МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(МОСКОВСКИЙ ПОЛИТЕХ)**

Транспортный факультет

Направление подготовки: 01.03.02 «Прикладная математика и информатика»

Отчет по Лабораторной работе №5

“Поиск текста и лиц на изображении”

Выполнил:

Студент:

Группа: 211-172

Тимошенко Ф.Р.

Проверил:

Кожухов Д.А.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2024

## 1. Введение

Цель данного проекта заключается в разработке программного средства для обработки изображений и видео с использованием графического интерфейса. Программа должна предоставлять пользователю возможность загрузки изображений и видео, а также выполнять различные операции обработки, такие как выделение текстовых областей, распознавание текста, обнаружение лиц и наложение масок на лица в реальном времени.

## 1. Введение

Программа разработана для обработки изображений и видео с использованием библиотек PyQt6 и OpenCV. Она предоставляет пользователю возможность загружать изображения и видео файлы, а затем выполнять различные операции обработки, такие как выделение текста на изображении, поиск лиц на видео и применение масок к лицам.

## 2. Реализация

Программа состоит из двух основных файлов: "window.py" и "main.py".

2.1 Файл "window.py"

В этом файле определен класс ImageWindow, который наследуется от QMainWindow. Этот класс представляет собой графическое окно приложения с различными элементами управления, такими как метки для изображений, кнопки и выпадающие списки. Он содержит следующие атрибуты и методы:

**Атрибуты:**

* initial\_path: строка, путь к изображению или видео.
* img\_or\_video: целое число, индикатор выбора изображения или видео (0 - изображение, 1 - видео).
* cap: объект захвата видео из OpenCV.
* textCount: целое число, количество выделенных текстовых областей на изображении.
* selected\_text\_index: целое число, индекс выбранной текстовой области.
* data: словарь, данные о текстовых областях на изображении.
* face\_cascade: объект классификатора каскадов Хаара для обнаружения лиц.
* mask\_image: изображение маски для применения к лицам на видео.

**Методы:**

* \_\_init\_\_(): конструктор класса.
* create\_video\_buttons(): создает кнопки управления обработкой видео.
* show\_video\_buttons(show): отображает или скрывает кнопки управления обработкой видео в зависимости от аргумента show.
* on\_button\_text\_clicked(): обработчик события нажатия кнопки "Текст".
* on\_button\_face\_clicked(): обработчик события нажатия кнопки "Лицо".
* on\_button\_mask\_clicked(): обработчик события нажатия кнопки "Маска".
* on\_button\_stop\_clicked(): обработчик события нажатия кнопки "Стоп".
* on\_combo\_box\_changed(value): обработчик события изменения выбора режима работы (изображение или видео).
* on\_button\_vtext\_clicked(): обработчик события нажатия кнопки "Выделить текст".
* on\_combo\_box\_selectText\_changed(value): обработчик события изменения выбора текстовой области.
* create\_combo\_box\_selectText(numItems): создает выпадающий список для выбора текстовой области.
* save\_text\_image(): сохраняет выбранную текстовую область как изображение.
* extract\_text(): извлекает текст из выбранной текстовой области.
* download\_img(i): загружает изображение.
* download\_video(i): загружает видео.
* video\_process1(): процесс обработки видео для первого режима.
* video\_process2(oper): процесс обработки видео для второго режима.
* detect\_text\_area(): обнаруживает текстовые области на изображении.
* video\_text(): выполняет поиск текста на видео.
* detect\_faces\_in\_video(): обнаруживает лица на видео.
* apply\_masks\_in\_video(): применяет маски к лицам на видео.

## 2.2 Файл "main.py"

Этот файл содержит класс MyWindow, который наследует функционал класса ImageWindow из файла "window.py". Здесь также определены атрибуты и методы для обработки изображений и видео.

Атрибуты:

* initial\_path: строка, путь к изображению или видео.
* img\_or\_video: целое число, индикатор выбора изображения или видео (0 - изображение, 1 - видео).
* cap: объект захвата видео из OpenCV.
* textCount: целое число, количество выделенных текстовых областей на изображении.
* selected\_text\_index: целое число, индекс выбранной текстовой области.
* data: словарь, данные о текстовых областях на изображении.
* face\_cascade: объект классификатора каскадов Хаара для обнаружения лиц.
* mask\_image: изображение маски для применения к лицам на видео.

