

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:
«Машинное обучение»
По лабораторной работе №4
«Реализация приложения по распознаванию номеров домов»

Выполнил: Сенькович Дмитрий Сергеевич
магистрант кафедры информатики
группы №858642

Проверил: Стержанов Максим Валерьевич
доцент, кандидат технических наук

Минск 2019

Оглавление

1 Постановка задачи	2
2 Предлагаемое решение	3
3 Реализация нейронной сети	4
3.1 Чтение данных и формирование выборок данных	4
3.2 Построение первой нейронной сети	9
4 Эксперименты	13
4.1 Первая сеть и плохое представление данных	13
4.2 Замена способа чтения изображения	13
4.3 Дальнейшие эксперименты с обучением на локальной видеокарте	13
4.4 Эксперименты с обучением в Google Colab	16
5 Мобильное приложение	24
5.1 Конвертация Keras модели в Tensorflow Lite	24
5.2 Мобильное приложение	24

1 Постановка задачи

Имеется выборка The Street View House Numbers изображений чисел на уличных столбах, домах, зданиях и т.д.. Требуется построить приложение на Android/iOS для распознавания чисел.

2 Предлагаемое решение

В лабораторной работе предлагается изучить работу команды Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud и Vinay Shet, которой посвящено выступление на конференции ICLR: "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". Предлагается к просмотру возможное решение задачи, а именно следующая архитектура сети: 8 сверточных слоев по 48, 64, 128, 160 и 4x192, затем 2 полносвязных слоя по 3072 нейрона с 6 выходными слоями. Первый выходной слой соответствует количеству цифр на изображении, остальные пять - по цифре (специальная метка для отсутствия цифры). Таким образом, данная сеть может распознать числа с количеством цифр не более 5. Авторы получили порядка 97% правильных распознаваний.

3 Реализация нейронной сети

3.1 Чтение данных и формирование выборок данных

Данный датасет представлен в двух вариантах:

- более простой, в котором на каждом изображении изображена одна цифра с каждого числа более сложного датасета
- более сложный, состоящий из оригинальных изображений с числами из нескольких цифр.

Так как мы будем работать над распознаванием изображений с end-to-end решением, то и выбран будет второй датасет.

К датасету прилагается структура изображений, а именно метки, т.е. цифры на изображении, и bounding boxes, указывающие месторасположение цифр на изображении.

Для начала, приведем код, позволяющей получить данные из некоторого референса (структура данных, используемая в формате h5):

```
def get_labels_from_h5py_reference(file, ref):  
    if len(ref) > 1:  
        return [int(file[ref.value[j].item()].value[0][0]) for j in  
range(len(ref))]  
    return [int(ref.value[0][0])]  
  
def get_bboxes_from_h5py_reference(file, ref):  
    if len(ref) > 1:  
        return [file[ref.value[j].item()].value[0][0] for j in range(len(ref))]  
    return [ref.value[0][0]]
```

Теперь приведем код парсинга меток (чисел):

```
def extract_labels(dataset_name):
```

```

cached_labels_file_name = dataset_name + '_labels.mat'

if os.path.isfile(cached_labels_file_name):

    return scipy.io.loadmat(cached_labels_file_name)['labels']

file = h5py.File(dataset_name + '/' + STRUCTURE_FILE_NAME, 'r')

digitStruct = file['digitStruct']

bboxes = digitStruct['bbox']

labels = []

for i in range(len(bboxes)):

    bbox = bboxes[i].item()

    target = file[bbox]['label']

    number_digits = get_labels_from_h5py_reference(file, target)

    # because one of the numbers is 135458 > 5

    number_digits = [0 if digit == 10 else digit for digit in
number_digits]

    length = len(number_digits)

    if length > MAX_LENGTH:

        length = length - 1

        del number_digits[MAX_LENGTH:]

    for _ in range(MAX_LENGTH - len(number_digits)):

        number_digits.append(10)

    number_digits = np.insert(number_digits, 0, length, axis=0)

    labels.append(np.array(number_digits))

labels = np.array(labels)

scipy.io.savemat(cached_labels_file_name, mdict={'labels': labels},
oned_as='row')

```

```
return labels
```

Парсинг занимает определенное время, поэтому результат парсинга будем сохранять в mat файлы. Заметим, что одно изображение в датасете имеет 6 цифр, поэтому все цифры после 5 мы просто пропускаем.

