Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №4**

«Приложение по распознаванию номеров домов»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2020

ХОД РАБОТЫ

**Задание.**

Набор изображений из Google Street View с изображениями номеров домов, содержащий 10 классов, соответствующих цифрам от 0 до 9.

● 73257 изображений цифр в обучающей выборке;

● 26032 изображения цифр в тестовой выборке;

● 531131 изображения, которые можно использовать как дополнение к обучающей выборке;

● В двух форматах:

* Оригинальные изображения с выделенными цифрами;
* Изображения размером 32 × 32, содержащих одну цифру;

● Данные первого формата можно скачать по ссылкам:

* http://ufldl.stanford.edu/housenumbers/train.tar.gz (обучающая выборка);
* http://ufldl.stanford.edu/housenumbers/test.tar.gz (тестовая выборка);
* http://ufldl.stanford.edu/housenumbers/extra.tar.gz (дополнительные данные);

● Данные второго формата можно скачать по ссылкам:

* http://ufldl.stanford.edu/housenumbers/train\_32x32.mat (обучающая выборка);
* http://ufldl.stanford.edu/housenumbers/test\_32x32.mat (тестовая выборка);
* http://ufldl.stanford.edu/housenumbers/extra\_32x32.mat (дополнительные данные);

● Описание данных на английском языке доступно по ссылке:

* http://ufldl.stanford.edu/housenumbers/

1. Реализуйте глубокую нейронную сеть (полносвязную или сверточную) и обучите ее на синтетических данных (например, наборы MNIST (http://yann.lecun.com/exdb/mnist/) или notMNIST).

Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf), видео на YouTube (https://www.youtube.com/watch?v=vGPI\_JvLoN0).

1. После уточнения модели на синтетических данных попробуйте обучить ее на реальных данных (набор Google Street View). Что изменилось в модели?
2. Сделайте множество снимков изображений номеров домов с помощью смартфона на ОС Android. Также можно использовать библиотеки OpenCV, Simple CV или Pygame для обработки изображений с общедоступных камер видеонаблюдения (например,https://www.earthcam.com/).

Пример использования библиотеки TensorFlow на смартфоне можете воспользоваться демонстрационным приложением от Google (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android).

1. Реализуйте приложение для ОС Android, которое может распознавать цифры в номерах домов, используя разработанный ранее классификатор. Какова доля правильных классификаций?

**Результат выполнения:**

Изначально я хотел обучать нейронную сеть на .mat файликах для распознавания цифр, в комбинации с алгоритмом YOLO для нахождения boundary box, т. е. использовать е2е learning. Однако, после изучения материалов, предоставленных в данной лабораторной работе, я понял, что надо использовать другую архитектуру нейронной сети.

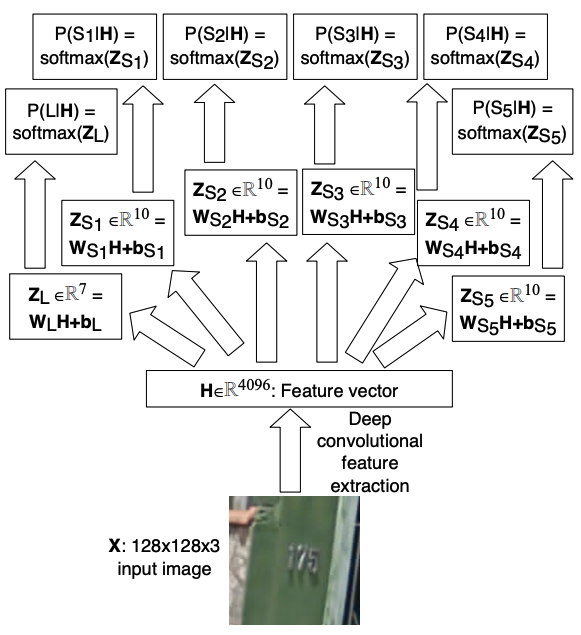


Рисунок 1 – рекомендуемая архитектура нейронной сети

Одна из отличительных особенностей данной нейронной сети – шесть выходных слоёв. Один (с шестью нейронами) – предсказывает длину номера дома, и ещё пять (в каждом по 11 нейронов) – для предсказывания цифры из номера.

