Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационные системы и анализ данных

Центр программной инженерии

**ОТЧЁТ**

к лабораторной работе №2 по дисциплине:

|  |
| --- |
| «Нейросетевые технологии» |
| Создание простейшей искусственной нейронной сети для решения задачи распознавания образов |
| наименование темы |

Выполнилстудент ИСТб-21-1 Д.И. Морозов

номер группы подпись И. О. Фамилия

дата

Проверил Доцент Е.А. Осипова

Должность подпись И. О. Фамилия

дата

Иркутск – 2023 г.

# 1 Постановка задачи

# Цель работы: изучение и программная реализация простейшей искусственной нейронной сети для решения задачи распознавания образов.

# Задание: В лабораторной работе исследуется двухслойная искусственная нейронная сеть (см. рис. 7), применяемая для распознавания простейших графических образов – букв и цифр. В качестве входных векторов в данном случае выступает изображение, преобразуемое во входной вектор по следующему принципу:

# изображение, содержащее символ, преобразуется в бинарный формат;

# полученная матрица построчно перестраивается в вектор.

# Ход работы: Обучение разработанной нейронной сети для всех вариантов осуществляется с помощью алгоритма обратного распространения ошибки. Количество нейронов в входном и скрытом слоях, а также число входных и выходных переменных автор лабораторной работы выбирает самостоятельно. В ходе работы должны быть сформированы обучающая и тестовая выборки. Причем в тестовой выборке используются символы из различных шрифтов, а также зашумленные и намеренно искаженные символы. В результате работы должна быть спроектирована и реализована искусственная нейронная сеть, обученная для распознавания образов в соответствии с вариантом. Программная реализация может быть на любом языке программирования, по усмотрению автора работы, при этом использование любых сторонних библиотек для реализации нейронных сетей

# Вариант 2: Сигмоидальная функция активации, α =1 0 T = 0 Строчные русские буквы от а до я.

# 2.1 Подготовка и проверка данных

Импорт библиотек для дальнейшей работы.

