МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНИВИДУАЛЬНОЙ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Искусственные нейронные сети»

Тема: «Датасет изображений Chars74K»

Студент гр. 7383 _____ Власов Р. А.

Студент гр. 7383 _____ Сычевский Р. А.

Студентка гр. 7383 _____ Ханова Ю. А.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

Цель работы.

В рамках выполнения ИДЗ необходимо разработать и реализовать модель ИНС решающую задачу распознавания символов на датасете Chars74K.

Требования.

- Модель должна быть разработана на языке Python с использованием Keras API;
- Исходный код проекта должен быть в формате PEP8;
- В исходном коде должны быть поясняющие комментарии;
- Модель не должна быть избыточной;
- Обучение модели должно быть стабильно.

Ход работы.

1. *Описание датасета*.

Датасет изображений. Включает в себя набор изображений символов (0-9,A-Z,a-z) полученных из фотографий.

Задача заключается в определении символа на изображении. Распознавание символов — это классическая проблема распознавания образов. Для латинского алфавита она, в значительной степени, считается решенной проблемой в частных ситуациях, таких как изображения отсканированных документов, содержащих шрифты с общими символами и равномерным фоном. Однако изображения, полученные с камер, по-прежнему представляют собой сложную задачу для распознавания символов. Сложные аспекты этой проблемы очевидны в этом наборе данных.

2. *Анализ данных.*

Набор данных состоит из:

- 62 классов, то есть арабские цифры от 0 до 9 и латинский алфавит в обоих регистрах A-Z, a-z;
- Около 7700 символов, полученных из естественных изображений;
- Около 3400 нарисованных с помощью планшетного ПК;
- Коло 64000 синтезированных из компьютерных шрифтов;

Это дает в общей сложности 78905 изображений.

Пример изображения из датасета представлен на рис.1.

Рисунок 1 — Изображение из датасета

Особенности, которые пришлось учесть при дальнейшей работе:

1) Встречаются изображения разного размера;
2) Изображений, синтезированных из компьютерных шрифтов на порядок больше, чем полученных с фотографий и рукописных, что может повлиять на работу сети в дальнейшем.

Целесообразно обучать сеть для распознавания символов, напечатанных шрифтами, так как количество примеров таких символов в наборе значительно (в 6 раз) превосходит количество всех остальных изображений.

3. Подготовка данных для обучения.

В файле settings.py были прописаны константы для загрузки данных и обучения сети. Среди таких констант ссылки на ресурсы, целевая директория, количество эпох обучения и другие.

После создания модели происходит загрузка данных. Функции для загрузки данных описаны в файле utils.py. Изначально скачиваются файлы (функция `download(url, path)`) и распаковываются полученные архивы

(функция `extract_file(path, to_dir='.')`). Для распаковки архивов была использована библиотека tarfile. Скачивание всех ресурсов осуществляется при помощи функции `download_resources()`. Ресурсы помещаются в директорию, указанную в файле с настройками.

Непосредственно загрузка данных в память осуществляется при помощи следующих функций:

`load_train_resources()` — Загрузка данных в память, возвращает изображения и метки;

`load_train_resources_low_RAM()` — Загрузка данных в память (версия для низкого потребления ОЗУ), возвращает список изображений и метки;

`get_labels_from_file(list)` — извлечение меток из файлов .m;

`get_names_from_file(list)` — извлечение имен из файлов .m.

Для считывания нужных данных из .m файлов используются регулярные выражения, описанные в константе `LIST_BORDERS`.

В процессе загрузки и подготовки данных выводится шкала прогресса, реализованная при помощи библиотеки tqdm.

4. *Процесс создания модели*

При построении модели, использовалась свертка, таким образом, мы снижаем разрешение картинки, оставляя при этом необходимую информацию. Для свертки использовались слои ZeroPadding2D, Conv2D и MaxPooling2D. После этого используется слой Conv2D с ядром свертки (1,1) для выравнивания данных. Подготока данных для Dense слоя осуществляется при помощи GlobalAveragePooling2D, так как картинки имеют разное разрешение. Так же используется слой Dropout для того чтобы избежать переобучения, то есть чтобы сеть хорошо работала не только на обучающей выборке, но и на других примерах. После этого идет Dense слой и Dense слой активации, выделяющий 62 класса. Для модели используется оптимизатор adadelta с начальной скоростью обучения 0,5.

Были добавлени колбэки `ModelCheckpoint` для сохранения модели в процессе обучения, `EarlyStopping` для ранней остановки на случай, если начнется переобучение сети, `ReduceLROnPlateau` для уменьшения скорости обучения, если результаты начали ухудшаться.

Максимальное количество эпох обучения было установлено в 100, 15% данных отводились для валидации.