Методы:

* \_\_init\_\_(): конструктор класса.
* create\_video\_buttons(): создает кнопки управления обработкой видео.
* show\_video\_buttons(show): отображает или скрывает кнопки управления обработкой видео в зависимости от аргумента show.
* on\_button\_text\_clicked(): обработчик события нажатия кнопки "Текст".
* on\_button\_face\_clicked(): обработчик события нажатия кнопки "Лицо".
* on\_button\_mask\_clicked(): обработчик события нажатия кнопки "Маска".
* on\_button\_stop\_clicked(): обработчик события нажатия кнопки "Стоп".
* on\_combo\_box\_changed(value): обработчик события изменения выбора режима работы (изображение или видео).
* on\_button\_vtext\_clicked(): обработчик события нажатия кнопки "Выделить текст".
* on\_combo\_box\_selectText\_changed(value): обработчик события изменения выбора текстовой области.
* create\_combo\_box\_selectText(numItems): создает выпадающий список для выбора текстовой области.
* save\_text\_image(): сохраняет выбранную текстовую область как изображение.
* extract\_text(): извлекает текст из выбранной текстовой области.
* download\_img(i): загружает изображение.
* download\_video(i): загружает видео.
* video\_process1(): процесс обработки видео для первого режима.
* video\_process2(oper): процесс обработки видео для второго режима.
* detect\_text\_area(): обнаруживает текстовые области на изображении.
* video\_text(): выполняет поиск текста на видео.
* detect\_faces\_in\_video(): обнаруживает лица на видео.
* apply\_masks\_in\_video(): применяет маски к лицам на видео.

## 3. Описание работы программы

1. Пользователь запускает программу и видит главное окно приложения.
2. Пользователь выбирает режим работы (обработка изображения или видео) с помощью выпадающего списка.
3. В зависимости от выбора пользователя, программа предоставляет возможность загрузить соответствующий файл (изображение или видео) с помощью соответствующей кнопки.
4. После загрузки файла пользователь может выполнить различные операции обработки, такие как выделение текста на изображении или поиск лиц на видео, с помощью доступных кнопок.
5. Результат обработки отображается в графическом интерфейсе программы.

## 4. Листинг кода

“window.py”