Далее, приведем код парсинга bounding boxes:

```
def extract_bboxes(dataset_name):  
    cached_bboxes_file_name = dataset_name + '_bboxes.mat'  
    if os.path.isfile(cached_bboxes_file_name):  
        return scipy.io.loadmat(cached_bboxes_file_name)['bboxes']  
  
    file = h5py.File(dataset_name + '/' + STRUCTURE_FILE_NAME, 'r')  
    digitStruct = file['digitStruct']  
    bboxes = digitStruct['bbox']  
    parsed_bboxes = []  
    for i in range(len(bboxes)):  
        bbox = bboxes[i].item()  
        heights = get_bboxes_from_h5py_reference(file, file[bbox]['height'])  
        widths = get_bboxes_from_h5py_reference(file, file[bbox]['width'])  
        tops = get_bboxes_from_h5py_reference(file, file[bbox]['top'])  
        lefts = get_bboxes_from_h5py_reference(file, file[bbox]['left'])  
        # because one of the numbers is 135458 > 5  
        if len(heights) > MAX_LENGTH:  
            del heights[MAX_LENGTH:]  
            del widths[MAX_LENGTH:]  
            del tops[MAX_LENGTH:]
```

```

        del lefts[MAX_LENGTH:]

    min_top = np.min(tops)

    min_left = np.min(lefts)

    max_bottom = max([tops[i] + heights[i] for i in range(len(tops))])

    max_right = max([lefts[i] + widths[i] for i in range(len(lefts))])

    parsed_bboxes.append(np.array([min_left, min_top, max_right,
max_bottom]))

    parsed_bboxes = np.array(parsed_bboxes)

    scipy.io.savemat(cached_bboxes_file_name, mdict={'bboxes': parsed_bboxes},
oned_as='row')

    return parsed_bboxes

```

Оригинальные bounding boxes состоят из координат левого верхнего угла, высоты и ширины этого bounding box. Мы приведем их к немного другому агрегированному формату: выберем из всех точек всех bounding boxes изображений самую левую, правую, верхнюю и нижнюю координаты. Таким образом мы получим новый bounding box для изображения - содержащий целое число. Это нужно для end-to-end обучения.

Следующим шагом мы обработаем оригинальные изображения, а именно: согласно bounding boxes “вырежем” часть изображения с числом, приведем к одинаковому разрешению 64x64 и сохраним как grayscale изображение для ускорения обучения:

```

def crop_and_resize_images(dataset_name, parsed_bboxes):

    processed_dataset_directory = 'processed_' + dataset_name

    if os.path.isdir(processed_dataset_directory):

        return

```



```

os.makedirs(processed_dataset_directory)

for i in range(len(parsed_bboxes)):

    image_file_name = str(i + 1) + '.png'

    full_file_name = dataset_name + '/' + image_file_name

    parsed_bbox = parsed_bboxes[i]

    cropped_image = Image.open(full_file_name).crop((parsed_bbox[0],
    parsed_bbox[1], parsed_bbox[2], parsed_bbox[3]))

    resized_image = cropped_image.resize([64, 64], Image.ANTIALIAS)

    grayscale_image = rgb2gray(np.array(resized_image))

    img = (((grayscale_image - grayscale_image.min()) /
    (grayscale_image.max() - grayscale_image.min())) * 255.9).astype(np.uint8)

    Image.fromarray(img).save(processed_dataset_directory + '/' +
    image_file_name)

```

Замечание: выше представлен фрагмент финального решения, первоначальный способ будет упомянут далее.

Построение датасетов: сеть обучалась на большей выборке в 33.000+ изображениях и проверялась на меньшей выборке в 13000+ изображениях. Выборки строились простым чтением изображений в numpy массивы с делением на 255 для ускорения вычислений:

```

def build_dataset(dataset_name, dataset_size):

    dataset = []

    processed_dataset_directory = 'processed_' + dataset_name

    for i in range(dataset_size):

        image_file_name = str(i + 1) + '.png'

        full_file_name = processed_dataset_directory + '/' + image_file_name

        dataset.append(io.imread(full_file_name, as_gray=True) / 255)

    return np.reshape(np.array(dataset), (len(dataset), 64, 64, 1))