1. Я решил обучать на mnist датасете. Согласно приведённой выше архитектуре, я перекодировал выходные данные. Таким образом у меня из ожидаемых шести выходов нейронной сети на первом была всегда единица (т. к. каждое изображение из набора состоит из одной цифры, в отличии от SVHN). Второй выход нейронной сети предсказывал класс цифры, все остальные выходы были замоканы на выход NaN (отсутствие числа). Чисто теоретически, такой подход на предварительном обучении должен дать буст в обучении на реальном датасете, потому что конволюционные слои смогли бы изучить «фичи» изображений.

Перед конструирование нейронной сети я провёл небольшой анализ, и выяснил, что средний размер изображений в SVHN примерно 64х64. Исходный размер mnist – 28x28. Через cv2.resize я заскейлил изображения из mnist датасета до 64х64. И также привёл их к RGB формату путём репликации монохромного канала на R, G, B канал соответственно.

Архитектура нейронной сети должна была быть такой, чтобы её можно было запустить на мобильных девайсах, а не в клиент-сервисной архитектуре. Поэтому количество параметров должно было быть максимально минимальным. Изначально я взял архитектуру, в которой было около 158М параметров, однако после её экспорта она занимала около 1.5Гб, что, очевидно, неприемлимо, даже если её сконвертировать в tflite формат. После некоторых поисков я создал модель, код которой ниже, с 1.7М параметров и которая весила около 16Мб.

Некоторые моменты, вроде фрейма (5,5) были взяты из модели, представленной в оригинальной публикаии.

Код реализации:

!wget "http://ufldl.stanford.edu/housenumbers/train.tar.gz"

!wget "http://ufldl.stanford.edu/housenumbers/test.tar.gz"

!mkdir train

!mkdir test

!tar xvzf train.tar.gz -C train

!tar xvzf test.tar.gz -C test

IMAGE\_WIDTH=64

IMAGE\_HEIGHT=64

IMAGE\_SIZE=(IMAGE\_WIDTH, IMAGE\_HEIGHT)

IMAGE\_CHANNELS=3

batch\_size=32

MAX\_LENGTH = 5

NOT\_A\_NUMBER = 10

import matplotlib.pyplot as plt

def show\_images\_with\_labels(image\_batch,label\_batch,batch\_size=batch\_size):

plt.figure(figsize=(10,10))

for n in range(batch\_size):

ax = plt.subplot(4,8,n+1)

plt.imshow(image\_batch[n])

length=label\_batch[0][n]

title=''.join([str(label\_batch[c+1][n]) for c in range(length)])

plt.title(title)

plt.axis('off')

from keras.datasets import mnist

# the data, split between train and test sets

(x\_train\_mnist, y\_train\_mnist), (x\_test\_mnist, y\_test\_mnist) = mnist.load\_data()

# Normalizing the input

x\_train\_mnist= x\_train\_mnist.reshape(x\_train\_mnist.shape[0], 28, 28, 1) / 255

x\_test\_mnist = x\_test\_mnist.reshape(x\_test\_mnist.shape[0],28,28,1) / 255

import numpy as np

import cv2

def x\_to\_nn\_input(X):

x = np.reshape(X, (X.shape[0], 28, 28))

x = np.asarray([cv2.resize(i, (IMAGE\_WIDTH,IMAGE\_HEIGHT)) for i in x], dtype=np.float32)

x = np.repeat(x[..., np.newaxis], 3, -1)

return x

def y\_mnist\_to\_NN\_architecture\_output(y):

arr = [np.array([]), np.array([]), np.array([]), np.array([]), np.array([]), np.array([])]

for i in range(6):

for j in range(y.shape[0]):

if i == 0:

arr[0] = np.append(arr[0], int(1))

if i == 1:

arr[1] = np.append(arr[1], int(y[j]))

if i >= 2:

arr[i] = np.append(arr[i], int(NOT\_A\_NUMBER))

return [i.astype(int) for i in arr]

from tensorflow.keras import Input, Model

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

input=Input(shape=(IMAGE\_WIDTH, IMAGE\_HEIGHT, IMAGE\_CHANNELS), name="input")

H=Conv2D(16,(5,5),activation='relu',padding='same')(input)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same')(H)

H=Dropout(0.25)(H)

H=Conv2D(32,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same',strides=1)(H)

H=Dropout(0.25)(H)

