ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО

Преподаватель департамента программной инженерии Факультета компьютерных наук

_____ И.М. Воронков

«____» _____ 2020 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор ДПИ ФКН

_____ В.В. Шилов

«____» _____ 2020 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Программа для классификации объектов мебели на фотографиях

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.13-01 12 01-1-ЛУ

Исполнитель

Студент группы БПИ171

_____ / Д. А. Потапенков /

«____» _____ 2020 г.

Москва 2020

УТВЕРЖДЕН

RU.17701729. 04.13-01 12 01-1-ЛУ

Программа для классификации объектов мебели на фотографиях

Текст программы

RU.17701729.04.13-01 12 01-1

Листов 45

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Москва 2020

ОГЛАВЛЕНИЕ

1. **MAIN.PY** 4
2. **MYAPP.PY** 4
3. **MAINWINDOW.PY** 10
4. **FILESLIST.PY** 16
5. **TRACKSLIST.PY** 20
6. **CLOTHESLIST.PY** 23
7. **CLOTHESPROCESSOR.PY** 24
8. **TRACKSPROCESSOR.PY** 27
9. **VIDEOPLAYERWIDGET.PY** 30
10. **VIDEOPLAYER.PY** 32
11. **VIDEOSTREAM.PY** 33
12. **VISUALIZER.PY** 35
13. **CATEGORY_PREDICTION.IPYNB** 36

1. Main.py

```
from PyQt5.QtWidgets import QApplication
from MyApp import MainApp
import sys

app = QApplication(sys.argv)
window = MainApp()
window.show()
sys.exit(app.exec_())
```