```

3.2 Построение первой нейронной сети

Приведем один из первых вариантов кода для сверточных выходных и слоев сети:

```
def add_conv_layer(inputs, kernel_size):

    inputs = keras.layers.Conv2D(kernel_size, (5, 5), padding='same',
kernel_initializer='he_normal', activation="relu")(inputs)

    inputs = keras.layers.BatchNormalization()(inputs)

    inputs = keras.layers.MaxPooling2D()(inputs)

    return keras.layers.Dropout(0.4)(inputs)


def build_output_layer(inputs, labels_count):

    return keras.layers.Dense(labels_count, kernel_initializer='he_normal',
activation="softmax")(inputs)
```

Каждый сверточный слой будет иметь `same` паддинг для сохранения изображения, функцию активации ReLU как один из самых широко используемых и быстрых вариантов и инициализацию ядер методом `he_normal` для ускорения сходимости, потому что именно этот инициализатор лучше всего подходит для функции активации ReLU.

Каждый выходной слой будет иметь функцию активации `softmax` и тот же инициализатор весов.

Приведем код построения и обучения одной из первых нейронных сетей с минимальными значениями количества нейронов на сверточных слоях и отсутствующими полносвязными слоями:

```
def evaluate(model, test_dataset, test_labels):

    predictions = model.predict(test_dataset)

    right_predictions_count = 0

    for i in range(len(test_dataset)):

        prediction = [np.argmax(predictions[j][i]) for j in range(6)]
```

```

        real = [np.asscalar(test_labels[0][i]), np.asscalar(test_labels[1][i]),
np.asscalar(test_labels[2][i]), np.asscalar(test_labels[3][i]),
np.asscalar(test_labels[4][i]), np.asscalar(test_labels[5][i])]

        if np.array_equal(prediction, real):

            right_predictions_count += 1

            # print(prediction)

            # print(real)

        return right_predictions_count / len(test_dataset)

train_labels = extract_labels(TRAIN_DATASET_NAME)

print(train_labels.shape)

test_labels = extract_labels(TEST_DATASET_NAME)

print(test_labels.shape)


train_parsed_bboxes = extract_bboxes(TRAIN_DATASET_NAME)

print(train_parsed_bboxes.shape)

test_parsed_bboxes = extract_bboxes(TEST_DATASET_NAME)

print(test_parsed_bboxes.shape)


crop_and_resize_images(TRAIN_DATASET_NAME, train_parsed_bboxes)

crop_and_resize_images(TEST_DATASET_NAME, test_parsed_bboxes)


train_dataset = build_dataset(TRAIN_DATASET_NAME, len(train_parsed_bboxes))

print(train_dataset.shape)

test_dataset = build_dataset(TEST_DATASET_NAME, len(test_parsed_bboxes))

print(test_dataset.shape)


train_labels_lengths = np.reshape(train_labels[:, 0], (len(train_dataset), 1))
- 1

```

```

train_labels_digits1 = np.reshape(train_labels[:, 1], (len(train_dataset), 1))
train_labels_digits2 = np.reshape(train_labels[:, 2], (len(train_dataset), 1))
train_labels_digits3 = np.reshape(train_labels[:, 3], (len(train_dataset), 1))
train_labels_digits4 = np.reshape(train_labels[:, 4], (len(train_dataset), 1))
train_labels_digits5 = np.reshape(train_labels[:, 5], (len(train_dataset), 1))

test_labels_lengths = np.reshape(test_labels[:, 0], (len(test_dataset), 1)) - 1
test_labels_digits1 = np.reshape(test_labels[:, 1], (len(test_dataset), 1))
test_labels_digits2 = np.reshape(test_labels[:, 2], (len(test_dataset), 1))
test_labels_digits3 = np.reshape(test_labels[:, 3], (len(test_dataset), 1))
test_labels_digits4 = np.reshape(test_labels[:, 4], (len(test_dataset), 1))
test_labels_digits5 = np.reshape(test_labels[:, 5], (len(test_dataset), 1))

inputs = keras.layers.Input(shape=(64, 64, 1))

temp = add_conv_layer(inputs, 48)
temp = add_conv_layer(inputs, 64)
temp = add_conv_layer(inputs, 128)
temp = add_conv_layer(inputs, 160)
temp = add_conv_layer(inputs, 192)
temp = add_conv_layer(inputs, 192)
temp = add_conv_layer(inputs, 192)
temp = add_conv_layer(inputs, 192)

temp = keras.layers.Flatten()(temp)

outputs = [build_output_layer(temp, MAX_LENGTH), build_output_layer(temp,
len(CLASSES)), build_output_layer(temp, len(CLASSES)), build_output_layer(temp,
len(CLASSES)), build_output_layer(temp, len(CLASSES)), build_output_layer(temp,
len(CLASSES))]

