.4526548683643341 loss: 2.038163661956787
epoch: 10 test accuracy: 0.7942478060722351 loss: 1.2534942626953125
epoch: 20 test accuracy: 0.8646017909049988 loss: 0.9993554949760437
epoch: 30 test accuracy: 0.8951327204704285 loss: 0.8664436340332031
epoch: 40 test accuracy: 0.9199115037918091 loss: 0.756486713886261
epoch: 50 test accuracy: 0.9314159154891968 loss: 0.7040141820907593

final results: accuracy: 0.9314159154891968 loss: 0.7040141820907593
```

Рис. 8. Ход выполнения обучения модели

Далее сохраним историю обучения и классификации данных, это потребуется для дальнейшей визуализации процесса обучения:

```
import pickle

pickle.dump(predictions, open("predictions.p", "wb"))
```

```
pickle.dump(history, open("history.p", "wb"))
tf.train.write_graph(sess.graph_def, '.', './checkpoint/har.pbtxt')
saver.save(sess, save_path = "./checkpoint/har.ckpt")
sess.close()
history = pickle.load(open("history.p", "rb"))
predictions = pickle.load(open("predictions.p", "rb"))
```

Численные результаты

С помощью matplotlib мы выведем график хода обучения (рис. 9) и граф с результатами распознавания (рис. 10):

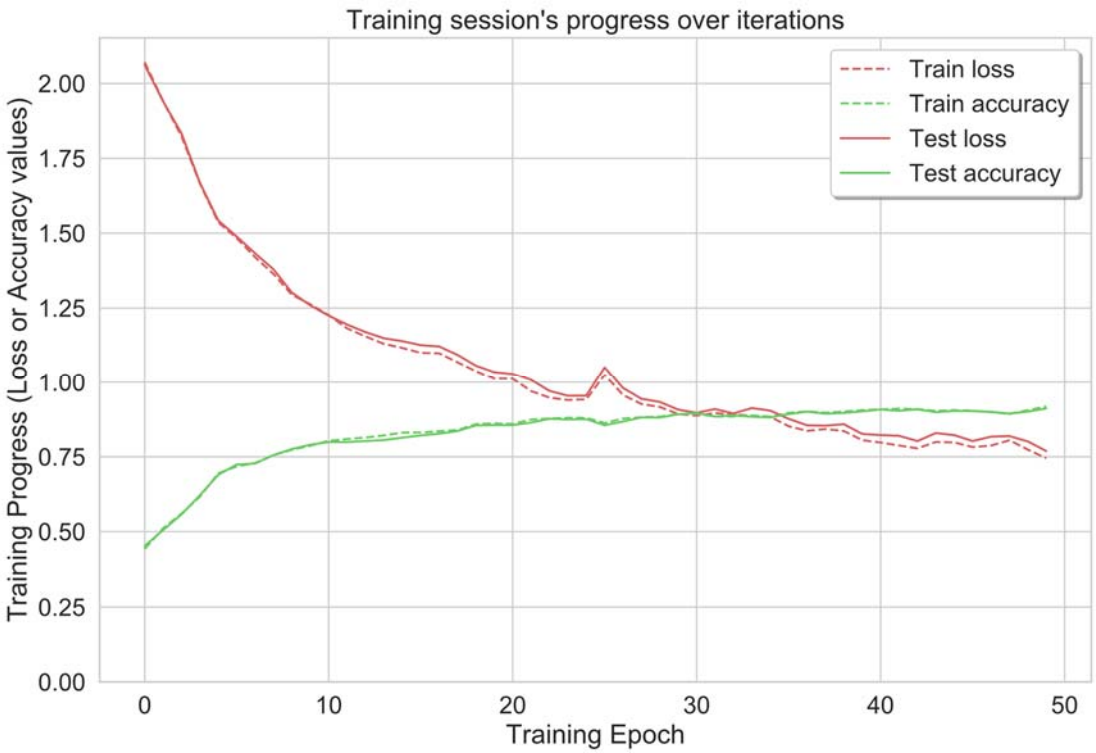

Рис. 9. График прогресса в ходе обучения модели

А теперь посмотрим на матрицу соответствия (точности распознавания) для нашей модели (рис. 10).

Наихудшая точность распознавания при доставании телефона из левого кармана. Однако в целом результаты работы модели можно считать приемлемыми.

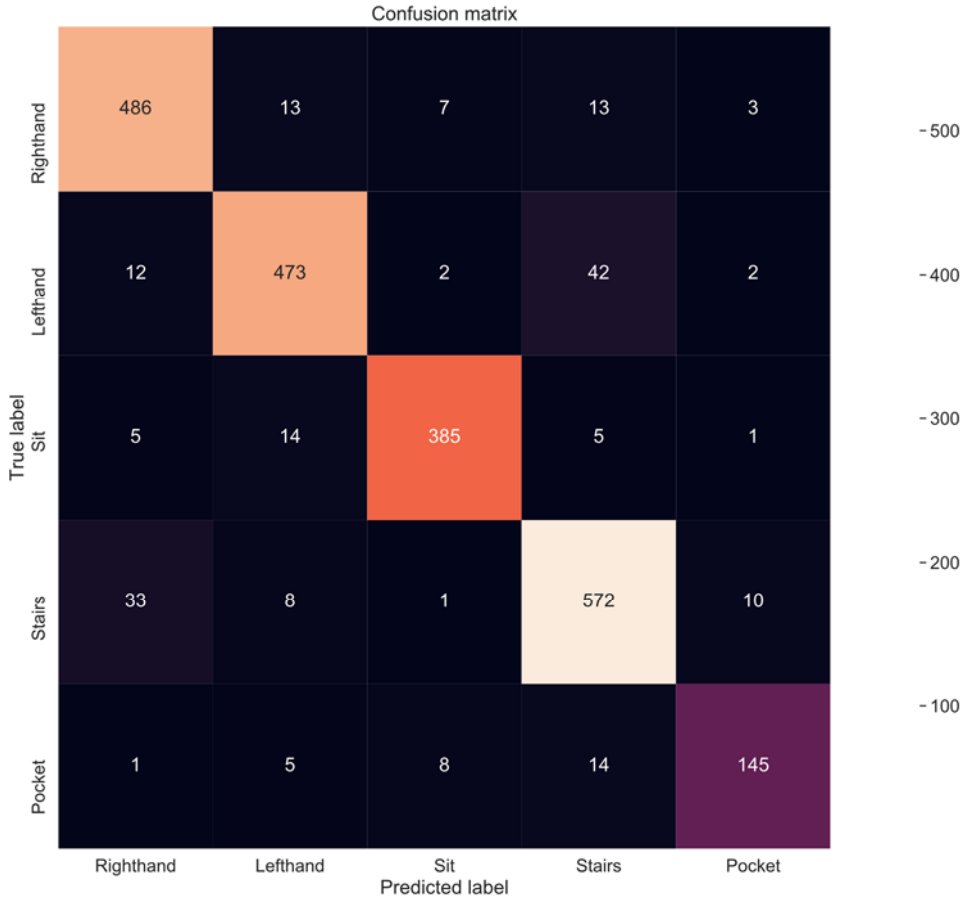

Рис. 10. Матрица соответствия

Сохраним нашу модель. После данных действий в нашей рабочей папке появится файл с моделью – «frozen_har.pb»:

