Документация по проекту “Handwritten Text Recognition (HTR)”

1. Описание задачи

Проект **Handwritten Text Recognition Model** направлен на обнаружение и распознавание рукописного текста на изображениях школьных тетрадей.

Основные задачи:

* **Сегментация текста** – выделение областей с текстом с помощью U-Net + ResNet50V2.
* **Распознавание текста (OCR)** – использование C-RNN + CTC для декодирования рукописного текста.

1. Структура проекта

* **Recognizer\_model.ipynb** - код для тестирования разных архитектур OCR модели на другом датасете.
* **Models.ipynb** - код с предобработкой данных и обучением моделей
* **General\_model.ipynb** - код с общей моделью HTR

1. Описание кода

**Recognizer\_model.ipynb**

| import tensorflow as tf import os import numpy as np from tqdm import tqdm import pandas as pd import matplotlib.pyplot as plt from sklearn.model\_selection import train\_test\_split from tensorflow.keras import backend as K from keras.models import Model from keras.layers import Input, Conv2D, MaxPool2D, Reshape, Bidirectional, LSTM, Dense, Lambda, Activation, BatchNormalization, Dropout, MaxPooling2D from keras.optimizers import Adam |
| --- |

Импорт необходимых библиотек, которые потребуются дальше.

| train\_df = pd.read\_csv("RU\_texts/train.tsv", sep="\t", header=None)  train\_df = train\_df.rename({0: "image\_name", 1: "label"}, axis=1)  train\_df = train\_df.dropna() |
| --- |

Загрузка csv-файла, переименование столбцов и удаление пустых строк.

| # Добавление контраста def adjust\_contrast(image):  return tf.image.adjust\_contrast(image, 2)  # Изменение размера, сохраняя пропорции (устанавливаем рамки) def distortion\_free\_resize(image, img\_size):  w, h = img\_size  image = tf.image.resize(image, size=(h, w), preserve\_aspect\_ratio=True) # Изменение размера с сохранением пропорций   # Получаем размеры рамок по ширине и длине  pad\_height = h - tf.shape(image)[0]  pad\_width = w - tf.shape(image)[1]   # Получаем размеры верхней и нижней, правой и левой рамок  if pad\_height % 2 != 0:  height = pad\_height // 2  pad\_height\_top = height + 1  pad\_height\_bottom = height  else:  pad\_height\_top = pad\_height\_bottom = pad\_height // 2    if pad\_width % 2 != 0:  width = pad\_width // 2  pad\_width\_left = width + 1  pad\_width\_right = width  else:  pad\_width\_left = pad\_width\_right = pad\_width // 2   # Добавляем рамки чёрного цвета  image = tf.pad(  image,  paddings=[  [pad\_height\_top, pad\_height\_bottom],  [pad\_width\_left, pad\_width\_right],  [0, 0]  ]  )  # Меняем размерность (h, w, channels) -> (w, h, channels)  image = tf.transpose(image, perm=[1, 0, 2])  # Разворот изображения по горизонтали  image = tf.image.flip\_left\_right(image)   return image # Предобработка изображения def preprocess\_image(image\_path, target\_size=(128, 32)):  image = tf.io.read\_file(image\_path)  image = tf.image.decode\_image(image, channels=1) # Декодирование изображения в ч/б канал  image = distortion\_free\_resize(image, target\_size)  image = adjust\_contrast(image)  image = tf.image.per Стандартизация значений  return image  # Загрузка изображений и меток def load\_images\_and\_labels(directory):  preprocessed\_images = [] # Преобразованные изображения  labels = [] # Метки   # Поиск всех файлов изображений в директории  for filename in tqdm(os.listdir(directory)):  file\_path = os.path.join(directory, filename)  if os.path.isfile(file\_path) and filename.lower().endswith(('.png', '.jpg', '.jpeg')):  try:  image = preprocess\_image(file\_path)  preprocessed\_images.append(image)  image\_name = filename.split("\_")[0] # Извлекаем имя из полного названия  label = train\_df[train\_df["image\_name"] == image\_name]["label"].iloc[0]  labels.append(label)  except Exception as e:  pass  return np.array(preprocessed\_images), np.array(labels) |
| --- |

Предобработка изображений (изменение размера, контраст, стандартизация)

| symbols = set(train\_df["label"].to\_string()) alphabets = set(" 0123456789:;?[]«»АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЭЮЯабвгдежзийклмнопрстуфхцчшщъыьэюяё") alphabets = "".join(list(alphabets | symbols)) |
| --- |