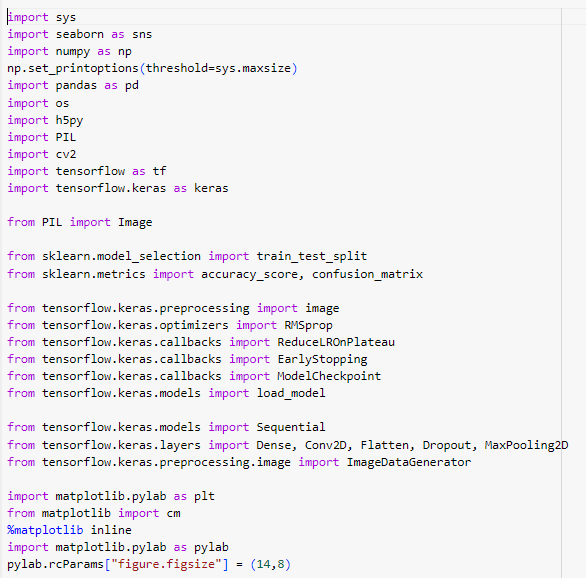


Рисунок 1 – Импорт библиотек

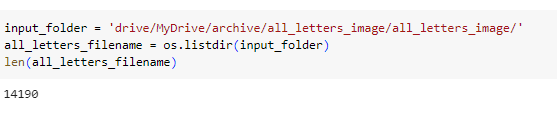


Рисунок 2 – Подключение датесета

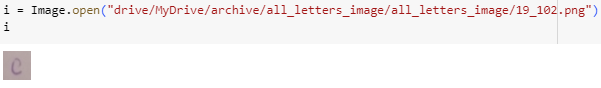


Рисунок 3 – Проверка случайного изображения

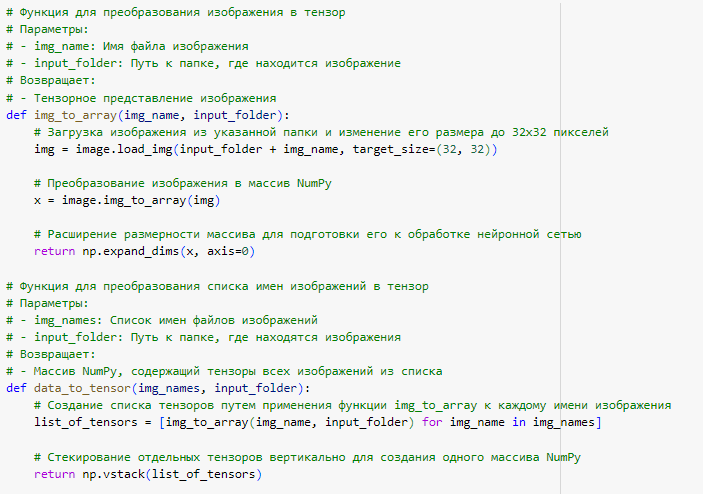


Рисунок 4 – Загрузка изображений и преобразование в тензоры

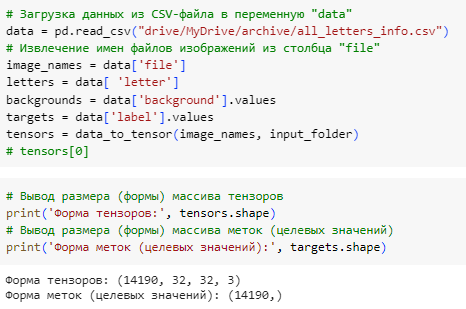


Рисунок 5 – Загрузка меток и создание тензоров

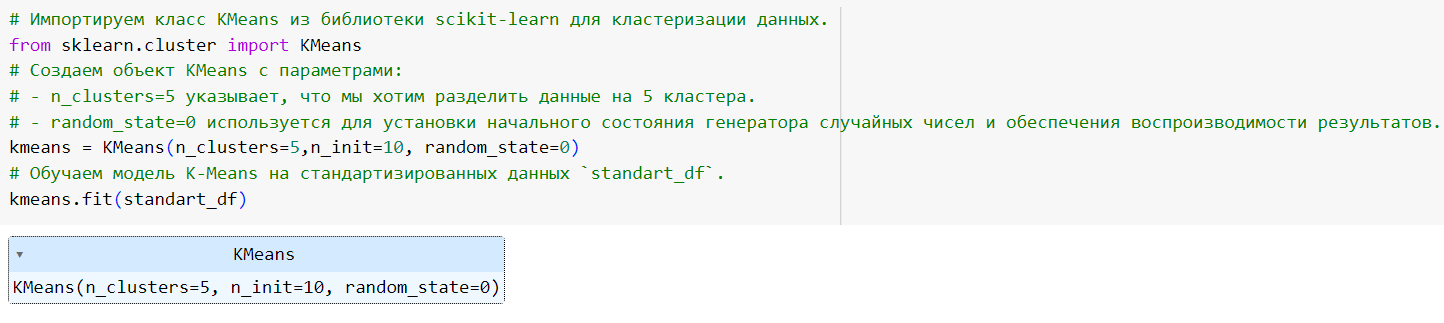
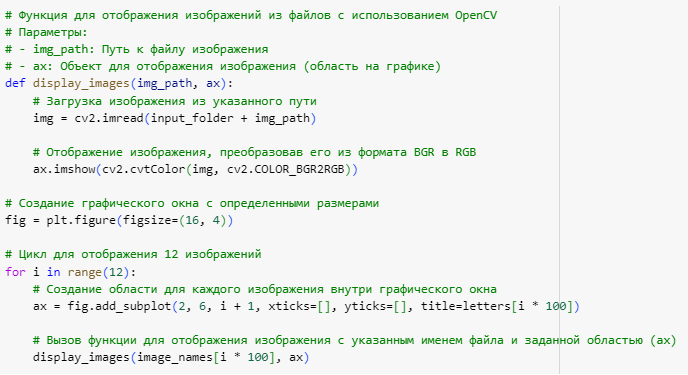