Итоговая модель представлена на рис.2.

```
input_layer = Input(shape=(None, None, 3))
hidden_layer = ZeroPadding2D((2, 2))(input_layer)
hidden_layer = Conv2D(32, (3, 3), activation='relu', padding="same")(hidden_layer)
hidden_layer = MaxPooling2D(pool_size=(2, 2))(hidden_layer)
hidden_layer = ZeroPadding2D((2, 2))(hidden_layer)
hidden_layer = Conv2D(64, (3, 3), activation='relu', padding="same")(hidden_layer)
hidden_layer = MaxPooling2D(pool_size=(2, 2))(hidden_layer)
hidden_layer = ZeroPadding2D((2, 2))(hidden_layer)
hidden_layer = Conv2D(128, (3, 3), activation='relu', padding="same")(hidden_layer)
hidden_layer = MaxPooling2D(pool_size=(2, 2))(hidden_layer)
hidden_layer = ZeroPadding2D((2, 2))(hidden_layer)
hidden_layer = Conv2D(128, (3, 3), activation='relu', padding="same")(hidden_layer)
hidden_layer = MaxPooling2D(pool_size=(2, 2))(hidden_layer)
hidden_layer = Conv2D(256, (1, 1))(hidden_layer)
hidden_layer = GlobalAveragePooling2D()(hidden_layer)
hidden_layer = Dropout(0.5)(hidden_layer)
hidden_layer = Dense(8192, activation="relu")(hidden_layer)
output_layer = Dense(62, activation="softmax")(hidden_layer)
self.model = Model(inputs=input_layer, outputs=output_layer)
# for layer in self.model.layers[:3]:
# layer.trainable = False
self.model.compile(optimizer=Adadelta(learning_rate=0.5), loss="categorical_crossentropy", metrics=["accuracy"])
```

Рисунок 2 — Итоговая модель сети

Код программы представлен в приложении А.

В процессе построения модели возникли некоторые проблемы:

1) Изображения были разного размера.

Чтобы решить эту проблему, вместо конкретной размерности входных данных на первом слое указывается None, то есть input_shape=(None, None, 3). Также перед использованием Dense слоев используем слой GlobalAveragePooling2D.

2) Для обучения модели с таким количеством изображений необходимо очень много оперативной памяти.

Чтобы решить эту проблему, было принято решение не загружать сразу все картинки в оперативную память, а считывать с жесткого диска по мере необходимости. Были реализованы замены некоторых методов с

пометкой low_RAM. Таким образом, программа стала работать намного быстрее, так как не задействуется файл подкачки, однако такой вариант работы означает, что с жесткого диска постоянно будут считываться небольшие файлы, и при длительной работе в таком режиме, жесткий диск может быстро изнашиваться.

3) Некоторые изображения имеют очень маленькое разрешение.

Чтобы решить эту проблему, мы использовали слои ZeroPadding2D(), которые добавляют отступы из нулей вокруг изображения.

5. *Предсказания по обученной модели.*

Для предсказания работы обученной модели был создан интерфейс с помощью библиотеки Qt, предусматривающий выбор тестового изображения и модели. Вид окна интерфейса представлен на рис.3. Результаты тестирования представлены в приложении Б.

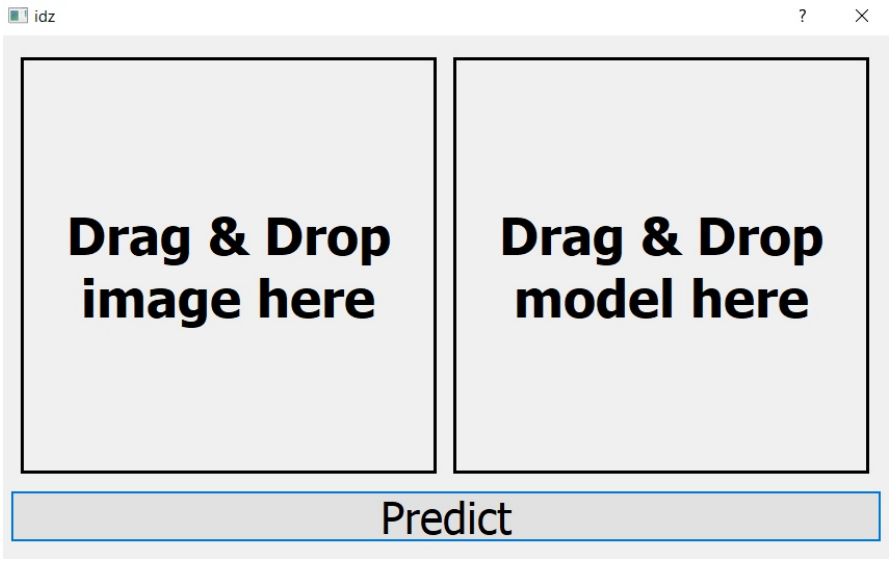

Рисунок 3 — Интерфейс предсказания результатов

6. *Сравнение разработанной модели с методами, решающими задачу и не относящимися к ИНС.*

В качестве метода распознования символа без использования искусственных нейронны сетей можно назвать сравнения изображения с подготовленным шаблоном. Данный метод имеет точность около 70%.

Однако в случаях, когда символ написан тем же шрифтом, что и шаблон, точность метода вырастает до 99%.

Точность распознавания данного метода можно считать примерно эквивалентной точности полученной нейронной сети. Однако скорость работы метода значительно уступает нейронной сети.

7. *Распределение зон ответственности в бригаде.*

   Власов Роман – предсказания с помощью обученной модели;

   Сычевский Радимир – создание и обучение модели;

   Ханова Юлия – обработка и подготовка данных для обучения.

Вывод.

В ходе выполнения лабораторной работы был изучен датасет Chars74K, содержащий шрифты, изображения символов с фотографий и рукописные символы. Была разработана модель искусственной нейронной сети с точностью 64% на всём датасете, что обусловливается малым содержанием фотографий и рукописных символов в датасете, относительно образцов шрифтов. Точность модели на шрифтах достигает 80%. По сравнению с методами, не использующими нейронную сеть для распознавания символов, точность можно считать эквивалентной, однако скорость распознавания значительно превосходит скорость таких методов.