Формирование строки из всех символов, которые встречаются в метках.

| max\_str\_len = 40 # Максимальная длина метки num\_of\_characters = len(alphabets) + 1 Кол-во символов + 1 (для пустого значения) num\_of\_timestamps = 64 # Максимальная длина предсказанной метки  # Перевод строки в числовой формат def label\_to\_num(label):  label\_num = []  for ch in label:  label\_num.append(alphabets.find(ch))    return np.array(label\_num)  # Перевод числового формата в строку def num\_to\_label(num):  ret = ""  for ch in num:  if ch == -1: # CTC пустое значение  break  else:  ret+=alphabets[ch]  return ret |
| --- |

Вспомогательные функции и переменные для обработки меток.

| train\_size = len(x\_train) # Размер тренировочных данных train\_y = np.ones([train\_size, max\_str\_len]) \* len(alphabets) # Метки классов в формате OneHotEncoding train\_label\_len = np.zeros([train\_size, 1]) # Размер меток для тренировочных данных train\_input\_len = np.ones([train\_size, 1]) \* (num\_of\_timestamps-2) # Размер меток для подачи в модель train\_output = np.zeros([train\_size]) # Заглушка для модели (y - не нужен в архитектуре)  for i in range(train\_size):  train\_label\_len[i] = len(y\_train[i])  train\_y[i, 0:len(y\_train[i])]= label\_to\_num(y\_train[i]) |
| --- |

Предобработка меток для тренировочных данных. Для валидационных аналогично.

| input\_data = Input(shape=(128, 32, 1), name='input')  inner = Conv2D(16, (3, 3), padding='same', name='conv1', kernel\_initializer='he\_normal')(input\_data)  inner = BatchNormalization()(inner) inner = Activation('relu')(inner) inner = Conv2D(32, (3, 3), padding='same', name='conv2', kernel\_initializer='he\_normal')(input\_data)  inner = BatchNormalization()(inner) inner = Activation('relu')(inner) |
| --- |

| inner = MaxPooling2D(pool\_size=(2, 2), name='max1')(inner)  inner = Conv2D(64, (3, 3), padding='same', name='conv3', kernel\_initializer='he\_normal')(inner) inner = BatchNormalization()(inner) inner = Activation('relu')(inner) inner = MaxPooling2D(pool\_size=(2, 2), name='max2')(inner) inner = Dropout(0.5)(inner)  inner = Conv2D(128, (3, 3), padding='same', name='conv4', kernel\_initializer='he\_normal')(inner) inner = BatchNormalization()(inner) inner = Activation('relu')(inner) inner = MaxPooling2D(pool\_size=(1, 2), name='max3')(inner) inner = Dropout(0.5)(inner)  # CNN to RNN inner = Reshape(target\_shape=((64, 256)), name='reshape')(inner) inner = Dense(512, activation='relu', kernel\_initializer='he\_normal', name='dense1')(inner)  ## RNN inner = Bidirectional(LSTM(256, return\_sequences=True, dropout=0.3), name = 'lstm1')(inner) inner = Bidirectional(LSTM(256, return\_sequences=True, dropout=0.3), name = 'lstm2')(inner)  # Выходной слой inner = Dense(num\_of\_characters, kernel\_initializer='he\_normal', name='dense3')(inner) y\_pred = Activation('softmax', name='softmax')(inner)  # Модель model = Model(inputs=input\_data, outputs=y\_pred) model.summary() |
| --- |

Построение модели C-RNN через Keras. Определены слои и их параметры, добавлены Dropout для борьбы с переобучением. Общее кол-во параметров: **3,430,893.**

| **# CTC Loss def ctc\_lambda\_func(args):  y\_pred, labels, input\_length, label\_length = args  # Пропуск первых 2 элементов предсказания, так как RNN обычно в начале содержит мусор  y\_pred = y\_pred[:, 2:, :]  return K.ctc\_batch\_cost(labels, y\_pred, input\_length, label\_length)** |
| --- |

Функция для подсчёта CTC Loss.

| labels = Input(name='gtruth\_labels', shape=[max\_str\_len], dtype='float32') input\_length = Input(name='input\_length', shape=[1], dtype='int64') label\_length = Input(name='label\_length', shape=[1], dtype='int64')  ctc\_loss = Lambda(ctc\_lambda\_func, output\_shape=(1,), name='ctc')([y\_pred, labels, input\_length, label\_length]) model\_final = Model(inputs=[input\_data, labels, input\_length, label\_length], outputs=ctc\_loss) |
| --- |