```
from tensorflow.python.tools import freeze_graph

MODEL_NAME = 'har'

input_graph_path = 'checkpoint/' + MODEL_NAME+'.pbtxt'
checkpoint_path = './checkpoint/' +MODEL_NAME+'.ckpt'
restore_op_name = "save/restore_all"
filename_tensor_name = "save/Const:0"
output_frozen_graph_name = 'frozen_'+MODEL_NAME+'.pb'
freeze_graph.freeze_graph(input_graph_path, input_saver="",
                          input_binary=False, input_checkpoint=checkpoint_path,
                          output_node_names="y_", restore_op_name="save/restore_all",
                          filename_tensor_name="save/Const:0",
                          output_graph=output_frozen_graph_name,                    clear_devices=True,
initializer_nodes="")
```

Реализация

В этом разделе мы рассмотрим, как использовать обученную модель в реальном приложении:

```
import numpy as np
import pandas as pd

labels = []
labels_filename = "labels.txt"
columns = ['timestamp', 'tan', 'xaxis', 'yaxis', 'zaxis']
df = pd.read_csv('./mansit1.txt', header = None, names = columns, sep = ';')

with open(labels_filename, 'rt') as lf:
    for l in lf:
        labels.append(l.strip())
```

Сделаем предобработку данных, аналогичную той, что мы делали для самой модели:

```
N_TIME_STEPS = 200
N_FEATURES = 3
step = 1
segments = []
for i in range(0, len(df) - N_TIME_STEPS, step):
    xs = df['xaxis'].values[i: i + N_TIME_STEPS]
    ys = df['yaxis'].values[i: i + N_TIME_STEPS]
    zs = df['zaxis'].values[i: i + N_TIME_STEPS]
    segments.append([xs, ys, zs])

sample = np.asarray(segments, dtype= np.float32).reshape( -1, N_TIME_STEPS, N_FEATURES)
```

И запустим процесс распознавания. Здесь мы загружаем нашу сохраненную модель «frozen_har.pb».