|  |
| --- |
| from PyQt6.QtWidgets import QComboBox, QSlider, QApplication, QMainWindow, QLabel, QVBoxLayout, QWidget, QPushButton, QHBoxLayout  from PyQt6.QtGui import QPixmap  from PyQt6.QtCore import Qt  class ImageWindow(QMainWindow):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.setWindowTitle("Изображения")          self.setGeometry(50, 50, 400, 400)          self.central\_widget = QWidget()          self.setCentralWidget(self.central\_widget)          self.layout = QHBoxLayout()          self.central\_widget.setLayout(self.layout)          self.image\_layout = QVBoxLayout()          self.layout.addLayout(self.image\_layout)          self.image\_label1 = QLabel()          self.label1\_title = QLabel('До обработки')          self.image\_label2 = QLabel()          self.label2\_title = QLabel('После обработки')          #self.update\_images1("stuff/images/white.jpg")          #self.update\_images2("stuff/images/white.jpg")          self.image\_layout.addWidget(self.label1\_title, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.image\_label1, alignment=Qt.AlignmentFlag.AlignCenter)          self.label1\_title.hide()          self.image\_layout.addWidget(self.label2\_title, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.image\_label2, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_label3 = QLabel()          self.label3\_title = QLabel('Участок с текстом')          self.image\_layout.addWidget(self.label3\_title, alignment=Qt.AlignmentFlag.AlignCenter)          self.image\_layout.addWidget(self.image\_label3, alignment=Qt.AlignmentFlag.AlignCenter)          self.button\_layout = QVBoxLayout()          self.layout.addLayout(self.button\_layout)          self.update\_button()          self.img\_hide()          self.show()      def update\_images1(self, image\_path1):          self.label1\_title.show()          pixmap1 = QPixmap(image\_path1)          scaled\_pixmap1 = pixmap1.scaled(400, 400, Qt.AspectRatioMode.KeepAspectRatio)          self.image\_label1.setPixmap(scaled\_pixmap1)          self.image\_label1.show()          self.update()      def img\_hide(self):          self.label1\_title.hide()          self.image\_label1.hide()          self.label2\_title.hide()          self.image\_label2.hide()          self.label3\_title.hide()          self.image\_label3.hide()      def update\_images2(self, image\_path2):          self.label2\_title.show()          pixmap2 = QPixmap(image\_path2)          scaled\_pixmap2 = pixmap2.scaled(400, 400, Qt.AspectRatioMode.KeepAspectRatio)          self.image\_label2.setPixmap(scaled\_pixmap2)          self.image\_label2.show()          self.update()      def update\_image3(self, image\_patch2):          self.label3\_title.show()          pixmap3 = QPixmap(image\_patch2)          scaled\_pixmap3 = pixmap3.scaled(400, 400, Qt.AspectRatioMode.KeepAspectRatio)          self.image\_label3.setPixmap(scaled\_pixmap3)          self.image\_label3.show()      def update\_button(self):          self.combo\_box = QComboBox()          self.combo\_box.addItem("Выбрать видео")          self.combo\_box.addItem("Выбрать изображение")          self.combo\_box.currentIndexChanged.connect(self.on\_combo\_box\_changed)          self.button\_layout.addWidget(self.combo\_box)          self.button\_vtext = QPushButton("Выделить текст")          self.button\_vtext.clicked.connect(self.on\_button\_vtext\_clicked)          self.button\_layout.addWidget(self.button\_vtext)          self.button\_vtext.hide()          self.combo\_box\_selectText = QComboBox()          self.combo\_box\_selectText.currentIndexChanged.connect(self.on\_combo\_box\_selectText\_changed)          self.create\_combo\_box\_selectText(10)          self.button\_layout.addWidget(self.combo\_box\_selectText)          self.combo\_box\_selectText.hide()      def show\_vtext(self, i):          if i:              self.button\_vtext.show()          else:              self.button\_vtext.hide()      def show\_uchastok(self, i):          if i:              self.combo\_box\_selectText.show()          else:              self.combo\_box\_selectText.hide()      def img\_selectedshow(self):          self.show\_vtext(1)          self.show\_uchastok(0)      def select\_text\_show(self):          self.show\_vtext(0)          self.show\_uchastok(1)      def select\_text\_rectangle(self):          self.show\_vtext(0)          self.show\_uchastok(1) |

“main.py”