Добавление входов модели, чтобы передавать все аргументы, которые требует CTC loss функция.

| callbacks = [  tf.keras.callbacks.ModelCheckpoint(  filepath='recognizer\_model.weights.h5',  save\_weights\_only=True),  tf.keras.callbacks.EarlyStopping(  monitor='val\_loss',  patience=5,  verbose=1,  restore\_best\_weights=True), ] |
| --- |

Добавление callbacks. ModelCheckpoint - сохранение весов модели, Early Stopping - остановка в случае остановки уменьшения val\_loss в течение 5 эпох.

| model\_final.compile(loss={'ctc': lambda y\_true, y\_pred: y\_pred}, optimizer=Adam(learning\_rate=0.001))  with tf.device("/GPU:0"):  history = model\_final.fit(x=[x\_train, train\_y, train\_input\_len, train\_label\_len], y=train\_output,  validation\_data=([x\_val, valid\_y, valid\_input\_len, valid\_label\_len], valid\_output),  epochs=60, batch\_size=128, verbose=1, callbacks=callbacks) |
| --- |

Обучение модели на 60 эпохах, batch\_size = 128 и с оптимизатором Adam.

| import matplotlib.pyplot as plt   plt.plot(history.history["loss"], label="Train") plt.plot(history.history["val\_loss"], label="Validation") plt.legend() plt.show() |
| --- |

Вывод результатов обучения на графике.

| preds = model.predict(x\_val) decoded = K.get\_value(K.ctc\_decode(preds, input\_length=np.ones(preds.shape[0])\*preds.shape[1],  greedy=True)[0][0])  prediction = [] for i in range(len(y\_val)):  prediction.append(num\_to\_label(decoded[i])) # Подсчёт метрик correct\_char = 0 total\_char = 0 correct = 0  for i in range(len(y\_val)):  pr = prediction[i]  tr = y\_val[i]  total\_char += len(tr)    for j in range(min(len(tr), len(pr))):  if tr[j] == pr[j]:  correct\_char += 1    if pr == tr :  correct += 1   print('Correct characters predicted : %.2f%%' %(correct\_char\*100/total\_char)) print('Correct words predicted : %.2f%%' %(correct\*100/len(x\_val))) |
| --- |

Предсказываем значения для тестовых данных. Правильно предсказанные символы: **79,74%.** Правильно предсказанные слова: **54,7%.**

**Models.ipynb**

| import tensorflow as tf import os import pandas as pd from tqdm import tqdm from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, Conv2DTranspose, Input, Concatenate, MaxPooling2D, Dropout, LSTM, \ Bidirectional, Reshape, Dense, Lambda from keras.applications.resnet\_v2 import ResNet50V2 from tensorflow.keras import backend as K from keras.models import Model from tensorflow.keras.optimizers import \* from keras.metrics import IoU import json import matplotlib.pyplot as plt import matplotlib.patches as patches import numpy as np from PIL import Image import cv2 from tensorflow.keras.utils import Sequence from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score from collections import Counter import seaborn as sns |
| --- |

Импорт необходимых библиотек.

| # Путь к JSON и изображениям json\_path = "annotations\_train.json" images\_folder = "images/"  # Загрузка аннотаций with open(json\_path, "r", encoding="utf-8") as f:  annotations\_data = json.load(f) # Парсинг данных images = {img["id"]: img for img in annotations\_data["images"]} categories = {cat["id"]: cat["name"] for cat in annotations\_data["categories"]} annotations = annotations\_data["annotations"]  words = []  def extract\_segments(image, annotations):  """  Вырезает сегменты из изображения на основе аннотаций.    Args:  image: PIL.Image.Image объект изображения.  annotations: Список аннотаций с полигонами.    Returns:  Список изображений-сегментов.  """  segments = []  image\_np = np.array(image) # Преобразование PIL.Image в NumPy array  for ann in annotations:  segmentation = np.array(ann['segmentation'][0], dtype=np.int32).reshape(-1, 2)  x, y, w, h = cv2.boundingRect(segmentation) # Ограничивающий прямоугольник  cropped = image\_np[y:y+h, x:x+w]    # Маска для вырезания конкретного сегмента  mask = np.zeros\_like(image\_np[:, :, 0]) # Черно-белая маска  cv2.fillPoly(mask, [segmentation], 255)  masked\_image = cv2.bitwise\_and(cropped, cropped, mask=mask[y:y+h, x:x+w])    segments.append(masked\_image)  return segments   # Визуализация аннотаций для одного изображения def visualize\_annotations(image\_id):  image\_info = images[image\_id]  image\_path = images\_folder + image\_info["file\_name"]    # Загружаем изображение  image = Image.open(image\_path)    # Создаем фигуру  fig, ax = plt.subplots(figsize=(12, 12))  ax.imshow(image)    # Получаем аннотации для изображения  image\_annotations = [ann for ann in annotations if ann["image\_id"] == image\_id]    # Визуализация многоугольников  for ann in image\_annotations:  if ann["category\_id"] == 0:  segmentation = ann["segmentation"][0] # Координаты сегментации  polygon = np.array(segmentation).reshape(-1, 2)  category\_name = categories[ann["category\_id"]]    # Отображаем многоугольник  patch = patches.Polygon(  polygon, closed=True, edgecolor="red", linewidth=1, facecolor="none"  )  ax.add\_patch(patch)    words.append(ann["attributes"]["translation"])      plt.axis("off")  plt.show()  # Пример: визуализация для изображения с ID = 1 visualize\_annotations(image\_id=1) |
| --- |