```
from tensorflow.python.platform import gfile
import tensorflow as tf
with tf.Session() as sess:
    with tf.gfile.GFile('frozen_har.pb', 'rb') as f:
        graph_def = tf.GraphDef()
        graph_def.ParseFromString(f.read())
        sess.graph.as_default()
        g_in = tf.import_graph_def(graph_def)
    tensor_output = sess.graph.get_tensor_by_name('import/y_:0')
    tensor_input = sess.graph.get_tensor_by_name('import/input:0')
    predictions = sess.run(tensor_output, {tensor_input:sample})
```

И выведем результат распознавания активности. С итоговым коэффициентом больше 95 наша модель правильно распознала действие.

```
print(labels)
print(predictions)
```

```
['Righthand', 'Lefthand', 'Stairs', 'Sit', 'Pocket']
[[0.10768578 0.25728878 0.00851523 0.62451625 0.00199398]
[0.10395094 0.26419628 0.00917482 0.62066394 0.00201407]
[0.10042222 0.27101937 0.00984641 0.6166784 0.00203352]
...
[0.00099214 0.01180475 0.01304535 0.95713097 0.01702669]
[0.00101008 0.01186318 0.01278894 0.9574538 0.01688392]
[0.00102522 0.01187726 0.0127431 0.9573983 0.01695619]]
```

Заключение

В данной работе мы прошли все этапы создания модели «с нуля». Коснулись базовой теории, сбора и предварительного анализа данных, разработали модель искусственной нейронной сети и интегрировали ее в приложение.

Обученная модель показала достаточно высокий процент точности на обучении (93 %) с относительно небольшим показателем потерь данных (большая часть из потерянных данных были блоки с шумами, что в итоге сказывается положительно, так как модель сама их отбрасывает). Процент точности при использовании данных не входящих в обучающую выборку оказался даже выше – 95 %, чего мы и пытались добиться изначально.

Литература

1. Pirmagomedov R. et al. Facilitating mmWave Mesh Reliability in PPDR Scenarios Utilizing Artificial Intelligence // IEEE Access. 2019. Т. 7. pp. 180700-180712.

2. Sandro T., Matplotlib for Python Developers. Packt Publ. 2009. Т. 1. P. 308. ISBN: 9781847197917.

3. Пирмагомедов Р. Я. Исследование отказов физического канала пассивных оптических сетей и разработка методики их прогнозирования: дисс. … канд. техн. наук / Пирмагомедов Рустам Ярахмедович. СПб., 2014.

4. Alpaydin E. Introduction to Machine Learning. Massachusetts institute of technology // DDC 006.3/1-dc23. 2020. Т. 4. pp. 1–23.

5. Venelin V. Human Activity Recognition using LSTMs on Android TensorFlow for Hackers (Part VI). Available at: https://medium.com/@curiousily/human-activity-recognition-using-lstms-on-android-tensorflow-for-hackers-part-vi-492da5adef64 (accessed 20 January 2020).

6. Xingjian S., Zhourong C., Hao W., Dit-Yan Y., Wai-kin W., Wang-chun W. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. 2015. P. 9. Available at: http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf (accessed 29 January 2020).

References

1. Pirmagomedov R. et al. Facilitating mmWave Mesh Reliability in PPDR Sce-narios Utilizing Artificial Intelligence // IEEE Access. 2019. Vol. 7. pp. 180700-180712.

2. Sandro T., Matplotlib for Python Developers. Packt Publ. 2009. Vol. 1. 308 p. ISBN: 9781847197917.

3. Pirmagomedov R. Ya. Issledovanie otkazov fizicheskogo kanala passiv-nyh opticheskih setej I razrabotka metodiki ih prognozirovaniya: diss. … kand. tekhn. nauk / Pirmagomedov Rustam YArahmedovich. SPb., 2014.

4. Alpaydin E. Introduction to Machine Learning. Massachusetts institute of technology // DDC 006.3/1-dc23. 2020. Vol. 4. pp. 1–23.

5. Venelin V. Human Activity Recognition using LSTMs on Android TensorFlow for Hackers (Part VI). Available at: https://medium.com/@curiousily/human-activity-recognition-using-lstms-on-android-tensorflow-for-hackers-part-vi-492da5adef64 (accessed 20 January 2020).

6. Xingjian S., Zhourong C., Hao W., Dit-Yan Y., Wai-kin W., Wang-chun W. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Now-casting. 2015. P. 9. Available at: http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf (accessed 29 January 2020).

Зеличенок Игорь Юрьевич – студент Санкт-Петербургского государственного университета телекоммуникаций им. проф. М. А. Бонч-Бруевича, zelichenok.igor@gmail.com

Zelichenok Igor – student, The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, zelichenok.igor@gmail.com

Пирмагомедов Рустам Ярахмедович – кандидат технических наук, научный сотрудник, Университет Тампере, Финляндия, rustam.pirmagomedov@tuni.fi

Pirmagomedov Rustam – Candidate of Engineering Sciences, Researcher, Tampere University, Finland, rustam.pirmagomedov@tuni.fi