ПРИЛОЖЕНИЕ А: ИСХОДНЫЙ КОД

train.py

```
import os
from utils import *
from settings import *
from model import RecognitionModel
# Создание модели
model = RecognitionModel()

if os.path.exists(SAVE_PATH_PREFIX + BACKUP_PATH): # В
случае существавания в данной директории модели
    print("Loading existing model...")
    model.load_model(SAVE_PATH_PREFIX + BACKUP_PATH) #
Загружается модель

images, targets = load_train_resources_low_RAM()
model.fit_model_low_RAM(images, targets)
```

evaluate.py

```
import os
from utils import *
from settings import *
from model import RecognitionModel

model = RecognitionModel()

if os.path.exists(SAVE_PATH_PREFIX + BACKUP_PATH):
    print("Loading existing model...")
    model.load_model(SAVE_PATH_PREFIX + BACKUP_PATH)
else:
    print("Warning! Model must be at '%s'. Evaluating
brand new model" % (SAVE_PATH_PREFIX + BACKUP_PATH))

images, targets = load_train_resources_low_RAM()
print(model.evaluate_low_RAM(images, targets))
```

model.py

```
import numpy as np
from keras.callbacks import ModelCheckpoint,
EarlyStopping, ReduceLROnPlateau
from keras.layers import Input, Dense, Dropout, Conv2D,
MaxPooling2D, \
    GlobalAveragePooling2D, ZeroPadding2D
from keras.models import Model
from keras.optimizers import Adadelta
from settings import *
from tqdm.keras import TqdmCallback
from utils import *
```

```
class RecognitionModel:
    model = None

    def __init__(self):
        input_layer = Input(shape=(None, None, 3))
        hidden_layer = ZeroPadding2D((2, 2))(input_layer)
        hidden_layer = Conv2D(32, (3, 3),
activation='relu', padding="same")(hidden_layer)
        hidden_layer = MaxPooling2D(pool_size=(2, 2))
(hidden_layer)
        hidden_layer = ZeroPadding2D((2, 2))
(hidden_layer)
        hidden_layer = Conv2D(64, (3, 3),
activation='relu', padding="same")(hidden_layer)
        hidden_layer = MaxPooling2D(pool_size=(2, 2))
(hidden_layer)
        hidden_layer = ZeroPadding2D((2, 2))
(hidden_layer)
        hidden_layer = Conv2D(128, (3, 3),
activation='relu', padding="same")(hidden_layer)
        hidden_layer = MaxPooling2D(pool_size=(2, 2))
(hidden_layer)
        hidden_layer = ZeroPadding2D((2, 2))
(hidden_layer)
        hidden_layer = Conv2D(128, (3, 3),
activation='relu', padding="same")(hidden_layer)
        hidden_layer = MaxPooling2D(pool_size=(2, 2))
(hidden_layer)
        hidden_layer = Conv2D(256, (1, 1))(hidden_layer)
        hidden_layer = GlobalAveragePooling2D()
(hidden_layer)
        hidden_layer = Dropout(0.5)(hidden_layer)
        hidden_layer = Dense(8192, activation="relu")
(hidden_layer)
        output_layer = Dense(62, activation="softmax")
(hidden_layer)
        self.model = Model(inputs=input_layer,
outputs=output_layer)
        # for layer in self.model.layers[:3]:
        # layer.trainable = False

self.model.compile(optimizer=Adadelta(learning_rate=0.5),
loss="categorical_crossentropy", metrics=["accuracy"])

    def predict(self, image):
        ans = self.model.predict(image)
        return get_character(np.argmax(ans))

    def get_model(self):
```

```
        return self.model

    def load_model(self, path):
        self.model.load_weights(path)

    def fit_model(self, X_train, Y_train):
        order = get_random_order(len(X_train))
        val = int(len(X_train) * VALIDATION)
        self.model.fit_generator(generate(X_train,
Y_train, order[val:]),
steps_per_epoch=len(X_train[val:]),

nb_epoch=EPOCHS_MAX, validation_steps=len(X_train[:val]),

validation_data=generate(X_train, Y_train, order[:val]),
verbose=0,

callbacks=self.get_callbacks())

    def fit_model_low_RAM(self, X_train, Y_train):
        print("WARNING! Using low RAM mode will lead to a
large number of read operations from your disk.")
        order = get_random_order(len(X_train))
        val = int(len(X_train) * VALIDATION)

self.model.fit_generator(generate_low_RAM(X_train,
Y_train, order[val:]),

steps_per_epoch=len(X_train[val:]),

nb_epoch=EPOCHS_MAX, validation_steps=len(X_train[:val]),

validation_data=generate_low_RAM(X_train, Y_train,
order[:val]), verbose=0,

callbacks=self.get_callbacks())

    def evaluate(self, X_train, Y_train):
        order = get_random_order(len(X_train))
        loss, acc =
self.model.evaluate_generator(generate(X_train, Y_train,
order), steps=len(X_train), verbose=1)
        return loss, acc

    def evaluate_low_RAM(self, X_train, Y_train):
        order = get_random_order(len(X_train))
        loss, acc =
self.model.evaluate_generator(generate_low_RAM(X_train,
Y_train, order), steps=len(X_train),

verbose=1)
```

```
        return loss, acc

    def get_callbacks(self):
        callbacks = []
        callbacks.append(TqdmCallback(verbose=1))
        callbacks.append(ModelCheckpoint(SAVE_PATH_PREFIX
+ MODEL_PATH, monitor='val_loss',

save_best_only=True, mode='min', verbose=1))

callbacks.append(EarlyStopping(monitor='val_loss',
mode='min', patience=5, verbose=1))

callbacks.append(ReduceLROnPlateau(monitor='val_loss',
mode='min', factor=0.3, patience=3, min_lr=0.001,
                                           verbose=1))
        return callbacks
```