Рисунок 6 – Вывод случайных изображений из выборки

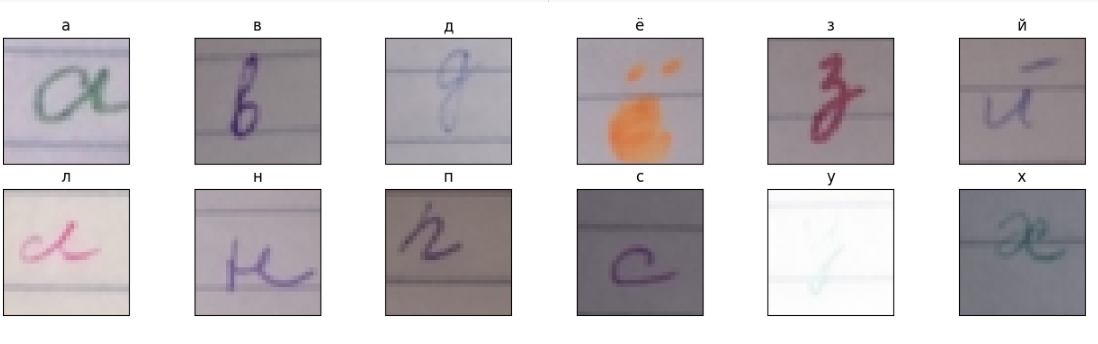


Рисунок 7 – Случайные изображения



Рисунок 8 – Вывод всего алфавита из датасета

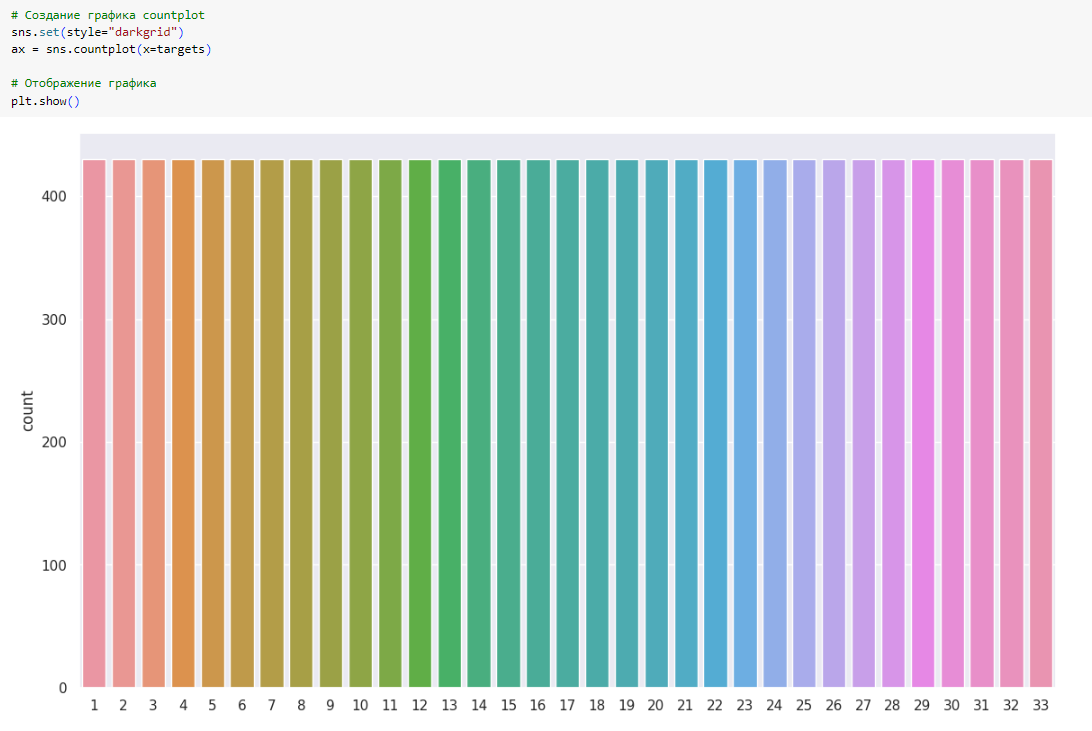


Рисунок 9 – Проверка данных на равномерное распределение по буквам

# 3 Работа с данными

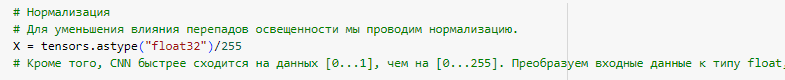


Рисунок 10 – Нормализация



Рисунок 11 – Нормализованное изображение

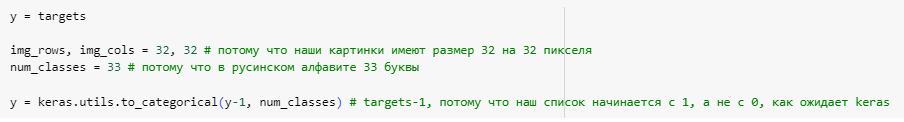


Рисунок 12 – Создание one\_hot меток

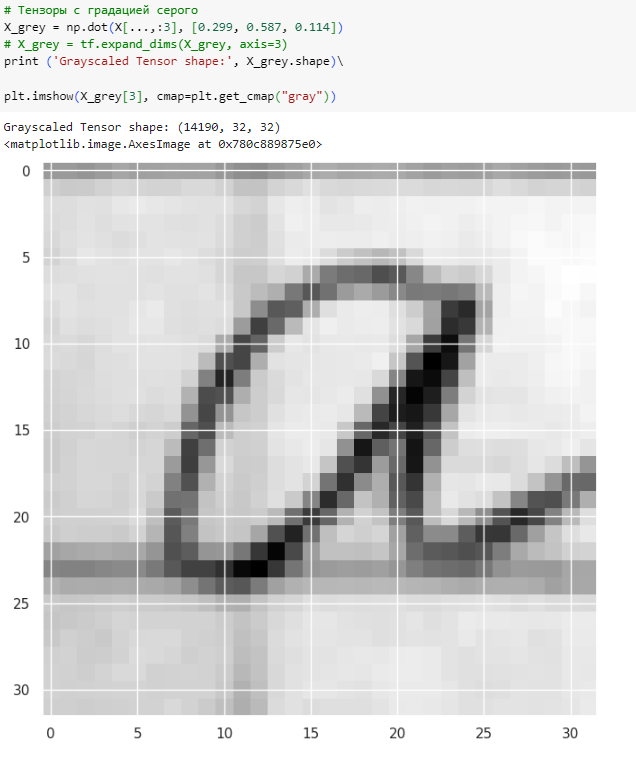


Рисунок 13 – Перевод в градации серого

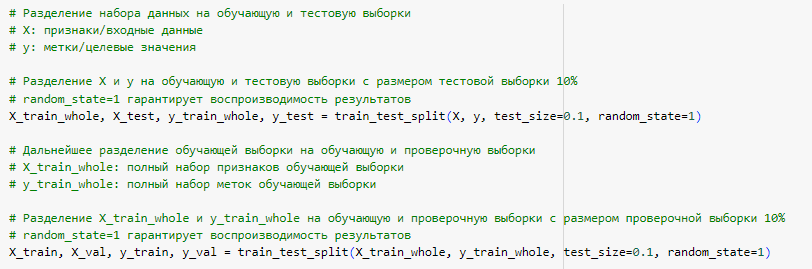


Рисунок 14 – Разделение данных на выборки

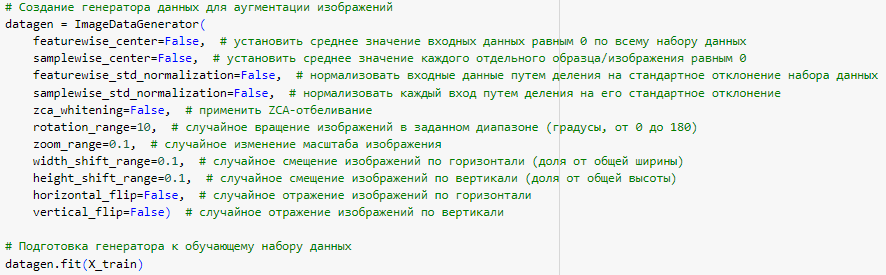


Рисунок 15 –Код для аугментации изображений

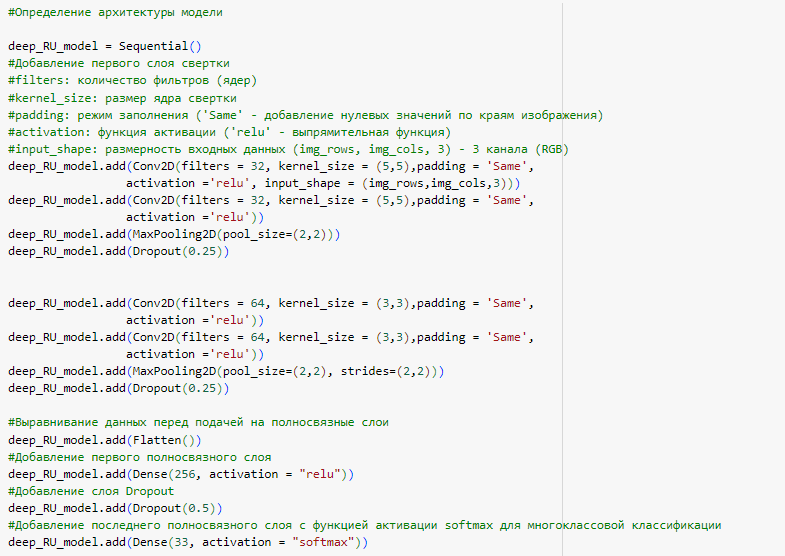


Рисунок 16 – Определение архитектуры модели м использованием Keras

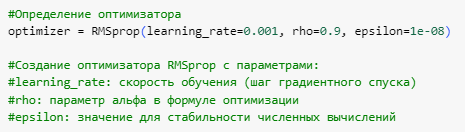


Рисунок 17 – Определение оптимизатора

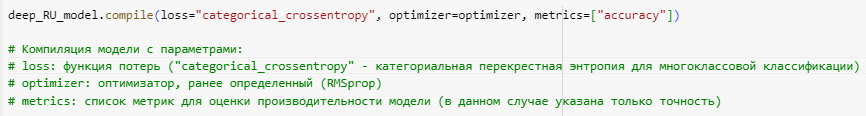


Рисунок 18 – Компиляция модели с использованием keras

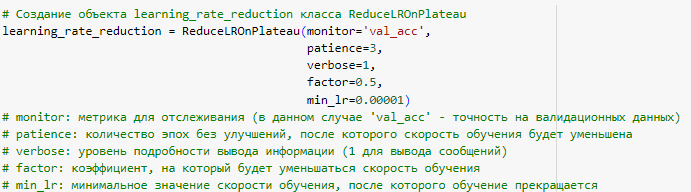


Рисунок 19 – (Сallback) для выполнения дополнительных действий на основе различных событий, таких как улучшение или ухудшение производительности модели