H=Conv2D(48,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same')(H)

H=Dropout(0.25)(H)

H=Conv2D(64,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same',strides=1)(H)

H=Dropout(0.25)(H)

H=Conv2D(128,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same')(H)

H=Dropout(0.25)(H)

H=Conv2D(16,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same',strides=1)(H)

H=Dropout(0.25)(H)

H=Conv2D(16,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same')(H)

H=Dropout(0.25)(H)

H=Conv2D(16,(5,5),activation='relu',padding='same')(H)

H=BatchNormalization()(H)

H=MaxPooling2D(padding='same',strides=1)(H)

H=Dropout(0.25)(H)

H=Flatten()(H)

hidden10=Dense(4096, activation='relu')(H)

#H=Dropout(0.25)(H)

#hidden10=Dense(4096, activation='relu')(H)

#hidden10=Dropout(0.25)(H)

length=Dense(6, activation='softmax', name='length')(hidden10)

digit1=Dense(11, activation='softmax', name='digit1')(hidden10)

digit2=Dense(11, activation='softmax', name='digit2')(hidden10)

digit3=Dense(11, activation='softmax', name='digit3')(hidden10)

digit4=Dense(11, activation='softmax', name='digit4')(hidden10)

digit5=Dense(11, activation='softmax', name='digit5')(hidden10)

model = Model(inputs=input, outputs=[length,digit1,digit2,digit3,digit4,digit5], name='svhn\_model')

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['acc'])

model.summary()

show\_images\_with\_labels(x\_to\_nn\_input(x\_test\_mnist[0:32])[0:32], y\_mnist\_to\_NN\_architecture\_output(y\_test\_mnist)[0:32])

Результат выполнения:

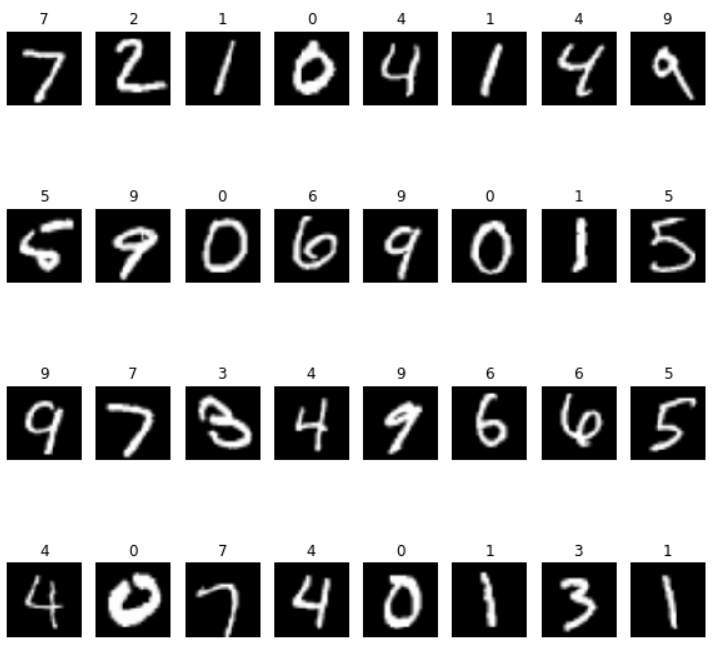


Рисунок 2 – mnist датасет после препроцессинга

Код реализации:

X = x\_to\_nn\_input(x\_train\_mnist[0:5000])

y = y\_mnist\_to\_NN\_architecture\_output(y\_train\_mnist[0:5000])

Xt = x\_to\_nn\_input(x\_train\_mnist[0:1000])

yt = y\_mnist\_to\_NN\_architecture\_output(y\_train\_mnist[0:1000])

history = model.fit(

X,

y,

validation\_data=(Xt, yt),

epochs=5,

verbose=2

)

После пяти эпох точность распознавания первой цифры (остальные данные нас пока что не интересуют, потому что они были замоканы) составляла 96.2% на тренировочных данных и 98.2% на валидационном датасете.

Модель обучалась без аугментации данных на mnist датасете.