utils.py

```
import os
import random
import re
import tarfile

import numpy as np
import requests
from PIL import Image
from settings import *
from tqdm import tqdm


# Скачивание файла
def download(url, path):
    chunk_size = 1024
    file = requests.get(url, stream=True)
    total_size = int(file.headers['content-length'])
    with open(path, 'wb') as f:
        for data in
tqdm(iterable=file.iter_content(chunk_size=chunk_size),
total=total_size // chunk_size + 1,
                             unit='KB'):
            f.write(data)


# Распаковка архива
def extract_file(path, to_dir='.'):
    with tarfile.open(name=path) as tar:
        for data in tqdm(iterable=tar.getmembers(),
total=len(tar.getmembers())):
            tar.extract(member=data, path=to_dir)
```

```
# Скачивание и подготовка всех необходимых ресурсов
def download_resources():
    for resource in RESOURCES:
        path = "%s%s" % (DOWNLOAD_DIRECTORY, resource)
        print("Downloading %s to %s" %
(RESOURCES.get(resource), path), flush=True)
        download(url=RESOURCES.get(resource), path=path)
        print("\nExtracting archive...", flush=True)
        extract_file(path=path,
to_dir=DOWNLOAD_DIRECTORY)
        os.remove(path)
        print("\nDone", flush=True)


# Проверка наличия ресурсов
def check_resources():
    if not os.path.exists(DOWNLOAD_DIRECTORY):
        os.mkdir(DOWNLOAD_DIRECTORY)
        return False
    for list in LISTS:
        if not os.path.exists(DOWNLOAD_DIRECTORY +
list[0]):
            return False
    return True


# Извлечение меток из файла .m
def get_labels_from_file(list):
    path = DOWNLOAD_DIRECTORY + list[0]
    with open(path) as f:
        content = f.read()
    labels = re.search(LIST_BORDERS[0][0], content,
re.DOTALL).group().replace(LIST_BORDERS[0][1],
"").replace(
        LIST_BORDERS[0][2], "").replace(";", "").split("\
n")
    return labels


# Извлечение имен из файла .m
def get_names_from_file(list):
    path = DOWNLOAD_DIRECTORY + list[0]
    with open(path) as f:
        content = f.read()
    names = re.search(LIST_BORDERS[1][0], content,
re.DOTALL).group().replace(LIST_BORDERS[1][1],
"").replace(
        LIST_BORDERS[1][2], "").replace(";",
"").replace("'", "").split("\n")
    for i in range(len(names)):
```

```
        names[i] = DOWNLOAD_DIRECTORY + list[1] +
names[i] + IMAGES_FORMAT
    return names


# Загрузка изображения в память
def load_image(path):
    image = Image.open(path).convert('RGB')
    image = np.asarray(image)
    return image


# Перемешивание изображений
def get_random_order(length):
    order = list(range(length))
    random.shuffle(order)
    return order


# Генератор изображений
def generate(X, Y, order):
    while 1:
        for i in order:
            img = np.reshape(X[i], (1, *(X[i].shape)))
            l = np.zeros(62)
            l[int(Y[i]) - 1] = 1
            l = l.reshape((1, 62))
            yield (img, l)


# Генератор изображений (низкое потребление ОЗУ)
def generate_low_RAM(X, Y, order):
    while 1:
        for i in order:
            img = load_image(X[i])
            img = np.reshape(img, (1, *(img.shape)))
            l = np.zeros(62)
            l[int(Y[i]) - 1] = 1
            l = l.reshape((1, 62))
            yield (img, l)


# Загрузка данных в память, возвращает список картинок и
меток
def load_train_resources():
    if not check_resources():
        download_resources()
    allLabels = list()
    allNames = list()
    for l in LISTS:
        allLabels += get_labels_from_file(l)
```

```
        allNames += get_names_from_file(l)
    images = []
    labels = []
    for i in tqdm(range(len(allNames)), desc="Loading
images: "):
        images.append(load_image(allNames[i]))
        labels.append(allLabels[i])
    return images, labels


# Загрузка данных (низкое потребление ОЗУ), возвращающая
список путей до изображений и меток
def load_train_resources_low_RAM():
    if not check_resources():
        download_resources()
    allLabels = list()
    allNames = list()
    for l in LISTS:
        allLabels += get_labels_from_file(l)
        allNames += get_names_from_file(l)
    return allNames, allLabels


# Восстановление символа его номеру
def get_character(num):
    if num < 10:
        return chr(ord('0') + num)
    elif num < 36:
        return chr(ord('A') + num - 10)
    else:
        return chr(ord('a') + num - 36)
```