Функции для извлечения и визуализации сегментов изображения на основе аннотаций.

| def create\_mask(image\_shape, annotations):  """  Создает бинарную маску из аннотаций.   Args:  image\_shape: Размер исходного изображения (высота, ширина).  annotations: Аннотации COCO формата (список словарей).    Returns:  Маска для данного изображения.  """  mask = np.zeros(image\_shape[:2], dtype=np.uint8) # Создаем пустую черно-белую маску  for ann in annotations:  segmentation = np.array(ann['segmentation'][0], dtype=np.int32).reshape(-1, 2)  cv2.fillPoly(mask, [segmentation], 1) # Заполняем область внутри полигона  return mask def resize\_image\_and\_mask(image, mask, target\_size=(3000, 4000)):  """  Изменяет размер изображения и маски до target\_size.   Args:  image: Исходное изображение (NumPy массив).  mask: Бинарная маска.  target\_size: Кортеж (высота, ширина).    Returns:  Измененные изображение и маска.  """  resized\_image = cv2.resize(image, target\_size, interpolation=cv2.INTER\_LINEAR)  resized\_mask = cv2.resize(mask, target\_size, interpolation=cv2.INTER\_NEAREST)  return resized\_image, resized\_mask |
| --- |

Функция для создания чёрно-белых масок на основе аннотаций. Эти маски в последующем будут переданы модели для обучения.

| class ImageMask(Sequence):  def \_\_init\_\_(self, images, annotations, images\_folder, batch\_size=8, target\_size=(256, 256)):  """  Инициализация генератора данных.  :param images: список информации об изображениях (из annotation["images"]).  :param annotations: список аннотаций (из annotation["annotations"]).  :param images\_folder: путь к папке с изображениями.  :param batch\_size: размер батча.  :param target\_size: желаемый размер изображений и масок (H, W).  """  self.images = images  self.annotations = annotations  self.images\_folder = images\_folder  self.batch\_size = batch\_size  self.target\_size = target\_size   def \_\_len\_\_(self):  return int(np.ceil(len(self.images) / self.batch\_size))   def \_\_getitem\_\_(self, idx):  batch\_images = self.images[idx \* self.batch\_size:(idx + 1) \* self.batch\_size]   images = []  masks = []    for image\_info in batch\_images:  # Путь к изображению  image\_path = self.images\_folder + image\_info["file\_name"]  image\_shape = (image\_info["height"], image\_info["width"])    # Загружаем изображение  image = Image.open(image\_path).convert("RGB")  image = np.array(image) / 255.0 # Нормализация пикселей    # Загружаем аннотации для данного изображения  image\_annotations = [  ann for ann in self.annotations  if ann["image\_id"] == image\_info["id"] and ann["category\_id"] == 0  ]    # Создаём маску  mask = create\_mask(image\_shape, image\_annotations)   image, mask = resize\_image\_and\_mask(image, mask, self.target\_size)    images.append(image)  masks.append(mask)    return np.array(images), np.array(masks) |
| --- |

Создание класса-генератора для изображения, который возвращает на каждой итерации batch, состоящий из данных в формате: (нормализованное изображение np.array, чёрно-белая маска np.array).