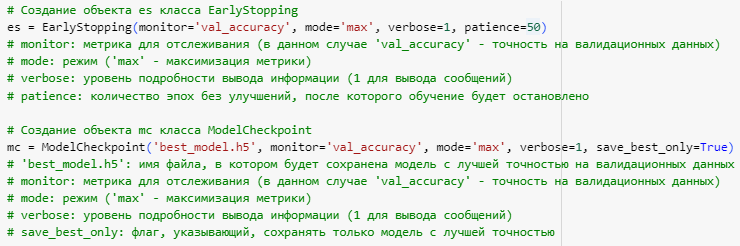


Рисунок 20 – Отслеживание и сохранение лучшей модели

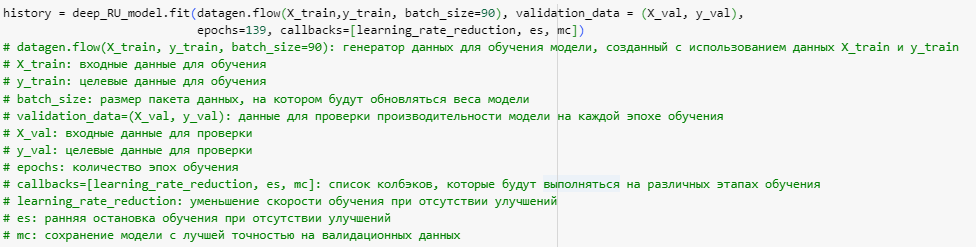


Рисунок 21 –Обучение модели на созданных данных

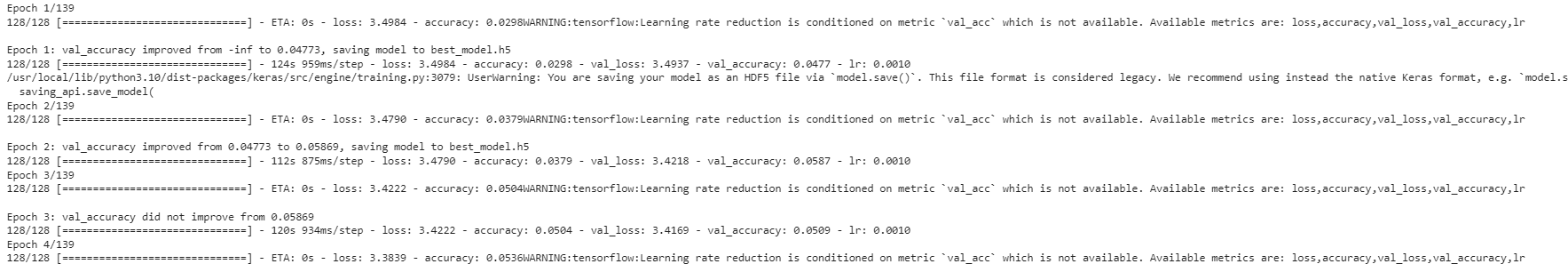


Рисунок 22 – Начало обучения модели

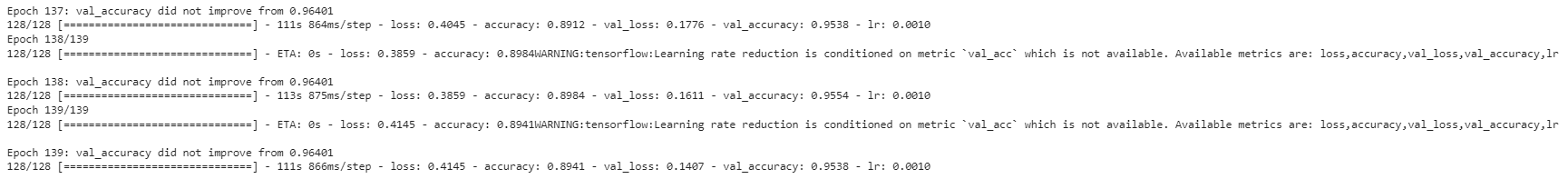


Рисунок 23 –Конец обучения модели



Рисунок 24 – Начальные значения при запуске обучения



Рисунок 25 – Значения после обучения

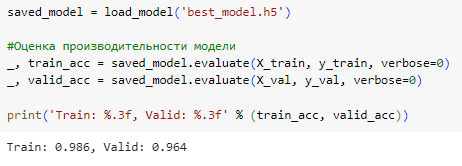


Рисунок 26 – Загрузка модели и её оценка

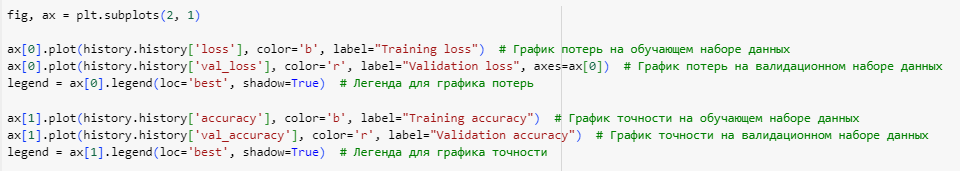


Рисунок 27 – Код отрисовки графиков потерь и точности

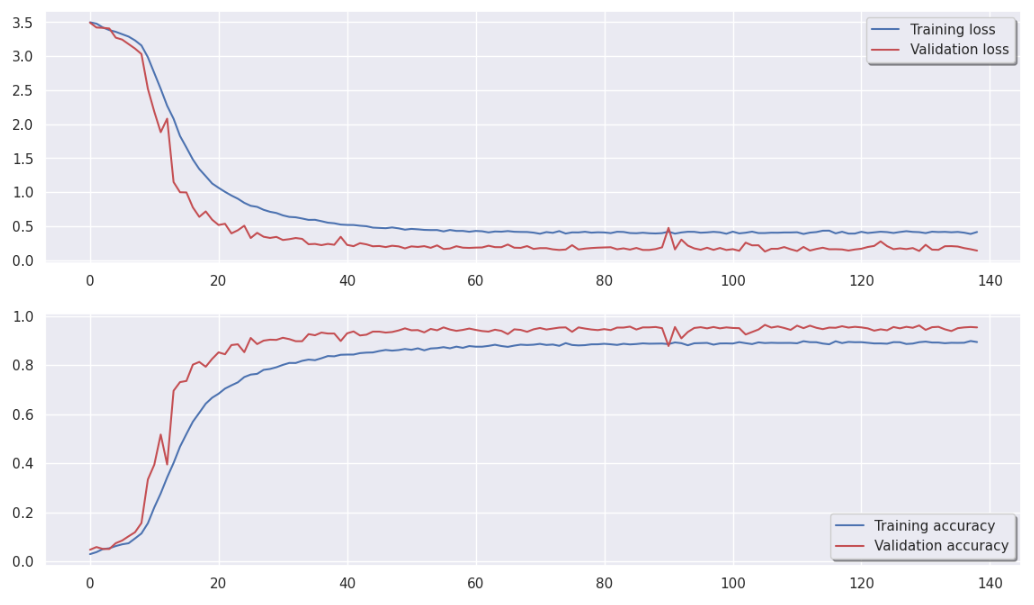


Рисунок 28 – Графики потерь и точности

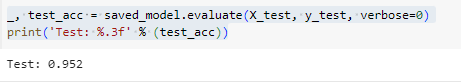


Рисунок 29 – Оценка производительности сохраненной модели

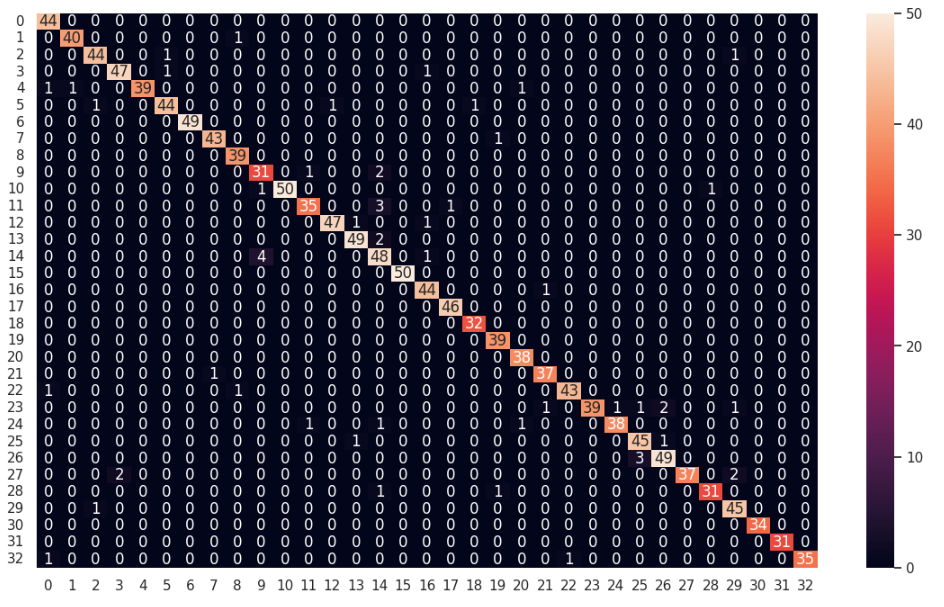


Рисунок 30 – Тепловая карта построенная на основе предсказанных меток



Рисунок 31 – Код для вывода ошибок

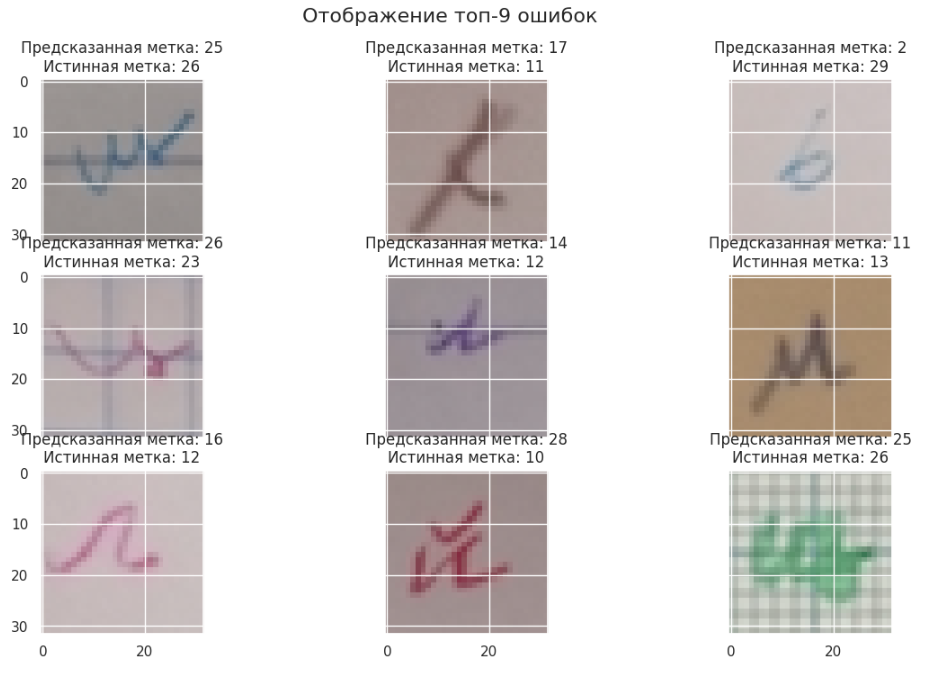


Рисунок 32 – Истинные и предсказанные метки

В процессе обработки данных мы сместили порядок букв на 1, чтобы он начинался с 0. Новый порядок выглядит следующим образом:

а=>0, б=>1, в=>2, г=>3, д=>4, е=>5, ё=>6, ж=>7

з=>8, и=>9, й=>10, к=>11, л=>12, м=>13, н=>14