|  |
| --- |
| import cv2  import sys  from window import ImageWindow  import numpy as np  from PyQt6.QtWidgets import QApplication, QFileDialog, QMessageBox, QPushButton  import pytesseract  pytesseract.pytesseract.tesseract\_cmd = "C:\\Program Files\\Tesseract-OCR\\tesseract.exe"  save\_process\_path = 'stuff/saved/save\_proc.jpg'  class Mywindow(ImageWindow):      def \_\_init\_\_(self):          super().\_\_init\_\_()          self.initial\_path = ''          self.img\_or\_video = 0          self.cap = None          self.textCount = None          self.selected\_text\_index = None          self.data = None          self.create\_video\_buttons()          self.face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')          self.mask\_image = cv2.imread('C:/opencv/lab5/stuff/images/mask.png', cv2.IMREAD\_UNCHANGED)  # Ensure you have the mask image with an alpha channel        def create\_video\_buttons(self):          self.button\_text = QPushButton("Текст")          self.button\_face = QPushButton("Лицо")          self.button\_mask = QPushButton("Маска")          self.button\_stop = QPushButton("Стоп")            self.button\_text.clicked.connect(self.on\_button\_text\_clicked)          self.button\_face.clicked.connect(self.on\_button\_face\_clicked)          self.button\_mask.clicked.connect(self.on\_button\_mask\_clicked)          self.button\_stop.clicked.connect(self.on\_button\_stop\_clicked)            self.button\_layout.addWidget(self.button\_text)          self.button\_layout.addWidget(self.button\_face)          self.button\_layout.addWidget(self.button\_mask)          self.button\_layout.addWidget(self.button\_stop)            self.show\_video\_buttons(False)      def show\_video\_buttons(self, show):          self.button\_text.setVisible(show)          self.button\_face.setVisible(show)          self.button\_mask.setVisible(show)          self.button\_stop.setVisible(show)      def on\_button\_text\_clicked(self):          self.stop\_video\_processing = False          self.video\_text()      def on\_button\_face\_clicked(self):          self.stop\_video\_processing = False          print("Ищем лица в видео")          # self.detect\_faces\_in\_video()          self.detect\_faces\_in\_video()      def on\_button\_mask\_clicked(self):          self.stop\_video\_processing = False          print("Ищем маски в видео")          # self.detect\_masks\_in\_video()          self.apply\_masks\_in\_video()      def on\_button\_stop\_clicked(self):          self.stop\_video\_processing = True      def on\_combo\_box\_changed(self, value):          self.img\_or\_video = value          self.img\_hide()          if self.img\_or\_video == 0:              self.combo\_box\_selectText.setVisible(0)              self.download\_video(1)              self.show\_video\_buttons(True)              self.show\_vtext(0)          elif self.img\_or\_video == 1:              self.download\_img(1)              self.img\_selectedshow()              self.show\_video\_buttons(False)      def on\_button\_vtext\_clicked(self):          print('Выделить текст')          self.detect\_text\_area()          self.select\_text\_show()      def on\_combo\_box\_selectText\_changed(self, value):          self.selected\_text\_index = value          if self.selected\_text\_index >= 0:              self.save\_text\_image()              self.extract\_text()              self.select\_text\_rectangle()      def create\_combo\_box\_selectText(self, numItems):          self.combo\_box\_selectText.clear()          for i in range(1, numItems + 1):              self.combo\_box\_selectText.addItem(f"Text {i}")      def save\_text\_image(self):          try:              if self.selected\_text\_index is not None and self.data is not None:                  x, y, w, h = self.data['left'][self.selected\_text\_index], self.data['top'][self.selected\_text\_index], \                               self.data['width'][self.selected\_text\_index], self.data['height'][self.selected\_text\_index]                  image = cv2.imread(self.initial\_path)                  selected\_text\_image = image[y:y+h, x:x+w]                  cv2.imwrite(save\_process\_path, selected\_text\_image)                  self.update\_image3(save\_process\_path)          except Exception as e:              print("Ошибка при сохранении участка изображения: ", e)      def extract\_text(self):          try:              if self.selected\_text\_index is not None and self.data is not None:                  text = self.data['text'][self.selected\_text\_index]                  print(f"Текст: {text}")                  QMessageBox.information(self, "Извлеченный текст", f"{text}")          except Exception as e:              print("Ошибка при выделении текста: ", e)      # сервисные функции      def download\_img(self, i):          try:              self.initial\_path, \_ = QFileDialog.getOpenFileName(self, "Выберите изображение", "", "Изображения (\*.png \*.jpg \*.jpeg)")              if not self.initial\_path:                  raise FileNotFoundError("Путь к изображению не был выбран.")              if i == 1:                  self.update\_images1(self.initial\_path)              else:                  raise FileNotFoundError("Куда ты хочешь картинку?")          except Exception as e:              print("Ошибка при загрузке изображения", e)              return None      def download\_video(self, i):          try:              self.initial\_path, \_ = QFileDialog.getOpenFileName(self, "Выберите видео", "", "Видео (\*.mp4)")              if not self.initial\_path:                  raise FileNotFoundError("Путь к видео не был выбран.")              if i == 1:                  self.cap = cv2.VideoCapture(self.initial\_path)                  \_, img = self.cap.read()                  cv2.imwrite(save\_process\_path, img)                  self.update\_images1(save\_process\_path)              else:                  raise FileNotFoundError("Куда ты хочешь картинку?")          