```

```

t = TicToc('learning')

t.tic()

model = keras.Model(inputs=inputs, outputs=outputs)

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

model.fit(batch_size=256, x=train_dataset, y=[train_labels_lengths,
train_labels_digits1, train_labels_digits2, train_labels_digits3,
train_labels_digits4, train_labels_digits5], epochs=EPOCHS_NUMBER)

t.toc()

print(t.elapsed / 60)

print("Accuracy: %s" % evaluate(model, test_dataset, [test_labels_lengths,
test_labels_digits1, test_labels_digits2, test_labels_digits3,
test_labels_digits4, test_labels_digits5]))

model.save("model.h5", include_optimizer=False)

```

Качество модели будем судить по точности по всем выходам сети: если хоть один выход (количество цифр или хоть одна цифры) неверно, то такой результат не засчитывается.

4 Эксперименты

4.1 Первая сеть и плохое представление данных

Первые сети состояли из 5-8 сверточных слоев с небольшим количеством нейронов из-за опаски не вместить сеть в память видеокарты (используется Nvidia GeForce 1060 Max-Q). Но ключевая проблема была в представлении данных, а именно конвертации RGB изображения в grayscale: изначально все изображения читались библиотекой Pillow методом `image.open('LA')`, который считывал изображения как grayscale. После различных попыток в течение небольшого количества эпох (порядка 10) максимальная точность составляла порядка 7% со значением функции потерь 2.6.

4.2 Замена способа чтения изображения

Первое значительное улучшение получилось при замене конвертации изображения в grayscale - методом `rgb2gray` библиотеки `skimage`. Одно это изменение позволило улучшить точность до 36% в течение 10 эпох со значением функции потерь 1.27.

4.3 Дальнейшие эксперименты с обучением на локальной видеокарте

Далее эксперименты на видеокарте сводились к увеличению количества нейронов на сверточных слоях, добавлении полносвязных слоев, изменении количества нейронов на них, добавление dropout на полносвязные слои, различная длительность обучения. Для некоторых экспериментов некоторые данные (например, длительность экспериментов и количество нейронов на сверточных слоях) не сохранились, потому что не ожидалось, что обучение займет так много времени (только на эксперименты ушло порядка недели):

Количество эпох	Точность (%)	Значение функции	Длительность обучения	Количество полно	Количество филь	Дополнительные	Размер файла	Drop out rate для	Drop out rate для
-----------------	--------------	------------------	-----------------------	------------------	-----------------	----------------	--------------	-------------------	-------------------

		потери	ния (минуты)	освязных слоев	тров на полно связном слое	параметры полно связного слоя		сверточный слой	полно связного слоя
10	36	1.27	-	0	-	-	3x3	-	-
50	32.1	0.25	-	0	-	-	3x3	-	-
10	29.6	1.37	6	0	-	-	3x3	-	-
10	31	1.44	5.16	2	128	-	3x3	-	-
10	38	1.42	5.26	2	256	-	3x3	-	-
10	32	2.05	6.07	2	512	-	3x3	-	-
10	45	1.25	5.8	2	512	relu and he_kernel	3x3	-	-
10	47	1.02	5.68	2	512	relu and he_kernel	5x5	-	-
20	48	0.45	10.88	2	512	relu and he_kernel	5x5	-	-
60	50	0.176	33.14	2	512	relu and he_kernel	5x5	-	-
10	47.9	1.10	5.52	2	512	relu and	5x5	0.3	-

						he_ke rnel			
10	46	0.90	6.07	2	512	relu and he_ke rnel	5x5	0.4	-
20	47.6	0.49	15.52	2	512	relu and he_ke rnel	5x5	0.4	-
10	39.9	3.45	7.85	2	512	relu and he_ke rnel	5x5	0.4	0.1
15	43	2.87	12.73	2	512	relu and he_ke rnel	5x5	0.4	0.1
25	50.9	2.19	14.8	2	512	relu and he_ke rnel	5x5	0.4	0.1
40	52	1.49	34.16	2	512	relu and he_ke rnel	5x5	0.4	0.1
40	49	0.90	31.39	2	512	relu and he_ke rnel	5x5	0.4	-
40	55	0.78	48.03	2	1024	relu and he_ke	5x5	0.4	0.1

						rn1			
150	54	1.43	236.5 9	2	1300	relu and he_ke rn1	5x5	0.4	0.3

На этом этапе стало понятно, что мощности видеокарты не хватает. Также видно, что воссоздать предлагаю архитектуру нейронной сети не удалось: ограничением стала память видеокарты, все 6GB. Далее, для ускорения вычислений, использовался сервис Google Colab.