о=>15, п=>16, р=>17, с=>18, т=>19, у=>20, ф=>21

х=>22, ц=>23, ч=>24, ш=>25, щ=>26, ъ=>27, ы=>28

ь=>29, э=>30, ю=>31, я=>32

Большинство ошибок, особенно во втором ряду, легко может быть сделано человеком таким же образом.

3 Результаты работы

В результате выполнения лабораторной работы была разработана и реализована двухслойная искусственная нейронная сеть (ANN) для распознавания строчных русских букв от "а" до "я". Нейронная сеть использовала сигмоидальную функцию активации и обучалась с использованием алгоритма обратного распространения ошибки. Шаги работы включали в себя подготовку данных, создание модели с учетом параметров, таких как количество нейронов в слоях и выбор функций активации, а также обучение и тестирование модели на соответствующих выборках. В тестовую выборку были включены символы из различных шрифтов, а также зашумленные и искаженные символы для оценки устойчивости модели. Анализ результатов показал, что разработанная нейронная сеть способна успешно распознавать строчные русские буквы. Эффективность модели была проверена на разнообразных входных данных, включая вариации шрифтов и искаженные символы. Полученные результаты могут быть использованы в качестве основы для дальнейших исследований и улучшений в области распознавания образов с использованием нейронных сетей.