| def conv\_block(inputs, num\_filters):  x = Conv2D(num\_filters, 3, padding="same")(inputs)  x = BatchNormalization()(x)  x = Activation("relu")(x)   x = Conv2D(num\_filters, 3, padding="same")(x)  x = BatchNormalization()(x)  x = Activation("relu")(x)   return x   def decoder\_block(inputs, skip\_features, num\_filters):  x = Conv2DTranspose(num\_filters, (2, 2), strides=2, padding="same")(inputs)  x = Concatenate()([x, skip\_features])  x = conv\_block(x, num\_filters)   return x   def build\_unet\_resnet\_model(input\_shape):  '''Input layer'''  inputs = Input(input\_shape)   '''Pre-trained ResNet50V2'''  resnet50 = ResNet50V2(include\_top=False, weights="imagenet", input\_tensor=inputs)   for layer in resnet50.layers:  layer.trainable = False   '''Skip connections'''  s1 = inputs # 1024  s2 = resnet50.get\_layer("conv1\_conv").output # 512  s3 = resnet50.get\_layer("conv2\_block3\_1\_relu").output # 256  s4 = resnet50.get\_layer("conv3\_block4\_1\_relu").output # 128   '''Bridge'''  b1 = resnet50.get\_layer("conv4\_block6\_1\_relu").output # 64   '''Decoder'''  d1 = decoder\_block(b1, s4, 1024) # 128  d2 = decoder\_block(d1, s3, 512) # 256  d3 = decoder\_block(d2, s2, 256) # 512  d4 = decoder\_block(d3, s1, 128) # 1024   '''Output Layer'''  outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4) # Только 1 класс - текст   model = Model(inputs, outputs)   return model |
| --- |

Построение модели сегментации на основе архитектуры U-Net с применением слоёв модели ResNet50V2 в качестве энкодера. Общее кол-во параметров: **38,234,753.**

| model.compile(optimizer=Adam(learning\_rate=1e-4), loss='binary\_crossentropy', metrics =["accuracy", IoU(num\_classes=2, target\_class\_ids=[0,1])]) |
| --- |

Сбор модели с добавлением метрик accuracy и IOU (Intersection-over-Union). В качестве оптимизатора используется Adam.

| callbacks = [  tf.keras.callbacks.ModelCheckpoint(  filepath='segmentation\_model.weights.h5',  save\_weights\_only=True),  tf.keras.callbacks.EarlyStopping(  monitor='val\_loss',  patience=3,  verbose=1,  restore\_best\_weights=True), ] |
| --- |

Добавление callbacks. ModelCheckpoint - сохранение весов модели, Early Stopping - остановка в случае остановки уменьшения val\_loss в течение 3 эпох.

| with tf.device("CPU:0/"):  history = model.fit(image\_mask, steps\_per\_epoch=len(image\_mask), epochs=50, verbose=1, callbacks=callbacks,  validation\_data=val\_image\_mask, validation\_steps=len(val\_image\_mask)) |
| --- |

Обучение модели с использованием CPU на 50 эпохах.

| json\_path = "annotations\_test.json"  with open(json\_path, "r", encoding="utf-8") as f:  test\_ann\_data = json.load(f)  test\_image\_mask = ImageMask(  images=test\_ann\_data["images"],  annotations=test\_ann\_data["annotations"],  images\_folder="images/",  batch\_size=5,  target\_size=(1024, 1024) ) x\_test, y\_test = test\_image\_mask[0] x\_pred = second\_model.predict(x\_test) threshold = 0.5 binary\_prediction = (x\_pred >= threshold).astype(np.uint8).squeeze() y\_true = y\_test.flatten() y\_pred = binary\_prediction.flatten()  accuracy = accuracy\_score(y\_true, y\_pred) precision = precision\_score(y\_true, y\_pred) recall = recall\_score(y\_true, y\_pred) f1 = f1\_score(y\_true, y\_pred)  print(f"Точность: {accuracy:.2f}") print(f"Полнота: {recall:.2f}") print(f"Точность: {precision:.2f}") print(f"F1-мера: {f1:.2f}") |
| --- |

Тестирование модели на тестовых данных. Точность: **0.95**, Полнота: **0.97**, Precision: **0.86**, F1-score: **0.91**.