except Exception as e:              print("Ошибка при загрузке видео", e)              return None      def video\_process1(self):          while True:              ret, img = self.cap.read()              if not ret:                  self.cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)                  continue              cv2.imwrite(save\_process\_path, img)              self.update\_images1(save\_process\_path)              if cv2.waitKey(50) & 0xFF == ord('q'):                  break      def video\_process2(self,oper='text'):          try:              tmp=0              while True:                  if self.stop\_video\_processing:  # Добавлено                      break                  tmp+=1                  if tmp%100000:                      continue                  ret, img = self.cap.read()                  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)                  \_, binary = cv2.threshold(gray, 150, 255, cv2.THRESH\_BINARY\_INV)                  self.label1\_title.hide()                  self.image\_label1.hide()                  if not ret:                      self.cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)                      continue                  if oper=='text':                      # Применяем OCR к изображению                      config = r'--oem 3 --psm 6'                      self.data = pytesseract.image\_to\_data(binary, config=config)                        for i, el in enumerate(self.data.splitlines()):                          if i ==0:                              continue                          el = el.split()                          try:                              x, y, w, h = int(el[6]), int(el[7]), int(el[8]), int(el[9])                              cv2.rectangle(img, (x,y), (w+x, h+y), (0,0,255),1)                              cv2.putText(img, el[11], (x,y), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255,255,255),2)                          except IndexError:                              print('')                      cv2.imwrite(save\_process\_path, img)                      self.update\_images2(save\_process\_path)                  if cv2.waitKey(1) & 0xFF == ord('q'):                      break          except Exception as e:              print("Ошибка при video\_process2: ", e)              return None      def detect\_text\_area(self):          try:              image = cv2.imread(self.initial\_path)              gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)              \_, binary = cv2.threshold(gray, 150, 255, cv2.THRESH\_BINARY\_INV)              # Применяем OCR к изображению              config = r'--oem 3 --psm 6'              self.data = pytesseract.image\_to\_data(binary, config=config, output\_type=pytesseract.Output.DICT)                number\_of\_boxes = 0              for i in range(len(self.data['text'])):                  # Получаем координаты и размеры текущего слова                  x, y, w, h = self.data['left'][i], self.data['top'][i], self.data['width'][i], self.data['height'][i]                  # Отфильтруем слова с низкой уверенностью                  #if int(self.data['conf'][i]) > 0:                      # Рисуем прямоугольник вокруг слова                  cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 1)                  number\_of\_boxes += 1                      # Отображаем номер слова                  cv2.putText(image, str(number\_of\_boxes), (x, y), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 0), 1)              self.textCount = number\_of\_boxes              cv2.imwrite(save\_process\_path, image)              self.update\_images2(save\_process\_path)              self.create\_combo\_box\_selectText(self.textCount)          except Exception as e:              print("Ошибка при detect\_text\_area: ", e)              return None      def video\_text(self):          print('ищу текст с видео')          self.stop\_video\_processing = False          self.video\_process2()      def detect\_faces\_in\_video(self):          try:              while True:                  if self.stop\_video\_processing:                      break                  ret, img = self.cap.read()                  if not ret:                      self.cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)                      continue                  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)                  faces = self.face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))                  for (x, y, w, h) in faces:                      cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)                  cv2.imwrite(save\_process\_path, img)                  self.update\_images2(save\_process\_path)                  if cv2.waitKey(50) & 0xFF == ord('q'):                      break          except Exception as e:              print("Ошибка при detect\_faces\_in\_video: ", e)      def apply\_masks\_in\_video(self):          try:              while True:                  if self.stop\_video\_processing:                      break                  ret, img = self.cap.read()                  if not ret:                      self.cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)                      continue                  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)                  faces = self.face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))                  for (x, y, w, h) in faces:                      mask\_resized = cv2.resize(self.mask\_image, (w, h))                      mask\_gray = mask\_resized[:, :, 3]                      \_, mask\_binary = cv2.threshold(mask\_gray, 1, 255, cv2.THRESH\_BINARY)                      mask\_binary\_inv = cv2.bitwise\_not(mask\_binary)                      mask\_rgb = mask\_resized[:, :, :3]                      roi = img[y:y+h, x:x+w]                      roi\_bg = cv2.bitwise\_and(roi, roi, mask=mask\_binary\_inv)                      roi\_fg = cv2.bitwise\_and(mask\_rgb, mask\_rgb, mask=mask\_binary)                      dst = cv2.add(roi\_bg, roi\_fg)                      img[y:y+h, x:x+w] = dst                  cv2.imwrite(save\_process\_path, img)                  self.update\_images2(save\_process\_path)                  if cv2.waitKey(50) & 0xFF == ord('q'):                      break          except Exception as e:              print("Ошибка при apply\_masks\_in\_video: ", e)    if \_\_name\_\_ == "\_\_main\_\_":      app = QApplication(sys.argv)      window = Mywindow()      sys.exit(app.exec()) |