2. Чтобы обучать на SVHN датасете надо было привести данные к нужному виду. Код реализации:

import h5py

import pandas as pd

import numpy as np

# The DigitStructFile is just a wrapper around the h5py data. It basically references

# inf: The input h5 matlab file

# digitStructName The h5 ref to all the file names

# digitStructBbox The h5 ref to all struct data

class DigitStructFile:

def \_\_init\_\_(self, inf):

self.inf = h5py.File(inf, 'r')

self.digitStructName = self.inf['digitStruct']['name']

self.digitStructBbox = self.inf['digitStruct']['bbox']

# getName returns the 'name' string for for the n(th) digitStruct.

def getName(self, n):

return ''.join([chr(c[0]) for c in self.inf[self.digitStructName[n][0]].value])

# bboxHelper handles the coding difference when there is exactly one bbox or an array of bbox.

def bboxHelper(self, attr):

if (len(attr) > 1):

attr = [self.inf[attr.value[j].item()].value[0][0] for j in range(len(attr))]

else:

attr = [attr.value[0][0]]

return attr

# Return a restructured version of the dataset.

#

# Return a list of such dicts:

# 'filename': filename of the samples

# 'labels': list of such digits presented by an 6-array

#

# Note: We may turn this to a generator, if memory issues arise.

def getAllDigitStructure(self):

labels = []

filenames=[]

for i in range(len(self.digitStructBbox)):

bbox = self.digitStructBbox[i].item()

label\_attribute = self.inf[bbox]['label']

label = self.bboxHelper(label\_attribute)

# since one of the numbers has 6 numbers! (it is > 5)

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

length = len(label)

name = self.getName(i)

if (length > MAX\_LENGTH):

continue

for \_ in range(MAX\_LENGTH - length):

label.append(NOT\_A\_NUMBER)

label.append(length)

labels.append(label)

filenames.append(name)

return np.array(filenames), np.array(labels).astype(int)

def to\_dataframe(self):

filenames, labels = self.getAllDigitStructure()

return pd.DataFrame({

'filename': filenames,

'length': labels[:,5],

'digit1': labels[:,0],

'digit2': labels[:,1],

'digit3': labels[:,2],

'digit4': labels[:,3],

'digit5': labels[:,4]

})

df = DigitStructFile('train/train/digitStruct.mat').to\_dataframe()

df.tail()

df\_test = DigitStructFile('test/test/digitStruct.mat').to\_dataframe()

df\_test.tail()

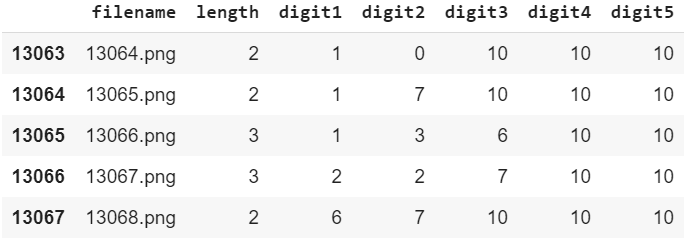


Рисунок 3 – результат выполнения

После этого я запустил обучение, и спустя несколько эпох она обучилась, но когда я проверил результаты на тестовой выборке, то ошибка оказалась выше в 20 раз (0.35 vs 7.06, на тренировочных данных я использовал сразу аугментацию данных). Добавление дополнительной L2-регуляризации и дропаут слоёв также не привели к улучшению показателей (до этого в модели они также использовались, просто не в таком количестве). Использование ранней остановки обучения также было бессмысленным, потому что ошибка на тестовых данных постоянно была высокой.

После серии попыток я решил детальнее посмотреть, что происходит с данными, и обнаружил одну деталь. Дело в том, что все изображения с исходного датасета имеют разное разрешение. И после приведения изображения к формату 64х64 приводит к артефактам.