| def extract\_text\_regions\_for\_all\_images(coco\_annotations, images\_folder, output\_dir):  """  Вырезает текстовые регионы из всех изображений в COCO-аннотациях.   Args:  coco\_annotations: JSON-данные COCO (словарь с ключами "images" и "annotations").  images\_folder: Папка с исходными изображениями.  output\_dir: Директория для сохранения вырезанных областей.    Returns:  DataFrame с информацией о вырезанных областях.  """  os.makedirs(output\_dir, exist\_ok=True)   # Загружаем список изображений и аннотаций  images = {img["id"]: img["file\_name"] for img in coco\_annotations["images"]}  annotations = coco\_annotations["annotations"]   dataset = []   # Проходим по всем изображениям  for image\_info in tqdm(coco\_annotations["images"]):  try:  image\_id = image\_info["id"]  image\_annotations = [  ann for ann in annotations  if ann["image\_id"] == image\_id and ann["category\_id"] == 0 # Фильтрация по категории (если нужно)  ]   if not image\_annotations:  continue # Если нет аннотаций для изображения, пропускаем   image\_path = os.path.join(images\_folder, image\_info["file\_name"])  if not os.path.exists(image\_path):  print(f"⚠ Пропущено: {image\_path} не найдено")  continue    image = cv2.imread(image\_path)  if image is None:  print(f"⚠ Ошибка загрузки: {image\_path}")  continue   # Проходим по всем аннотациям для текущего изображения  for index, ann in enumerate(image\_annotations):  # Получаем координаты региона  segmentation = np.array(ann['segmentation'][0], dtype=np.int32).reshape(-1, 2)    # Вычисляем ограничивающий прямоугольник  x, y, w, h = cv2.boundingRect(segmentation)  cropped = image[y:y+h, x:x+w] # Вырезаем область   # Делаем черно-белым  cropped\_gray = cv2.cvtColor(cropped, cv2.COLOR\_BGR2GRAY)   # Сохраняем  file\_name = f"{image\_id}\_region\_{index}.jpg"  file\_path = os.path.join(output\_dir, file\_name)  Image.fromarray(cropped\_gray).save(file\_path)   # Добавляем в датасет  dataset.append({"file\_name": file\_name, "translation": ann.get("attributes", {}).get("translation", "")})    except:  continue   return pd.DataFrame(dataset) |
| --- |

Функция для выделения отдельных слов на основе аннотаций для формирования датасета, предназначенного для обучения OCR модели.

| coco\_annotations\_path = "annotations\_train.json" # Ваш COCO JSON images\_folder = "images/" # Папка с изображениями output\_dir = "dataset/" # Куда сохранять  # Загружаем аннотации with open(coco\_annotations\_path, "r", encoding="utf-8") as f:  coco\_data = json.load(f)  # Генерируем датасет df = extract\_text\_regions\_for\_all\_images(coco\_data, images\_folder, output\_dir)  # Сохраняем метки в CSV df.to\_csv(os.path.join(output\_dir, "labels.csv"), index=False)  print("Датасет успешно создан!") |
| --- |

Формирование датасета.

| # Разбиваем слова на буквы и считаем частоту letter\_counts = Counter("".join(df["translation"].astype(str)))  # Создаём DataFrame из частот letters\_df = pd.DataFrame(letter\_counts.items(), columns=["Буква", "Частота"]).sort\_values(by="Частота", ascending=False)[:100]  # Визуализация plt.figure(figsize=(14, 5)) sns.barplot(x="Буква", y="Частота", data=letters\_df, palette="viridis") plt.title("Распределение букв в датасете") plt.xlabel("Буквы") plt.ylabel("Частота") plt.show() |
| --- |

Построение графика распределения букв в сформированном датасете.

*\*Далее код для OCR модели, который совпадает с кодом определённым в файле* **Recognizer\_model.ipynb.** Правильно предсказанные символы: **82,82%.** Правильно предсказанные слова: **62,14%**

**General\_model.ipynb**

| **import cv2 import tensorflow as tf import numpy as np from PIL import Image from keras.applications.resnet\_v2 import ResNet50V2 from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, Conv2DTranspose, Input, Concatenate, MaxPooling2D, Dropout, LSTM, \ Bidirectional, Reshape, Dense from tensorflow.keras import backend as K from keras.models import Model import matplotlib.pyplot as plt import pandas as pd import os** |
| --- |

Импорт необходимых библиотек.