2. MyApp.py

```
from PyQt5 import QtWidgets
from PyQt5.QtCore import Qt, QTime
from PyQt5.QtWidgets import QStyle
from VideoPlayerWidget import VideoPlayerQWidget
from FilesProcessing import FilesGetter
from TrackList import TrackList
from FilesList import FilesList
from ClothesList import ClothesList
import MainWindow

from PyQt5.QtCore import QObject, QThread, pyqtSignal
import time

from functools import partial


class MainApp(QtWidgets.QMainWindow, MainWindow.Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.VideoWidget = VideoPlayerQWidget(self)
        self.TrackList = TrackList(self)
        self.FilesList = FilesList(self)
        self.ClothesList = ClothesList(self)

        self.addWidgets()
        self.makeConnects()

        self.clicked = False
        self.duration = 0
        self.UpdateRate = 24
        self.SliderFrozen = False
        self.Fined = False
        self.checked_clothes = []
        self.setEditBtns(False)

    '''
    Buttons
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
        '''
        # Files

        def _getCurRowName(self):
            if self.listWidgetFiles.currentRow() < 0:
                self.statusBar.showMessage("Select file")
                return
            item = self.listWidgetFiles.item(self.listWidgetFiles.currentRow())
            if item.flags() & Qt.ItemIsEnabled:
                return item.text()
            else:
                return ""

        def _preFilesClick(self):
            name = self._getCurRowName()
            if self.clicked:
                self.VideoWidget.release()
            self.btnPlay.setEnabled(False)
            self.sliderTime.setEnabled(False)
            self.TrackList.clearTracks()
            return name

        def btnOpen_clicked(self):
            name = self._preFilesClick()
            if name == "":
                return
            self.clicked = True

            tracks, tracks_path, clothes, clothes_path = self.FilesList.openFile(name)

            self.btnPlay.setEnabled(True)
            self.sliderTime.setEnabled(True)
            self.TrackList.clearTracks()
            if tracks_path != "" and clothes_path != "":
                self.TrackList.setInformation(tracks, clothes, tracks_path, clothes_path)
                return
            if tracks_path != "":
                self.setEditBtns(True)
                self.TrackList.setTracks(tracks, tracks_path)

        def btnTrack_clicked(self):
            name = self._preFilesClick()
            self.FilesList.trackFile(name)

        def btnClothes_clicked(self):
            name = self._preFilesClick()
            self.FilesList.clothesFile(name)

        def btnAdd_clicked(self):
            currentFileName = \
            QtWidgets.QFileDialog.getOpenFileName(None, "Select Video File",
    "D:/Programming/CourseWork_3_dev/output/video",
                                                  "*.mp4")[0]
            if currentFileName == "":
                return
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
        self.FilesList.addNewFile(currentFileName)

    def btnDeleteFiles_clicked(self):
        name = self._preFilesClick()
        self.VideoWidget.stopPlaying()
        self.TrackList.clearTracks()
        self.TrackList.applyTracks()
        self.FilesList.deleteFile(name)

    # Tracks

    def btnApply_clicked(self):
        self.TrackList.applyTracks()

    # Editing

    def btnConcat_clicked(self):
        if self.VideoWidget.status != self.VideoWidget.STATUS_PAUSE:
            self.DisplayMsg("You can split only on PAUSE")
            return
        self.TrackList.makeConcat()

    def btnSplit_clicked(self):
        if self.VideoWidget.status != self.VideoWidget.STATUS_PAUSE:
            self.DisplayMsg("You can split only on PAUSE")
            return
        self.TrackList.makeSplit(self.VideoWidget.getCurrentTime_frame())

    def btnReset_clicked(self):
        self.TrackList.resetChanges()

    def btnSave_clicked(self):
        self.TrackList.saveChanges()

    def btnDeleteEdit_clicked(self):
        self.TrackList.deleteTracks()

    # Clothes

    def btnFined_clicked(self):
        if self.clicked:
            self.VideoWidget.release()
        self.btnPlay.setEnabled(False)
        self.sliderTime.setEnabled(False)
        self.TrackList.clearTracks()
        self.checked_clothes = self.ClothesList.getCheckedClothes()
        if len(self.checked_clothes) > 0:
            self.Fined = True
            self.FilesList.displayWithClothes(self.checked_clothes)

    def btnWithout_clicked(self):
        self.FilesList.hideAll()
        self.FilesList.matchAll()
        self.FilesList.showMatched()
```

```
        # Other

        def btnPlay_clicked(self):
            if self.VideoWidget.status in (self.VideoWidget.STATUS_PAUSE,
    self.VideoWidget.STATUS_INIT):
                self.VideoWidget.play()
                self.btnPlay.setIcon(self.style().standardIcon(QStyle.SP_MediaPause))
            elif self.VideoWidget.status == self.VideoWidget.STATUS_PLAYING:
                self.VideoWidget.pause()
                self.btnPlay.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))

        '''
        Buttons Set
        '''

        def setClothesBtns(self, flag):
            self.btnFined.setEnabled(flag)
            self.btnWithout.setEnabled(flag)

        def setEditBtns(self, flag):
            self.btnSplit.setEnabled(flag)
            self.btnConcat.setEnabled(flag)
            self.btnSave.setEnabled(flag)
            self.btnReset.setEnabled(flag)
            self.btnDeleteEdit.setEnabled(flag)

        def setTracksBtns(self, state):
            self.btnApply.setEnabled(state)
            self.btnCheckAll.setEnabled(state)
            self.btnUncheckAll.setEnabled(state)

        def set_btnPlay(self):
            self.btnPlay.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))

        def freezeSlider(self):
            self.SliderFrozen = True

        def makeConnects(self):
            self.btnOpen.clicked.connect(self.btnOpen_clicked)
            self.sliderTime.sliderReleased.connect(self.set_time)
            self.sliderTime.sliderPressed.connect(self.freezeSlider)
            self.btnPlay.clicked.connect(self.btnPlay_clicked)
            self.btnApply.clicked.connect(self.btnApply_clicked)
            self.cbPlaySpeed.activated[str].connect(self.cbPlaySpeed_changed)
            self.cbRewindRate.activated[str].connect(self.cbRewindRate_changed)
            self.btnCheckAll.clicked.connect(self.TrackList.checkAll)
            self.btnUncheckAll.clicked.connect(self.TrackList.uncheckAll)
            self.btnAdd.clicked.connect(self.btnAdd_clicked)
            self.btnSplit.clicked.connect(self.btnSplit_clicked)
            self.btnClothes.clicked.connect(self.btnClothes_clicked)
            self.btnTrack.clicked.connect(self.btnTrack_clicked)
            self.btnReset.clicked.connect(self.btnReset_clicked)
            self.btnSave.clicked.connect(self.btnSave_clicked)
            self.btnFined.clicked.connect(self.btnFined_clicked)
            self.btnConcat.clicked.connect(self.btnConcat_clicked)
```

```
        self.btnWithout.clicked.connect(self.btnWithout_clicked)
        self.btnDeleteEdit.clicked.connect(self.btnDeleteEdit_clicked)
        self.btnDeleteFiles.clicked.connect(self.btnDeleteFiles_clicked)

    '''
    Video methods
    '''

    def set_time(self):
        self.SliderFrozen = False
        value = self.sliderTime.value()
        if self.cbRewindRate.currentText() == "Sec":
            value *= 24
        self.VideoWidget.setFrame(value)

    def set_slider_time(self, value):
        if not self.SliderFrozen:
            self.sliderTime.setValue(value)
        if  self.cbRewindRate.currentText() == "Frame":
            value = value // 24
        self.updateDurationInfo(value)

    def updateDurationInfo(self, currentInfo):
        duration = self.duration // 24
        if currentInfo or duration:
            currentTime = QTime((currentInfo / 3600) % 60, (currentInfo / 60) % 60,
                                currentInfo % 60)
            totalTime = QTime((duration / 3600) % 60, (duration / 60) % 60,
                              duration % 60);

            format = 'hh:mm:ss' if duration > 3600 else 'mm:ss'
            tStr = currentTime.toString(format) + " / " + totalTime.toString(format)
        else:
            tStr = ""

        self.labelDuration.setText(tStr)

    def openVideo(self, path):
        self.VideoWidget.setVideo(path)
        self.duration = self.VideoWidget.getFrameCount()
        self.sliderTime.setRange(0, self.duration // self.UpdateRate)
        self.sliderTime.setValue(0)
        self.updateDurationInfo(0)

    def cbPlaySpeed_changed(self, text):
        self.VideoWidget.setPlaySpeed(float(text))

    def cbRewindRate_changed(self, text):
        if text == "Sec":
            self.UpdateRate = 24
        elif text == "Frame":
            self.UpdateRate = 1

        self.VideoWidget.setUpdateRate(self.UpdateRate)
        self.sliderTime.setRange(0, self.duration // self.UpdateRate)
```

```
            self.set_slider_time(self.VideoWidget.getCurrentTime())

        '''
        For Lists
        '''


        def GetTracksList(self):
            return self.verticalLayoutListTracks

        def GetFilesList(self):
            return self.listWidgetFiles

        def GetClothesList(self):
            return self.verticalLayoutClothes

        def addCHeckBoxTrack(self, name):
            wid = QtWidgets.QCheckBox(self.scrollAreaWidgetContentsTracks)
            wid.setText(f"{name}")
            wid.setChecked(True)
            self.verticalLayoutListTracks.addWidget(wid, 0, alignment=Qt.AlignTop)
            return wid

        def addCHeckBoxClothes(self, name):
            wid = QtWidgets.QCheckBox(self.scrollAreaWidgetContentsClothes)
            wid.setText(f"{name}")
            wid.setChecked(False)
            self.verticalLayoutClothes.addWidget(wid, 0, alignment=Qt.AlignTop)
            return wid

        '''
        Other
        '''

        def DisplayMsg(self, str):
            self.statusBar.showMessage(str)

        def addWidgets(self):
            sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
    QtWidgets.QSizePolicy.Preferred)
            self.VideoWidget.setSizePolicy(sizePolicy)
            self.VideoWidget.setStyleSheet("background-color: rgb(255, 255, 255);")
            self.verticalLayoutMain.insertWidget(0, self.VideoWidget)
            self.btnPlay.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))

        def closeEvent(self, event):
            self.FilesList.endProcessing()
            self.VideoWidget.release()
```