4.4 Эксперименты с обучением в Google Colab

Google Colab предоставляет Jupiter environment для машинного обучения на Tensorflow и Keras. Используя его, можно проводить вычисления на более мощных девайсах бесплатно. Более того, можно использовать обучение двух сетей одновременно. Сервис предоставляет обучение в трех режимах: на CPU, GPU и TPU. CPU в эксперимента не рассматривался, для экспериментов на GPU использовалось большее значение `batch_size = 256`, что дало некоторый прирост производительности - порядка на 30% времени меньше занимала каждая эпоха - но не такой, как на TPU, поэтому было решено использовать TPU. TPU - разработанное Google устройство специально для машинного обучения. Он позволяет значительно увеличить используемый `batch_size`, что ускоряет вычисление эпохи. В экспериментах использовались значения `batch_size` от 128×8 (минимально рекомендуемое значение) до 512×8 , что позволило ускорить вычисление каждой эпохи с 30-50 секунд на локальной видеокарте до 7-2 секунд на TPU. Ниже представлены эксперименты в Google Colab:

Количество эпох	Точность (%)	Значение функции потерь	Длительность обучения	Количество полных проходов	Количество фильмов	Размер фильтра	Drop out rate на свер	Drop out rate на полн	Сверточные слои	Batch size
-----------------	--------------	-------------------------	-----------------------	----------------------------	--------------------	----------------	-----------------------	-----------------------	-----------------	------------

		рь	(мин ут)	ЗНЫХ слое в	на полн освя жном слое		ТОЧН ЫХ СЛОЯ Х	ОСВЯ ЗНЫХ СЛОЯ Х		
50	55.8	1.21	3.31	3	1024	5x5	0.3	0.25	4 от 48 до 160, 4 по 192	128* 8
50	53.3	1.88	2.95	5	1024	5x5	0.3	0.25	4 от 48 до 160, 4 по 192	128* 8
1000	59	0.42	61.86	2	1400	5x5	0.4	0.25	4 от 48 до 160, 4 по 192	128* 8
50	53.4	2.13	3.05	5	1024	5x5	0.4	0.25	4 от 48 до 160, 4 по 192	128* 8
50	53.7	2.11	3.07	5	1024	5x5	0.3	0.25	4 от 48 до 160, 8 по 192	128* 8

50	51	2.5	4.14	5	1024	5x5	0.3	0.25	4 от 48 до 160, 4 от 228 до 288	128* 8
500	57.9	0.17	26.51	3	1024	5x5	0.3	0.25	4 от 48 до 160, 4 по 192	128* 8
50	49	2.4	4.06	3	1024	5x5	0.3	0.25	4 от 48 до 160, 4 от 228 до 288	128* 8
50	52	2.03	2.07	3	1024	5x5	0.3	0.25	4 от 48 до 160, 8 по 192	256* 8
50	51	1.94	2.07	3	1024	5x5	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	57.6	0.16	26.78	3	1024	5x5	0.3	0.25	4 от	512*

									48 до 160, 8 по 192	8
1000	58.8	0.16	27.69	5	1024	5x5	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
2000	56.3	0.03	52.47	2	1024	5x5	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	59.1	0.19	27.72	2	1024	7x7	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	60.75	0.19	27.72	2	1024	9x9	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	61.3	0.14	29.81	2	1024	11x1 1	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8

2000	61.6	0.04	55.12	2	1024	9x9	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	65.5	0.09	37.45	2	1024	21x2 1	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
100	0.06	11.8	5.97	2	1024	51x5 1	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	65.9	0.18	57.6	2	1024	31x3 1	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	55	2.15	111.6 4	2	1024	51x5 1	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	65.1	0.38	83.21	2	1024	41x4 1	0.3	0.25	4 от 48 до 160, 8 по	512* 8