| class HTRModel:  def \_\_init\_\_(self, alphabets, max\_str\_len, num\_of\_characters, num\_of\_timestamps, htr\_weights\_path, target\_size=(128, 32)):  self.alphabets = alphabets  self.max\_str\_len = max\_str\_len  self.num\_of\_characters = num\_of\_characters  self.num\_of\_timestamps = num\_of\_timestamps  self.htr\_weights\_path = htr\_weights\_path  self.target\_size = target\_size    def \_\_num\_to\_label(self, num):  ret = ""  for ch in num:  if ch == -1: # CTC Blank  break  else:  ret+=self.alphabets[ch]  return ret   def \_\_get\_model(self):  input\_data = Input(shape=(self.target\_size[0], self.target\_size[1], 1), name='input')   inner = Conv2D(16, (3, 3), padding='same', name='conv1', kernel\_initializer='he\_normal')(input\_data)   inner = BatchNormalization()(inner)  inner = Activation('relu')(inner)  inner = Conv2D(32, (3, 3), padding='same', name='conv2', kernel\_initializer='he\_normal')(input\_data)   inner = BatchNormalization()(inner)  inner = Activation('relu')(inner)  inner = MaxPooling2D(pool\_size=(2, 2), name='max1')(inner)   inner = Conv2D(64, (3, 3), padding='same', name='conv3', kernel\_initializer='he\_normal')(inner)  inner = BatchNormalization()(inner)  inner = Activation('relu')(inner)  inner = MaxPooling2D(pool\_size=(2, 2), name='max2')(inner)  inner = Dropout(0.5)(inner)   inner = Conv2D(128, (3, 3), padding='same', name='conv4', kernel\_initializer='he\_normal')(inner)  inner = BatchNormalization()(inner)  inner = Activation('relu')(inner)  inner = MaxPooling2D(pool\_size=(1, 2), name='max3')(inner)  inner = Dropout(0.5)(inner)   # b to c  inner = Reshape(target\_shape=((64, 256)), name='reshape')(inner)  inner = Dense(512, activation='relu', kernel\_initializer='he\_normal', name='dense1')(inner)   ## RNN  inner = Bidirectional(LSTM(256, return\_sequences=True, dropout=0.3), name = 'lstm1')(inner)  inner = Bidirectional(LSTM(256, return\_sequences=True, dropout=0.3), name = 'lstm2')(inner)   # Выходной слой  inner = Dense(self.num\_of\_characters, kernel\_initializer='he\_normal', name='dense3')(inner)  y\_pred = Activation('softmax', name='softmax')(inner)   # Модель  model = Model(inputs=input\_data, outputs=y\_pred)   return model     def predict\_words(self, segments):  model = self.\_\_get\_model()  model.load\_weights(self.htr\_weights\_path)   preds = model.predict(segments)  decoded = K.get\_value(K.ctc\_decode(preds, input\_length=np.ones(preds.shape[0])\*preds.shape[1],  greedy=True)[0][0])   prediction = []  for i in range(len(decoded)):  prediction.append(self.\_\_num\_to\_label(decoded[i]))   return prediction |
| --- |

Класс HTRModel содержит в себе функции num\_to\_label (работает аналогично одноименной из **Models.ipynb**), get\_model (создаёт модель Keras) и predict\_words (загружает веса в модель, передаёт изображения для предсказания и возвращает предсказания модели).