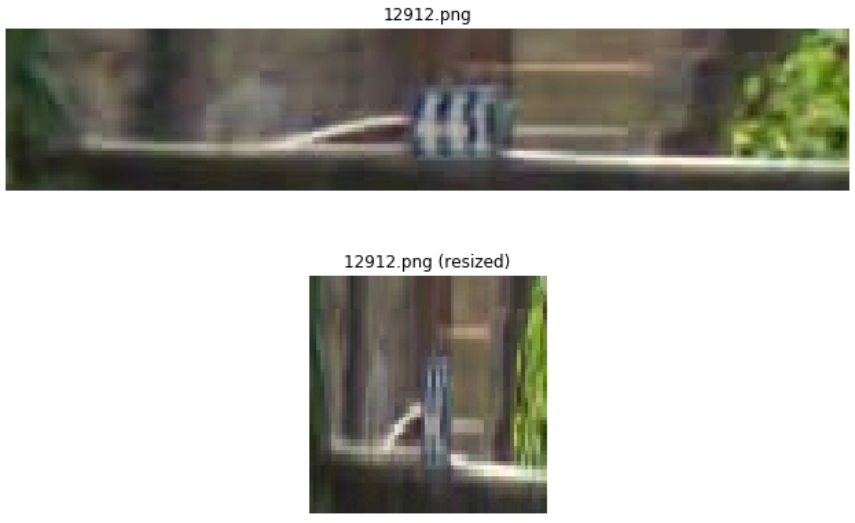


Рисунок 4 – проблема ресайзинга изображений

Как видно из рисунка, в первом случае почти все люди смогут сказать, что на изображении есть цифры 441. Однако в случае 64х64 изображения догадаться что там за цифры уже сложно, и я бы сказал даже нереально. Поэтому я попытался решить эту проблему с помощью «умного» кропинга изображений. Дело в том, что файл digitStruct.mat содержит информацию о boundary box каждой цифры, поэтому суть моего дополнительного препроцессинга сводилась к тому, чтобы найти все боксы чисел, после этого найти самый верхний/нижний/левый/верхний угол из массива этих боксов, и после этого обрезать изображение по этим координатам.



Рисунок 5 – результаты дополнительного препроцессинга

После этого я запустил обучение ещё раз. Код реализации:

from sklearn.model\_selection import train\_test\_split

train, val = train\_test\_split(df, test\_size=0.10)

from keras.preprocessing.image import ImageDataGenerator

# use data augmentation

train\_datagen\_aug = ImageDataGenerator(

rotation\_range=10,

rescale=1./255,

shear\_range=0.1,

zoom\_range=0.1,

width\_shift\_range=0.1,

height\_shift\_range=0.1

)

test\_gen = ImageDataGenerator(rescale=1./255)

TRAIN\_PATH = 'processed/train/train'

TEST\_PATH = 'processed/test/test'

y\_col = ['length', 'digit1', 'digit2', 'digit3', 'digit4', 'digit5']

total\_train = train.shape[0]

total\_validate = val.shape[0]

batch\_size = 32 # since Andrew Ng told, that better use 2^n

train\_generator = train\_datagen\_aug.flow\_from\_dataframe(

train,

TRAIN\_PATH,

x\_col='filename',

y\_col=y\_col,

target\_size=IMAGE\_SIZE,

class\_mode='multi\_output',

batch\_size=batch\_size

)

val\_generator = test\_gen.flow\_from\_dataframe(

val,

TRAIN\_PATH,

x\_col='filename',

y\_col=y\_col,

target\_size=IMAGE\_SIZE,

class\_mode='multi\_output',

batch\_size=batch\_size

)

from tensorflow.keras.callbacks import ModelCheckpoint

# define the checkpoint

filepath = "model-80.h5"

checkpoint = ModelCheckpoint(filepath, monitor='val\_loss', verbose=1, save\_best\_only=True, mode='min')

callbacks\_list = [checkpoint]

history = model.fit\_generator(

train\_generator,

epochs=20,

validation\_data=val\_generator,

validation\_steps=total\_validate//batch\_size,

steps\_per\_epoch=total\_train//batch\_size,

callbacks=callbacks\_list

)

После этого получились следующие результаты.

Таблица 1 – результаты обучения

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Loss | Len | Dig1 | Dig2 | Dig3 | Dig4 | Dig5 |
| Тренировочный датасет | 0.7985 | 97.04 | 92.42 | 92.45 | 95.18 | 98.51 | 99.97 |
| Валидационный датасет | 0.6561 | 97.21 | 94.24 | 94.42 | 96.23 | 98.41 | 99.97 |
| Тестовый датасет | 0.5721 | 97.52 | 94.88 | 94.52 | 97.56 | 99.62 | 99.98 |

В таблице выше, loss – значение функции потерь, Len, Dig1, Dig2, Dig3, Dig4, Dig5 – accuracy, данная в процентах.