# Приложение

import sys

import seaborn as sns

import numpy as np

np.set\_printoptions(threshold=sys.maxsize)

import pandas as pd

import os

import h5py

import PIL

import cv2

import tensorflow as tf

import tensorflow.keras as keras

from PIL import Image

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

from tensorflow.keras.preprocessing import image

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.callbacks import ReduceLROnPlateau

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.models import load\_model

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pylab as plt

from matplotlib import cm

%matplotlib inline

import matplotlib.pylab as pylab

pylab.rcParams["figure.figsize"] = (14,8)

input\_folder = 'drive/MyDrive/archive/all\_letters\_image/all\_letters\_image/'

all\_letters\_filename = os.listdir(input\_folder)

len(all\_letters\_filename)

#Это одно из наших изображений. Размер каждого изображения составляет 32 на 32 пикселя.

#Затем мы преобразуем каждое изображение в 3d-массив numpy.

i = Image.open("drive/MyDrive/archive/all\_letters\_image/all\_letters\_image/20\_102.png")

i

i\_arr = np.array(i)

# i\_arr

# Все 32 матрицы внутри этого массива представляют собой одно изображение.

# Каждая матрица представляет собой 1 строку этого изображения.

# Длина одной строки изображения составляет 32 пикселя, поэтому каждая матрица имеет 32 строки.

# Каждая строка матрицы имеет 4 столбца и представляет собой 1 пиксель. Для этого пикселя каждый столбец представляет собой значения цвета - насколько он красный, зеленый и синий - плюс непрозрачность цветов (последний столбец).

# Поэтому каждая матрица имеет размер 32 на 4. Общее количество пикселей в одном изображении составляет 32 \* 32 = 1024.

# Каждое значение цвета находится в диапазоне [0:255].

# Это означает, что для каждого цвета существует 256 оттенков. В сумме все комбинации этих цветов дают нам 256ˆ3 = 16 777 216 возможных цветов.

from keras.preprocessing import image

import numpy as np

# Функция для преобразования изображения в тензор

# Параметры:

# - img\_name: Имя файла изображения

# - input\_folder: Путь к папке, где находится изображение

# Возвращает:

# - Тензорное представление изображения

def img\_to\_array(img\_name, input\_folder):

# Загрузка изображения из указанной папки и изменение его размера до 32x32 пикселей

img = image.load\_img(input\_folder + img\_name, target\_size=(32, 32))

# Преобразование изображения в массив NumPy

x = image.img\_to\_array(img)

# Расширение размерности массива для подготовки его к обработке нейронной сетью

return np.expand\_dims(x, axis=0)

# Функция для преобразования списка имен изображений в тензор

# Параметры:

# - img\_names: Список имен файлов изображений

# - input\_folder: Путь к папке, где находятся изображения

# Возвращает:

# - Массив NumPy, содержащий тензоры всех изображений из списка

def data\_to\_tensor(img\_names, input\_folder):

# Создание списка тензоров путем применения функции img\_to\_array к каждому имени изображения

list\_of\_tensors = [img\_to\_array(img\_name, input\_folder) for img\_name in img\_names]

# Стекирование отдельных тензоров вертикально для создания одного массива NumPy

return np.vstack(list\_of\_tensors)

# Загрузка данных из CSV-файла в переменную "data"

data = pd.read\_csv("drive/MyDrive/archive/all\_letters\_info.csv")

# Извлечение имен файлов изображений из столбца "file"

image\_names = data['file']

letters = data[ 'letter']

backgrounds = data['background'].values

targets = data['label'].values

tensors = data\_to\_tensor(image\_names, input\_folder)

# tensors[0]

# Вывод размера (формы) массива тензоров

print('Форма тензоров:', tensors.shape)

# Вывод размера (формы) массива меток (целевых значений)

print('Форма меток (целевых значений):', targets.shape)

# Функция для отображения изображений из файлов с использованием OpenCV

# Параметры:

# - img\_path: Путь к файлу изображения

# - ax: Объект для отображения изображения (область на графике)

def display\_images(img\_path, ax):

# Загрузка изображения из указанного пути

img = cv2.imread(input\_folder + img\_path)

# Отображение изображения, преобразовав его из формата BGR в RGB

ax.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

# Создание графического окна с определенными размерами

fig = plt.figure(figsize=(16, 4))

# Цикл для отображения 12 изображений

for i in range(12):

# Создание области для каждого изображения внутри графического окна

ax = fig.add\_subplot(2, 6, i + 1, xticks=[], yticks=[], title=letters[i \* 50])

# Вызов функции для отображения изображения с указанным именем файла и заданной областью (ax)

display\_images(image\_names[i \* 50], ax)

# Создание графика countplot

sns.set(style="darkgrid")

ax = sns.countplot(x=targets)

# Отображение графика

plt.show()

X = tensors.astype("float32")/255

arr = X[2]

arr\_ = np.squeeze(arr)

plt.imshow(arr\_)

plt.show()

targets[0]

y = targets

img\_rows, img\_cols = 32, 32 # потому что наши картинки имеют размер 32 на 32 пикселя

num\_classes = 33 # потому что в русинском алфавите 33 буквы

y = keras.utils.to\_categorical(y-1, num\_classes)

print(X.shape)

print(y.shape)

def captch\_ex(file\_name):

img = cv2.imread(file\_name)

img\_final = cv2.imread(file\_name)

img2gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, mask = cv2.threshold(img2gray, 180, 255, cv2.THRESH\_BINARY)

image\_final = cv2.bitwise\_and(img2gray, img2gray, mask=mask)

ret, new\_img = cv2.threshold(image\_final, 180, 255, cv2.THRESH\_BINARY) # для черного текста использовать cv2.THRESH\_BINARY\_INV

'''

line 8 to 12 : Remove noisy portion

'''

kernel = cv2.getStructuringElement(cv2.MORPH\_CROSS, (3,

3)) # для манипуляции ориентацией расширения, большое значение x означает большее горизонтальное расширение, большое значение y означает большее вертикальное расширение

dilated = cv2.dilate(new\_img, kernel, iterations=9) # расширяем, чем больше итераций, тем больше расширение