| class TextRecognition:  def \_\_init\_\_(self, images\_paths, target\_size, seg\_weights\_path, htr\_model, segment\_threshold=0.5):  self.images\_paths = images\_paths  self.target\_size = target\_size  self.seg\_weights\_path = seg\_weights\_path  self.segment\_threshold = segment\_threshold  self.htr\_model = htr\_model    def \_\_prepare\_image(self, image\_path):  image = Image.open(image\_path).convert("RGB")  image = np.array(image) / 255.0 # Нормализация пикселей  resized\_image = cv2.resize(image, self.target\_size, interpolation=cv2.INTER\_CUBIC)  return resized\_image    def \_\_conv\_block(self, inputs, num\_filters):  x = Conv2D(num\_filters, 3, padding="same")(inputs)  x = BatchNormalization()(x)  x = Activation("relu")(x)   x = Conv2D(num\_filters, 3, padding="same")(x)  x = BatchNormalization()(x)  x = Activation("relu")(x)   return x   def \_\_decoder\_block(self, inputs, skip\_features, num\_filters):  x = Conv2DTranspose(num\_filters, (2, 2), strides=2, padding="same")(inputs)  x = Concatenate()([x, skip\_features])  x = self.\_\_conv\_block(x, num\_filters)   return x   def \_\_build\_unet\_resnet\_model(self, channels=3):  '''Input layer'''  inputs = Input((self.target\_size[0], self.target\_size[1], channels))    '''Pre-trained ResNet50V2'''  resnet50 = ResNet50V2(include\_top=False, weights="imagenet", input\_tensor=inputs)   for layer in resnet50.layers:  layer.trainable = False    '''Skip connections'''  s1 = inputs # 1024  s2 = resnet50.get\_layer("conv1\_conv").output # 512  s3 = resnet50.get\_layer("conv2\_block3\_1\_relu").output # 256  s4 = resnet50.get\_layer("conv3\_block4\_1\_relu").output # 128   '''Bridge'''  b1 = resnet50.get\_layer("conv4\_block6\_1\_relu").output # 64   '''Decoder'''  d1 = self.\_\_decoder\_block(b1, s4, 256) # 128  d2 = self.\_\_decoder\_block(d1, s3, 128) # 256  d3 = self.\_\_decoder\_block(d2, s2, 64) # 512  d4 = self.\_\_decoder\_block(d3, s1, 32) # 1024   '''Output Layer'''  outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)    model = Model(inputs, outputs)  return model   def \_\_get\_segmentation(self, images):  model = self.\_\_build\_unet\_resnet\_model()  model.load\_weights(self.seg\_weights\_path)  pred\_mask = model.predict(images)  binary\_prediction = (pred\_mask >= self.segment\_threshold).astype(np.uint8)  return binary\_prediction   def \_\_adjust\_contrast(self, image):  return tf.image.adjust\_contrast(image, 2)   def \_\_distortion\_free\_resize(self, image, img\_size):  w, h = img\_size  image = tf.image.resize(image, size=(h, w), preserve\_aspect\_ratio=True)   # Определяем отступы  pad\_height = h - tf.shape(image)[0]  pad\_width = w - tf.shape(image)[1]   pad\_height\_top = pad\_height // 2  pad\_height\_bottom = pad\_height - pad\_height\_top  pad\_width\_left = pad\_width // 2  pad\_width\_right = pad\_width - pad\_width\_left   image = tf.pad(  image,  paddings=[  [pad\_height\_top, pad\_height\_bottom],  [pad\_width\_left, pad\_width\_right],  [0, 0]  ]  )   image = tf.transpose(image, perm=[1, 0, 2])  image = tf.image.flip\_left\_right(image)   return image   # ✅ Исправленный метод для обработки изображений в виде list  def \_\_preprocess\_image(self, image\_list, target\_size=(128, 32)):  # Преобразуем list в numpy-массив, затем в tf.Tensor  image = tf.convert\_to\_tensor(np.array(image\_list), dtype=tf.float32)   # Если изображение 2D, добавляем канал (ч/б изображение)  if len(tf.shape(image)) == 2:  image = tf.expand\_dims(image, axis=-1)   if tf.shape(image)[-1] == 3:  image = tf.image.rgb\_to\_grayscale(image)   image = self.\_\_distortion\_free\_resize(image, target\_size)  image = self.\_\_adjust\_contrast(image)  image = tf.image.per\_image\_standardization(image)   return image    def \_\_extract\_word\_segments(self, image\_np, mask):  contours, \_ = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  word\_segments = []   for contour in contours:  x, y, w, h = cv2.boundingRect(contour)  cropped\_word = image\_np[y:y + h, x:x + w].copy()   word\_segments.append(cropped\_word)    return word\_segments    def extract\_words(self):  images = [self.\_\_prepare\_image(image\_path) for image\_path in self.images\_paths]  images = np.array(images)  pred\_segments = self.\_\_get\_segmentation(images)   all\_words\_segments = []    for image\_index, segmented\_mask in enumerate(pred\_segments):  image\_np = (images[image\_index] \* 255).astype(np.uint8)  words\_segments = self.\_\_extract\_word\_segments(image\_np, segmented\_mask)  all\_words\_segments.extend(words\_segments)   preds\_segments = [self.\_\_preprocess\_image(segment) for segment in all\_words\_segments]   preds\_words = self.htr\_model.predict\_words(tf.stack(preds\_segments))   return preds\_segments, preds\_words |
| --- |

Класс TextRecognizer содержит в себе функции prepare\_image (чтение изображения, нормализация, изменения размера); encoder\_block, decoder\_block и build\_unet\_resnet\_model (построение модели U-Net с применением слоёв ResNet50V2); get\_segmentation (загрузка весов в модель, возвращение результата предсказания модели); adjust\_contrast, distortion\_free\_resize, preprocess\_image (преобразование изображений для подачи в модель OCR); extract\_word\_segments (извлечение слов из предсказанных областей); extract\_words (предсказание моделью OCR/HTR).