settings.py

```
# Ссылки на архивы
RESOURCES = {
    "listsTXT.tar.gz": "http://www.ee.surrey.ac.uk/CVSSP/
demos/chars74k/ListsTXT.tgz",
    "img.tar.gz":
"http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/EnglishI
mg.tgz",
    "hnd.tar.gz":
"http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/EnglishH
nd.tgz",
    "fnt.tar.gz":
"http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/EnglishF
nt.tgz"
}

# Связь между списком изображений и папкой их хранения
LISTS = [
```

```
    ["list_English_Fnt.m", "English/Fnt/"],
    ["list_English_Hnd.m", "English/Hnd/"],
    ["list_English_Img.m", "English/Img/"]
]

LIST_PREFIXES = [
    ""
]

# Регулярные выражения для считывания нужных данных из .m
файлов
LIST_BORDERS = [[r'list.ALLlabels = \[.*?\n\]',
"list.ALLlabels = [", "\n]"],
                 [r'list.ALLnames = \[.*?\n\]',
"list.ALLnames = [", "\n]"]]

SAVE_PATH_PREFIX = ""
# Директория для загрузки датасета
DOWNLOAD_DIRECTORY = "./cache/"
IMAGES_FORMAT = ".png"

EPOCHS_MAX = 100
MODEL_PATH = "trained_model.hdf5"
BACKUP_PATH = "backup_model.hdf5"
VALIDATION = 0.15
```

predict.py

```
import sys
from model import RecognitionModel
from utils import *
import numpy as np
from PySide2.QtCore import Qt
from PySide2.QtGui import QPixmap
from PySide2.QtWidgets import QApplication, QDialog,
QHBoxLayout, QLabel, QPushButton, QVBoxLayout,
QMessageBox
from sys import platform

class MainWindow(QDialog):
    drop_image_text = "Drag & Drop\nimage here"
    drop_model_text = "Drag & Drop\nmodel here"
    predict_button_text = "Predict"
    usual_stylesheet = "border: 3px solid black; color:
black"
    has_model_stylesheet = "border: 3px solid black;
color: black; font-size: 200px"
    dragged_stylesheet = "border: 3px solid blue; color:
blue"
    dragged_has_model_stylesheet = "border: 3px solid
blue; color: blue; font-size: 200px"
```

```
    error_stylesheet = "border: 3px solid red; color:
red"
    image_box_size = 400
    model_box_size = 400
    predict_button_height = 50
    indents_size = 20
    image_path = None
    model_path = None

    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.model = RecognitionModel()
        self.setFixedSize(self.image_box_size +
self.model_box_size + 3 * self.indents_size,
                          self.image_box_size +
self.predict_button_height + 3 * self.indents_size)
        self.setWindowTitle("idz")
        self.setAcceptDrops(True)

        self.image_label = self.get_drop_image_label()
        self.model_label = self.get_drop_model_label()
        self.predict_button = self.get_predict_button()

        layout = QHBoxLayout()
        layout.addWidget(self.image_label)
        layout.addWidget(self.model_label)

        window_layout = QVBoxLayout()
        window_layout.addLayout(layout)
        window_layout.addWidget(self.predict_button)

        self.setLayout(window_layout)

    def show_prediction(self, prediction):
        message = QMessageBox(QMessageBox.NoIcon,
"Prediction", prediction)
        message.layout().setMargin(0)
        message.layout().setSpacing(0)
        message.setStyleSheet("QLabel{max-width: 95px;
min-height: 100px; font-size:100px; font-weight: bold;}"
                              "QPushButton{min-width:
120px;min-height: 30px; font-size:25px;}")
        message.exec_()

    def predict(self):
        if self.image_path is not None and
self.model_path is not None:
            img = load_image(self.image_path)
            img = np.reshape(img, (1, *(img.shape)))
            prediction = self.model.predict(img)
            self.show_prediction(prediction)
```

```
            return
        if self.image_path is None:

self.image_label.setStyleSheet(self.error_stylesheet)
        if self.model_path is None:

self.model_label.setStyleSheet(self.error_stylesheet)

    def get_drop_model_label(self):
        label = QLabel(self.drop_model_text)
        label.setAlignment(Qt.AlignCenter)
        label.setAutoFillBackground(True)
        font = label.font()
        font.setPointSize(30)
        font.setBold(True)
        label.setFont(font)
        label.setFixedSize(self.model_box_size,
self.model_box_size)
        label.setStyleSheet(self.usual_stylesheet)
        return label

    def get_drop_image_label(self):
        label = QLabel(self.drop_image_text)
        label.setAlignment(Qt.AlignCenter)
        label.setAutoFillBackground(True)
        font = label.font()
        font.setPointSize(30)
        font.setBold(True)
        label.setFont(font)
        label.setFixedSize(self.image_box_size,
self.image_box_size)
        label.setStyleSheet(self.usual_stylesheet)
        return label

    def get_predict_button(self):
        btn = QPushButton(self.predict_button_text)
        btn.setFixedHeight(self.predict_button_height)
        font = btn.font()
        font.setPointSize(25)
        btn.setFont(font)
        btn.clicked.connect(self.predict)
        return btn

    def dragEnterEvent(self, e):
        if e.mimeData().hasUrls() and
len(e.mimeData().urls()) == 1:
            e.acceptProposedAction()

    def dragMoveEvent(self, e):
        c = self.childAt(e.pos())
        if c is not None:
```

```
            if c is self.image_label:

self.image_label.setStyleSheet(self.dragged_stylesheet)
                self.model_label.setStyleSheet(
                    self.usual_stylesheet if
self.model_path is None else self.has_model_stylesheet)

            elif c is self.model_label:

self.image_label.setStyleSheet(self.usual_stylesheet)

self.model_label.setStyleSheet(self.dragged_stylesheet if
self.model_path is None else
self.dragged_has_model_stylesheet)
            else:

self.image_label.setStyleSheet(self.usual_stylesheet)
                self.model_label.setStyleSheet(
                    self.usual_stylesheet if
self.model_path is None else self.has_model_stylesheet)


        else:

self.image_label.setStyleSheet(self.usual_stylesheet)
            self.model_label.setStyleSheet(
                self.usual_stylesheet if self.model_path
is None else self.has_model_stylesheet)

    def dragLeaveEvent(self, e):

self.image_label.setStyleSheet(self.usual_stylesheet)

self.model_label.setStyleSheet(self.usual_stylesheet if
self.model_path is None else self.has_model_stylesheet)

    def dropEvent(self, e):

self.image_label.setStyleSheet(self.usual_stylesheet)

self.model_label.setStyleSheet(self.usual_stylesheet if
self.model_path is None else self.has_model_stylesheet)
        c = self.childAt(e.pos())
        url = e.mimeData().urls()[0].path()
        if platform == "win32":
            url = '//'.join(url.rsplit('/', 1))
            if url.startswith("/"):
                url = url[1:]
        if c is not None:
            if c is self.image_label and
url.endswith(".png"):
```

```
                self.image_path = url

self.image_label.setPixmap(QPixmap(self.image_path).scale
d(400, 400, Qt.KeepAspectRatio))
            elif c is self.model_label and
url.endswith(".hdf5"):
                self.model_path = url
                self.model_label.setText('✓')

self.model_label.setStyleSheet(self.has_model_stylesheet)
                self.model.load_model(url)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()
```