# for cv2.x.x

contours, hierarchy = cv2.findContours(new\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE) # findContours возвращает 3 переменные для получения контуров

for contour in contours:

# получаем ограничивающий прямоугольник контура

[x, y, w, h] = cv2.boundingRect(contour)

# Не рисуем маленькие контуры, которые не являются текстом

if w < 35 and h < 35:

continue

# рисуем прямоугольник вокруг контура на исходном изображении

cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 255), 2)

'''

#you can crop image and send to OCR , false detected will return no text :)

cropped = img\_final[y :y + h , x : x + w]

s = file\_name + '/crop\_' + str(index) + '.png'

cv2.imwrite(s , cropped)

index = index + 1

'''

# записываем исходное изображение с добавленными контурами на диск

cv2.imshow('captcha\_result', img)

cv2.waitKey()

file\_name = 'drive/MyDrive/archive/all\_letters\_image/all\_letters\_image/04\_100.png'

# captch\_ex(file\_name)

# Тензоры с градацией серого

X\_grey = np.dot(X[...,:3], [0.299, 0.587, 0.114])

# X\_grey = tf.expand\_dims(X\_grey, axis=3)

print ('Grayscaled Tensor shape:', X\_grey.shape)

plt.imshow(X\_grey[3], cmap=plt.get\_cmap("gray"))

# Разделение набора данных на обучающую и тестовую выборки

# X: признаки/входные данные

# y: метки/целевые значения

# Разделение X и y на обучающую и тестовую выборки с размером тестовой выборки 10%

# random\_state=1 гарантирует воспроизводимость результатов

X\_train\_whole, X\_test, y\_train\_whole, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=1)

# Дальнейшее разделение обучающей выборки на обучающую и проверочную выборки

# X\_train\_whole: полный набор признаков обучающей выборки

# y\_train\_whole: полный набор меток обучающей выборки

# Разделение X\_train\_whole и y\_train\_whole на обучающую и проверочную выборки с размером проверочной выборки 10%

# random\_state=1 гарантирует воспроизводимость результатов

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_whole, y\_train\_whole, test\_size=0.1, random\_state=1)

# Создание генератора данных для аугментации изображений

datagen = ImageDataGenerator(

featurewise\_center=False, # установить среднее значение входных данных равным 0 по всему набору данных

samplewise\_center=False, # установить среднее значение каждого отдельного образца/изображения равным 0

featurewise\_std\_normalization=False, # нормализовать входные данные путем деления на стандартное отклонение набора данных

samplewise\_std\_normalization=False, # нормализовать каждый вход путем деления на его стандартное отклонение

zca\_whitening=False, # применить ZCA-отбеливание

rotation\_range=10, # случайное вращение изображений в заданном диапазоне (градусы, от 0 до 180)

zoom\_range=0.1, # случайное изменение масштаба изображения

width\_shift\_range=0.1, # случайное смещение изображений по горизонтали (доля от общей ширины)

height\_shift\_range=0.1, # случайное смещение изображений по вертикали (доля от общей высоты)

horizontal\_flip=False, # случайное отражение изображений по горизонтали

vertical\_flip=False) # случайное отражение изображений по вертикали

# Подготовка генератора к обучающему набору данных

datagen.fit(X\_train)

#Определение архитектуры модели

deep\_RU\_model = Sequential()

#Добавление первого слоя свертки

#filters: количество фильтров (ядер)

#kernel\_size: размер ядра свертки

#padding: режим заполнения ('Same' - добавление нулевых значений по краям изображения)

#activation: функция активации ('relu' - выпрямительная функция)

#input\_shape: размерность входных данных (img\_rows, img\_cols, 3) - 3 канала (RGB)

deep\_RU\_model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = 'Same',

activation ='relu', input\_shape = (img\_rows,img\_cols,3)))

deep\_RU\_model.add(Conv2D(filters = 32, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

deep\_RU\_model.add(MaxPooling2D(pool\_size=(2,2)))

deep\_RU\_model.add(Dropout(0.25))

deep\_RU\_model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

deep\_RU\_model.add(Conv2D(filters = 64, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

deep\_RU\_model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2)))

deep\_RU\_model.add(Dropout(0.25))

#Выравнивание данных перед подачей на полносвязные слои

deep\_RU\_model.add(Flatten())

#Добавление первого полносвязного слоя

deep\_RU\_model.add(Dense(256, activation = "relu"))

#Добавление слоя Dropout

deep\_RU\_model.add(Dropout(0.5))

#Добавление последнего полносвязного слоя с функцией активации softmax для многоклассовой классификации

deep\_RU\_model.add(Dense(33, activation = "softmax"))

#Определение оптимизатора

optimizer = RMSprop(learning\_rate=0.001, rho=0.9, epsilon=1e-08)

#Создание оптимизатора RMSprop с параметрами:

#learning\_rate: скорость обучения (шаг градиентного спуска)

#rho: параметр альфа в формуле оптимизации

#epsilon: значение для стабильности численных вычислений

deep\_RU\_model.compile(loss="categorical\_crossentropy", optimizer=optimizer, metrics=["accuracy"])

# Компиляция модели с параметрами:

# loss: функция потерь ("categorical\_crossentropy" - категориальная перекрестная энтропия для многоклассовой классификации)

# optimizer: оптимизатор, ранее определенный (RMSprop)

# metrics: список метрик для оценки производительности модели (в данном случае указана только точность)

# Создание объекта learning\_rate\_reduction класса ReduceLROnPlateau

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

patience=3,

verbose=1,

factor=0.5,

min\_lr=0.00001)

# monitor: метрика для отслеживания (в данном случае 'val\_acc' - точность на валидационных данных)

# patience: количество эпох без улучшений, после которого скорость обучения будет уменьшена

# verbose: уровень подробности вывода информации (1 для вывода сообщений)

# factor: коэффициент, на который будет уменьшаться скорость обучения

# min\_lr: минимальное значение скорости обучения, после которого обучение прекращается

# Создание объекта es класса EarlyStopping

es = EarlyStopping(monitor='val\_accuracy', mode='max', verbose=1, patience=50)

# monitor: метрика для отслеживания (в данном случае 'val\_accuracy' - точность на валидационных данных)

# mode: режим ('max' - максимизация метрики)

# verbose: уровень подробности вывода информации (1 для вывода сообщений)

# patience: количество эпох без улучшений, после которого обучение будет остановлено

# Создание объекта mc класса ModelCheckpoint

mc = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', mode='max', verbose=1, save\_best\_only=True)