									192	
2000	65.6	0.05	113.7 1	2	1024	31x3 1	0.3	0.25	4 от 48 до 160, 8 по 192	512* 8
5000	68	0.12	280.4 0	5	1024	31x3 1	0.4	0.4	4 от 48 до 160, 8 по 192	512* 8
100	47	3.32	6.21	2	1024	31x3 1	0.2, 0.25, 0.25, 0.3, 0.5, 0.5, 0.5, 0.5	0.25	4 от 48 до 160, 8 по 192	512* 8
100	46	3.37	6.34	2	1024	31x3 1	0.2, 0.25, 0.25, 0.25, 0.25, 0.3, 0.3, 0.5	0.25	4 от 48 до 160, 8 по 192	512* 8
100	51	3.12	6.3	2	1024	31x3 1	0.2, 0.25, 0.3, 0.4, 0.5, 0.5,	0.25	4 от 48 до 160, 8 по 192	512* 8

							0.5, 0.5			
100	46	1.6	3.14	2	1024	3x3	0.2, 0.25, 0.3, 0.4, 0.5, 0.5, 0.5, 0.5	0.25	4 от 48 до 160, 8 по 192	512* 8
1000	68	0.20	56.48	2	1024	31x3 1	0.2, 0.25, 0.3, 0.4, 0.5, 0.5, 0.5, 0.5	0.25	4 от 48 до 160, 8 по 192	512* 8

Лучшая получившаяся модель максимально приближена к эталонной, но содержит значительно меньше нейронов на полносвязных слоях - 1024 вместе 3072 из-за ограничений TPU. Лучшая точность невысока - 68%. Некоторые замечания: значительное улучшение точности получилось при увеличении размера фильтра, но при этом и значительно замедлилось обучение. Также, фильтр 61x61 использовать большого значения не имеет, как и 41x41 и 51x51: в первом случае каждая эпоха занимает порядка 26 секунд, что не дает возможности обучить сеть в течение 1000-2000 эпох. Увеличение сверточных слоев и нейронов на них не дало никаких улучшений, как и увеличение количества полносвязных слоев. Значительно ускорение дает увеличение batch_size. Пришлось немного урезать датасеты, потому что тестовая выборка должна делиться на 8, а обучающая - на batch_size - ограничения TPU. Также обучение значительно ускоряется для Tensorflow 2.x - на 50%.

После нескольких других лабораторных работ было решено попробовать цветные изображения, так как до этого были опасения, что цветные изображения будут обрабатываться неприемлемо долго, но после “открытия” TPU и Google Colab уже неважно. К сожалению, получилось получить примерно те же результаты, с почти такой же архитектурой сети (разве что размер фильтра не 31×31 , а 13×13). Но такую сеть проще использовать для мобильного приложения: не нужно конвертировать в изображение в оттенках серого.

5 Мобильное приложение

5.1 Конвертация Keras модели в Tensorflow Lite

Сохранить веса и структуру модели можно следующим образом:

```
tf.keras.models.save_model(model, "model")
```

К сожалению, модель TPU сохранить не удалось, и пришлось обучать модель на GPU в течение порядка 5-6 часов. Покажем, как сконвертировать модель:

```
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model("model")

tflite_model = converter.convert()

open("converted_model.tflite", "wb").write(tflite_model)
```

Конвертация модели в разные моменты времени занимает до 20+гб оперативной памяти, поэтому Google Colab с этим не справился. Более того, скачать архив с моделью на локальную машину напрямую тоже не удалось, так как размер ее 2.4гб (максимум скачивалось порядка 34мб). Следующий код используется для загрузки архива с моделью на Google Drive:

```
!zip -r model.zip model

!ls -l --block-size=M model.zip

from google.colab import drive

drive.mount('/content/gdrive', force_remount=True)

!cp model.zip '/content/gdrive/My Drive/'
```

Только после этого можно скачать модель оттуда и сконвертировать ее.