Как видно, модель хорошо обучилась, и имеет неплохие показатели (все значения более-менее равны относительно друг-друга). У меня не получилось добиться точности, о которой говорилась в публикации (97.5%), но я довольно близко к ней приблизился. Данная модель обучалась около 30 эпох (только на SVHN датасете, не включая пять эпох на mnist). Как мне кажется, если её обучать дольше, то тех результатов, которые были в статье, можно было бы добиться.

3. Я не совсем понял задание номер три, потому что было сказано, что надо сделать снимки, потом была дана ссылка на тензорфлоу и её использование на андроид, а в чётвертом пункте как раз-таки шла речь о реализации приложения, поэтому в этом пукте я просто сделал несколько снимков на свой телефон, и нашёл несколько изображений в интернете, и протестировал модель на этих картинках:



Рисунок 6 – фотка этикетки напитка «Coca-Cola»

4. Я решил не ограничиваться одной андроид платформой и написать кроссплатформенное приложение (iOS/Android):

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex1data2.csv')

data = pd.read\_csv(file\_path)

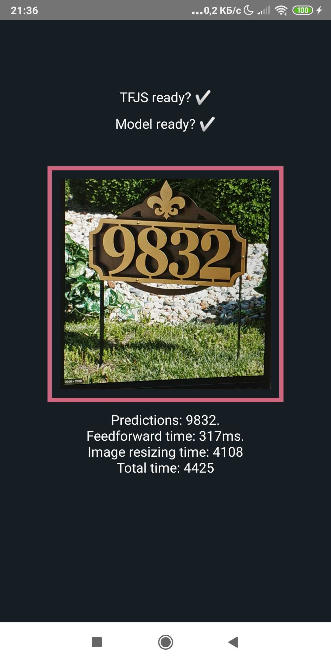
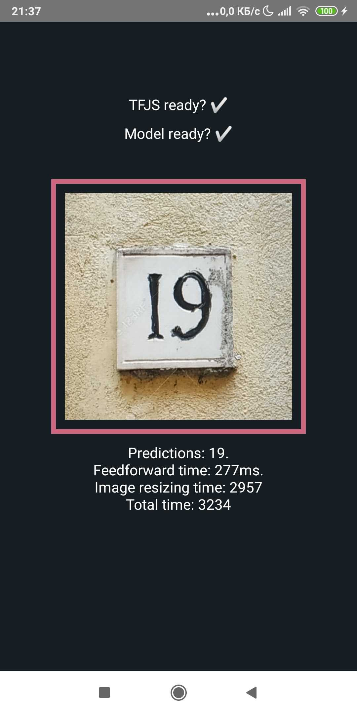
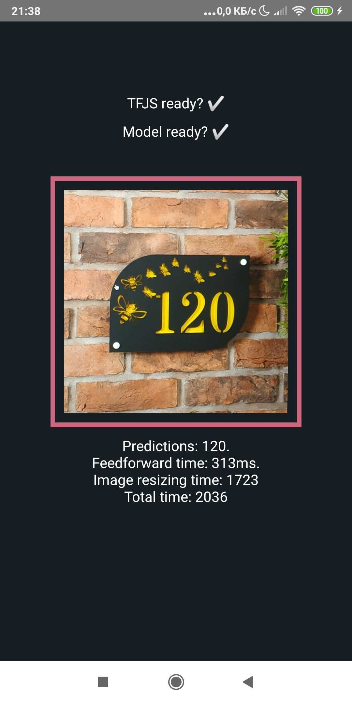
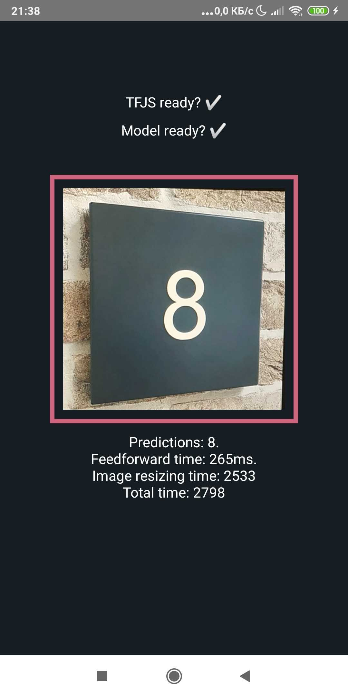


Рисунок 5 – результаты работы кроссплатформенного приложения на Android