# 'best\_model.h5': имя файла, в котором будет сохранена модель с лучшей точностью на валидационных данных

# monitor: метрика для отслеживания (в данном случае 'val\_accuracy' - точность на валидационных данных)

# mode: режим ('max' - максимизация метрики)

# verbose: уровень подробности вывода информации (1 для вывода сообщений)

# save\_best\_only: флаг, указывающий, сохранять только модель с лучшей точностью

history = deep\_RU\_model.fit(datagen.flow(X\_train,y\_train, batch\_size=90), validation\_data = (X\_val, y\_val),

epochs=139, callbacks=[learning\_rate\_reduction, es, mc])

# datagen.flow(X\_train, y\_train, batch\_size=90): генератор данных для обучения модели, созданный с использованием данных X\_train и y\_train

# X\_train: входные данные для обучения

# y\_train: целевые данные для обучения

# batch\_size: размер пакета данных, на котором будут обновляться веса модели

# validation\_data=(X\_val, y\_val): данные для проверки производительности модели на каждой эпохе обучения

# X\_val: входные данные для проверки

# y\_val: целевые данные для проверки

# epochs: количество эпох обучения

# callbacks=[learning\_rate\_reduction, es, mc]: список колбэков, которые будут выполняться на различных этапах обучения

# learning\_rate\_reduction: уменьшение скорости обучения при отсутствии улучшений

# es: ранняя остановка обучения при отсутствии улучшений

# mc: сохранение модели с лучшей точностью на валидационных данных

saved\_model = load\_model('best\_model.h5')

#Оценка производительности модели

\_, train\_acc = saved\_model.evaluate(X\_train, y\_train, verbose=0)

\_, valid\_acc = saved\_model.evaluate(X\_val, y\_val, verbose=0)

print('Train: %.3f, Valid: %.3f' % (train\_acc, valid\_acc))

fig, ax = plt.subplots(2, 1)

ax[0].plot(history.history['loss'], color='b', label="Training loss") # График потерь на обучающем наборе данных

ax[0].plot(history.history['val\_loss'], color='r', label="Validation loss", axes=ax[0]) # График потерь на валидационном наборе данных

legend = ax[0].legend(loc='best', shadow=True) # Легенда для графика потерь

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy") # График точности на обучающем наборе данных

ax[1].plot(history.history['val\_accuracy'], color='r', label="Validation accuracy") # График точности на валидационном наборе данных

legend = ax[1].legend(loc='best', shadow=True) # Легенда для графика точности

# Создание графиков потерь и точности

# Параметры:

# history.history['loss']: значения функции потерь на обучающем наборе данных

# history.history['val\_loss']: значения функции потерь на валидационном наборе данных

# history.history['accuracy']: значения точности на обучающем наборе данных

# history.history['val\_accuracy']: значения точности на валидационном наборе данных

# Легенда и тени для каждого графика

# ax[0]: первый подграфик для графика потерь

# ax[1]: второй подграфик для графика точности

\_, test\_acc = saved\_model.evaluate(X\_test, y\_test, verbose=0)

print('Test: %.3f' % (test\_acc))

y\_pred = deep\_RU\_model.predict(X\_test)# Получение предсказаний модели для тестовых данных X\_test

y\_pred = np.argmax(y\_pred, axis=1)# Преобразование предсказаний в индексы с наибольшим значением (классы)

y\_test = np.argmax(y\_test, axis=1)# Преобразование истинных меток y\_test в индексы с наибольшим значением (классы)

confusion\_mtx = confusion\_matrix(y\_test, y\_pred)# Вычисление матрицы ошибок между истинными метками y\_test и предсказанными метками y\_pred

sns.heatmap(confusion\_mtx, annot=True, fmt='d')

# Параметры:

# confusion\_mtx: матрица ошибок

# annot=True: отображение значений в клетках тепловой карты

# fmt='d': формат чисел в клетках - десятичное целое число

# Преобразование меток из формата one-hot в исходный формат меток

Y\_true = y\_test

# Предсказание значений на тестовом наборе данных с помощью сохраненной модели

Y\_pred = saved\_model.predict(X\_test)

# Преобразование предсказанных значений из формата one-hot в исходный формат меток

Y\_pred\_classes = np.argmax(Y\_pred, axis=1)

# Вычисление ошибок путем сравнения предсказанных меток с истинными метками

errors = (Y\_pred\_classes - Y\_true != 0)

# Получение предсказанных меток для ошибочных предсказаний

Y\_pred\_classes\_errors = Y\_pred\_classes[errors]

# Получение предсказанных значений для ошибочных предсказаний

Y\_pred\_errors = Y\_pred[errors]

# Получение истинных меток для ошибочных предсказаний

Y\_true\_errors = Y\_true[errors]

# Получение изображений, соответствующих ошибочным предсказаниям

X\_val\_errors = X\_test[errors]

def display\_errors(errors\_index, img\_errors, pred\_errors, obs\_errors):

"""Функция отображает 9 изображений с предсказанными и истинными метками"""

n = 0

nrows = 3

ncols = 3

fig, ax = plt.subplots(nrows, ncols, sharex=True, sharey=True)

fig.suptitle("Отображение топ-9 ошибок", fontsize=16)

for row in range(nrows):

for col in range(ncols):

error = errors\_index[n]

ax[row, col].imshow((img\_errors[error]))

ax[row, col].set\_title("Предсказанная метка: {}\nИстинная метка: {}".format(pred\_errors[error], obs\_errors[error]))

n += 1

# Вероятности неправильно предсказанных символов

Y\_pred\_errors\_prob = np.max(Y\_pred\_errors, axis=1)

# Предсказанные вероятности истинных значений в ошибочном наборе

true\_prob\_errors = np.diagonal(np.take(Y\_pred\_errors, Y\_true\_errors, axis=1))

# Разница между вероятностью предсказанной метки и истинной меткой

delta\_pred\_true\_errors = Y\_pred\_errors\_prob - true\_prob\_errors

# Сортированный список ошибок по возрастанию разницы вероятностей

sorted\_delta\_errors = np.argsort(delta\_pred\_true\_errors)

# Топ-9 ошибок

most\_important\_errors = sorted\_delta\_errors[-9:]

# Отображение топ-9 ошибок

display\_errors(most\_important\_errors, X\_val\_errors, Y\_pred\_classes\_errors, Y\_true\_errors)