5.2 Мобильное приложение

Так как я не мобильный разработчик, за основу взято следующее приложение:

<https://github.com/teresawu/random.git>

Заменена модель и усложнен способ распознавания модели, так как в оригинальном приложении модель с одним входом и выходом. Более того, из-за `variable output length`, пришлось заменить выходы на `float` значения, а из-за того, что Kotlin не поддерживает Java примитивы, пришлось переписать и классификатор:

```
@RequiresApi(api = Build.VERSION_CODES.CUPCAKE)

public class ImageClassifier {

    private AssetManager assetManager;

    private Interpreter interpreter;

    private ByteBuffer imgData;

    private int[] intValues = new int[INPUT_SIZE * INPUT_SIZE];

    public ImageClassifier(AssetManager assetManager) {

        this.assetManager = assetManager;

        imgData = ByteBuffer.allocateDirect(DIM_BATCH_SIZE * DIM_IMG_SIZE_X *
DIM_IMG_SIZE_Y * DIM_PIXEL_SIZE * 4);

        imgData.order(ByteOrder.nativeOrder());

        try {

            interpreter = new Interpreter(loadModelFile(assetManager,
MODEL_PATH));

        } catch (Exception e) {

            throw new RuntimeException(e);

        }

    }

    private void convertBitmapToByteBuffer(Bitmap bitmap) {

        imgData.rewind();

    }

}
```

```

        bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0,
        bitmap.getWidth(), bitmap.getHeight());

        int pixel = 0;

        for (int i = 0; i < DIM_IMG_SIZE_X; i++) {
            for (int j = 0; j < DIM_IMG_SIZE_Y; j++) {
                int value = intValues[pixel++];

                imgData.putFloat((shr(value, 16) & 0xFF) / 255f);

                imgData.putFloat((shr(value, 8) & 0xFF) / 255f);

                imgData.putFloat((value & 0xFF) / 255f);
            }
        }
    }

    public static int shr(int i, int distance) {
        return (i >>> distance) | (i << -distance);
    }

    @RequiresApi(api = Build.VERSION_CODES.CUPCAKE)
    private MappedByteBuffer loadModelFile(AssetManager assets, String
    modelFilename) {

        AssetFileDescriptor fileDescriptor = null;

        try {
            fileDescriptor = assets.openFd(modelFilename);
        } catch (IOException e) {
            e.printStackTrace();
        }

        FileInputStream inputStream = new
        FileInputStream(fileDescriptor.getFileDescriptor());

```

```

        FileChannel fileChannel = inputStream.getChannel();

        long startOffset = fileDescriptor.getStartOffset();

        long declaredLength = fileDescriptor.getDeclaredLength();

        try {

            return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength);

        } catch (IOException e) {

            e.printStackTrace();

        }

        return null;

    }

    public String recognizeImage(Bitmap bitmap) {

        convertBitmapToByteBuffer(bitmap);

        Object[] inputs = new Object[]{imgData};

        Map<Integer, Object> outputs = new HashMap();

        outputs.put(0, new float[][] { { 128f, 128f, 128f, 128f, 128f } });

        outputs.put(1, new float[][] { { 128f, 128f, 128f, 128f, 128f, 128f,
128f, 128f, 128f, 128f } });

        outputs.put(2, new float[][] { { 128f, 128f, 128f, 128f, 128f, 128f,
128f, 128f, 128f, 128f } });

        outputs.put(3, new float[][] { { 128f, 128f, 128f, 128f, 128f, 128f,
128f, 128f, 128f, 128f } });

        outputs.put(4, new float[][] { { 128f, 128f, 128f, 128f, 128f, 128f,
128f, 128f, 128f, 128f } });

        outputs.put(5, new float[][] { { 128f, 128f, 128f, 128f, 128f, 128f,
128f, 128f, 128f, 128f } });

        interpreter.runForMultipleInputsOutputs(inputs, outputs);

        int numbersCountRecognized = getMaxProbLabel(((float[][])
outputs.get(0))[0]) + 1;

```

```

        String result = "";

        for (int i = 0; i < numbersCountRecognized; i++) {

            int label = getMaxProbLabel(((float[][][]) outputs.get(i+1))[0]);

            result += String.valueOf(label);

        }

        return result;

    }

    private int getMaxProbLabel(float[] probs) {

        float max = Float.MIN_VALUE;

        int maxIndex = -1;

        for (int i = 0; i < probs.length; i++) {

            if (probs[i] > max) {

                max = probs[i];

                maxIndex = i;

            }

        }

        return maxIndex;

    }

    public void close() {

        interpreter.close();

    }

}

```

Покажем работу приложения:

tensorflow/tensorflow:2.0.0

tensorflow/tensorflow:2.0.0

51

TENSORFLOW IMAGE



Рисунок 5.2.1. Демонстрация приложения