ПРИЛОЖЕНИЕ Б: ПРИМЕРЫ ТЕСТИРОВАНИЯ МОДЕЛИ

Результаты тестирования представлены на рис. 4-13.

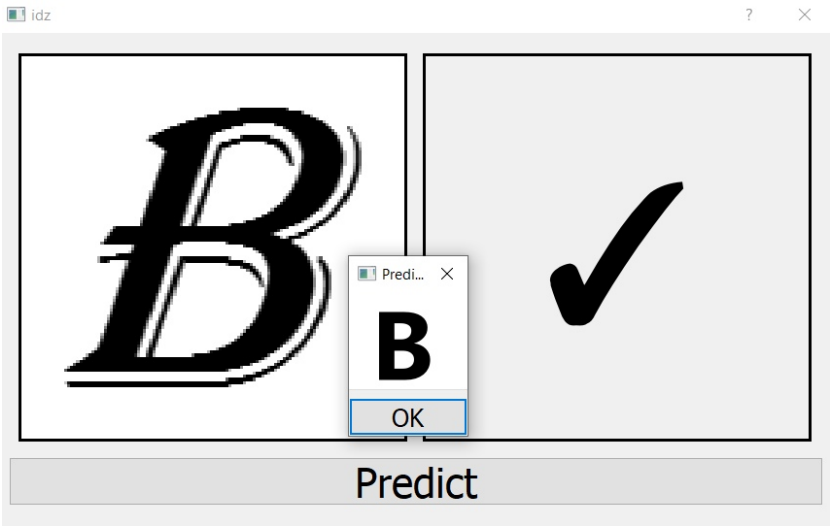

Рисунок 4 — Символ «B», определен верно.

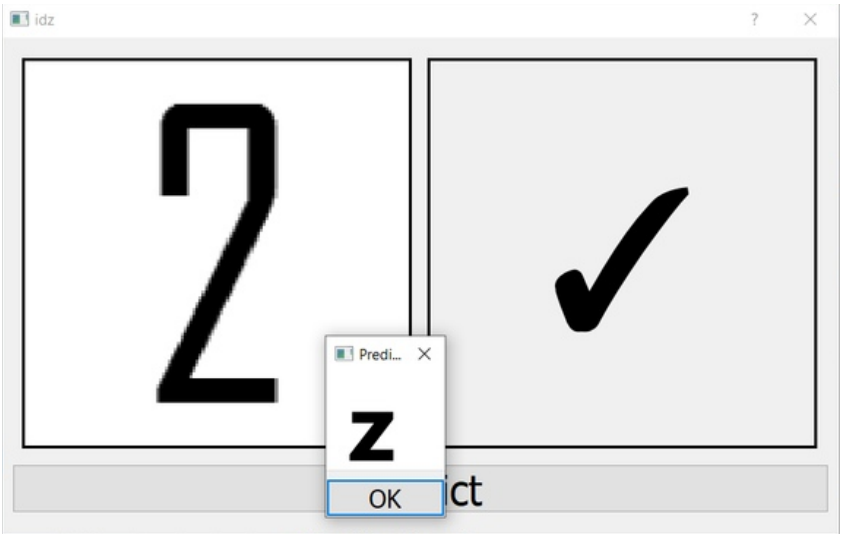

Рисунок 5 — Символ «2», определен как «z»

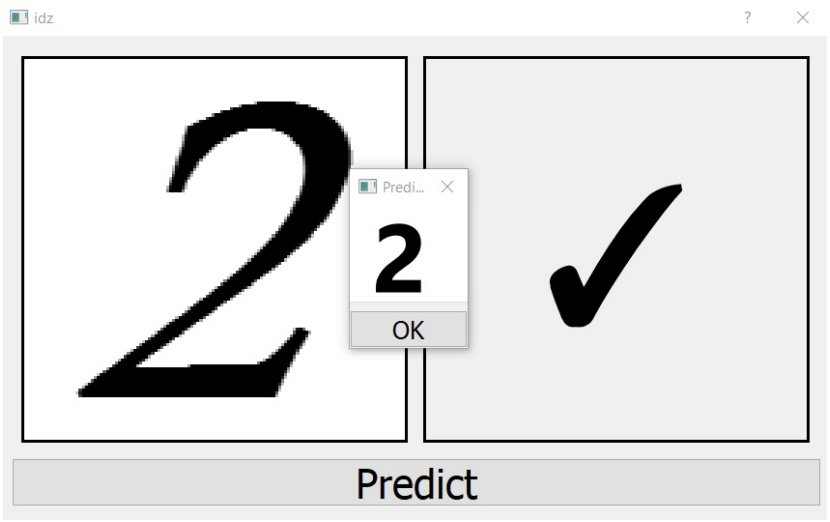

Рисунок 6 — Символ «2», определен верно.

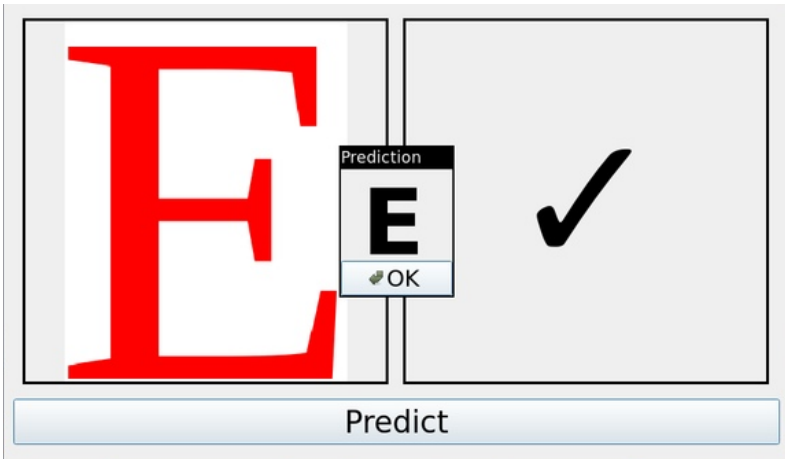

Рисунок 7 — Символ «E», определен верно.

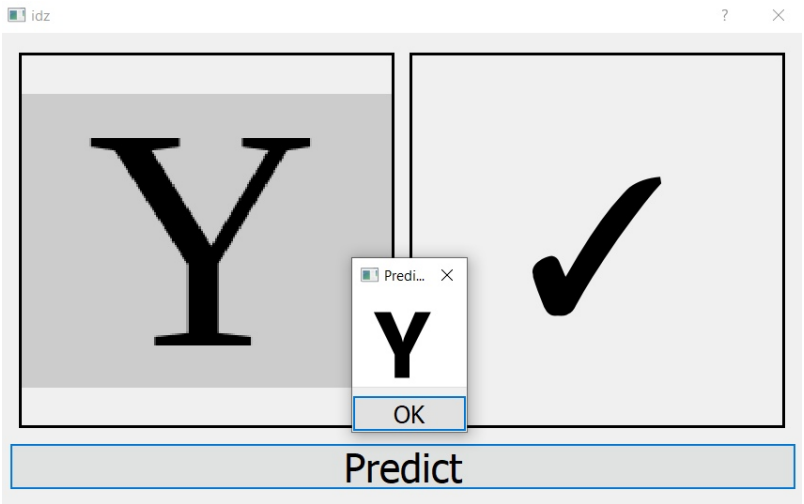

Рисунок 8 — Символ «Y», определен верно.

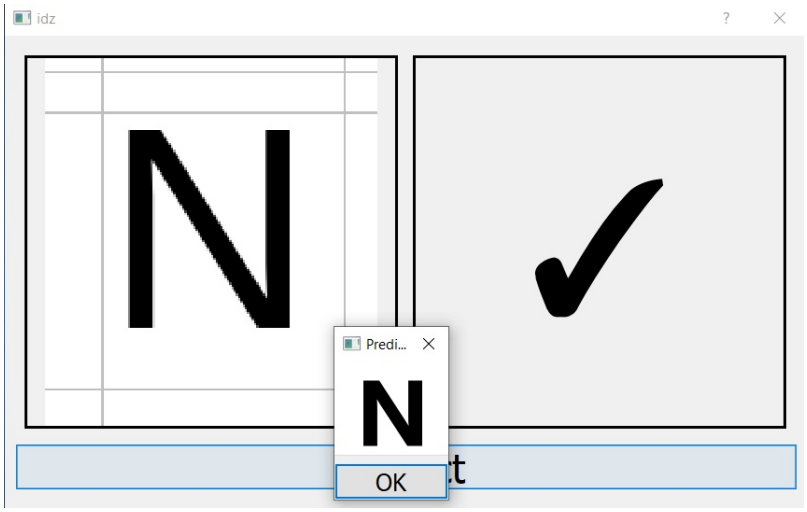

Рисунок 9 — Символ «N», определен верно.

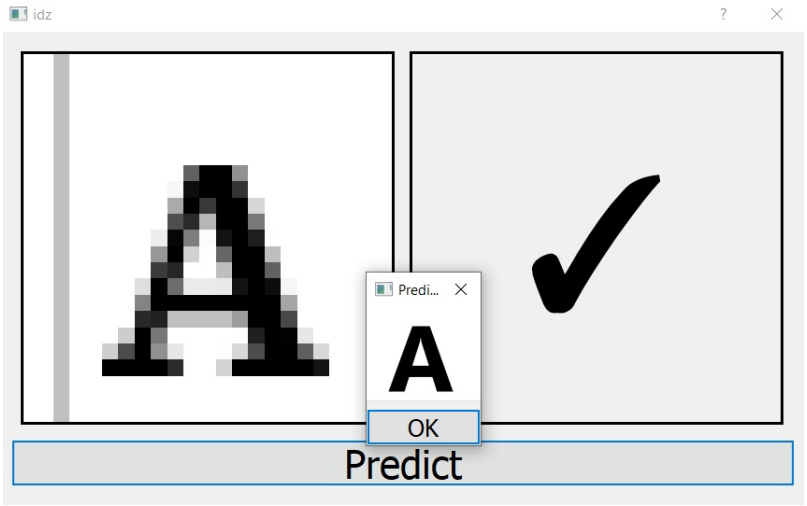

Рисунок 10 — Символ «A», определен верно.

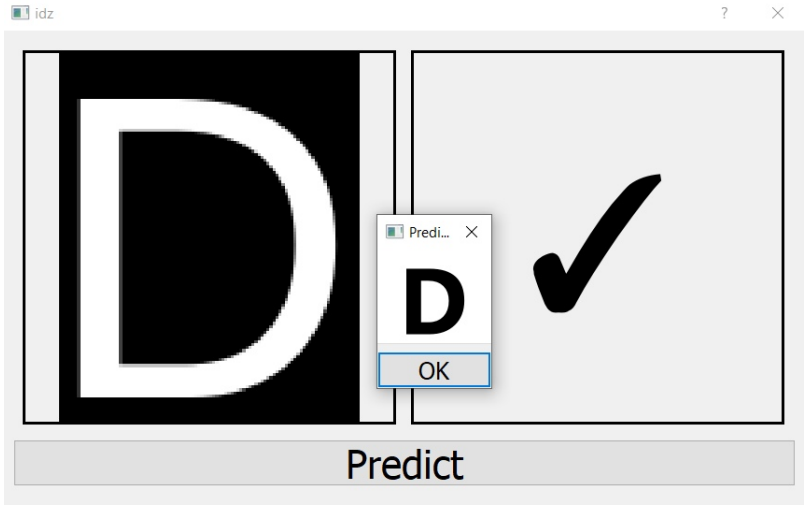

Рисунок 11 — Символ «D», определен верно.

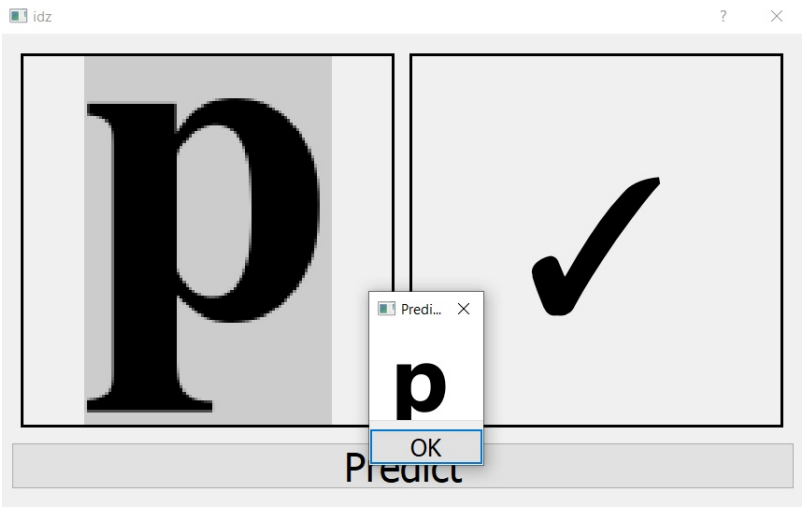

Рисунок 12 — Символ «p», определен верно.

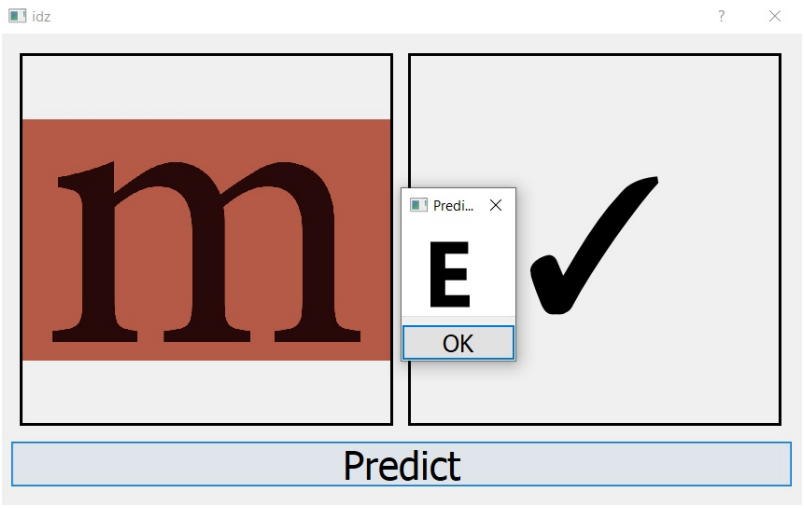

Рисунок 13 — Символ «m», определен как «E».