3. MainWindow.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'MainWindow.ui'
#
# Created by: PyQt5 UI code generator 5.13.0
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1121, 662)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.layoutMain = QtWidgets.QHBoxLayout(self.centralwidget)
        self.layoutMain.setObjectName("layoutMain")
        self.groupBoxVideo = QtWidgets.QGroupBox(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.groupBoxVideo.sizePolicy().hasHeightForWidth())
        self.groupBoxVideo.setSizePolicy(sizePolicy)
        self.groupBoxVideo.setObjectName("groupBoxVideo")
        self.verticalLayoutMain = QtWidgets.QVBoxLayout(self.groupBoxVideo)
        self.verticalLayoutMain.setContentsMargins(5, 5, 5, 5)
        self.verticalLayoutMain.setObjectName("verticalLayoutMain")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setSpacing(2)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.btnPlay = QtWidgets.QPushButton(self.groupBoxVideo)
        self.btnPlay.setEnabled(False)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.btnPlay.sizePolicy().hasHeightForWidth())
        self.btnPlay.setSizePolicy(sizePolicy)
        self.btnPlay.setMinimumSize(QtCore.QSize(0, 0))
        self.btnPlay.setMaximumSize(QtCore.QSize(25, 16777215))
        self.btnPlay.setSizeIncrement(QtCore.QSize(0, 0))
        self.btnPlay.setBaseSize(QtCore.QSize(0, 0))
        self.btnPlay.setText("")
        self.btnPlay.setObjectName("btnPlay")
        self.horizontalLayout.addWidget(self.btnPlay)
        self.cbPlaySpeed = QtWidgets.QComboBox(self.groupBoxVideo)
        self.cbPlaySpeed.setMaximumSize(QtCore.QSize(40, 16777215))
```

```
        self.cbPlaySpeed.setObjectName("cbPlaySpeed")
        self.cbPlaySpeed.addItem("")
        self.cbPlaySpeed.addItem("")
        self.horizontalLayout.addWidget(self.cbPlaySpeed)
        self.sliderTime = QtWidgets.QSlider(self.groupBoxVideo)
        self.sliderTime.setMinimumSize(QtCore.QSize(100, 0))
        self.sliderTime.setOrientation(QtCore.Qt.Horizontal)
        self.sliderTime.setObjectName("sliderTime")
        self.horizontalLayout.addWidget(self.sliderTime)
        self.cbRewindRate = QtWidgets.QComboBox(self.groupBoxVideo)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.cbRewindRate.sizePolicy().hasHeightForWidth())
        self.cbRewindRate.setSizePolicy(sizePolicy)
        self.cbRewindRate.setMaximumSize(QtCore.QSize(55, 16777215))
        self.cbRewindRate.setEditable(False)
        self.cbRewindRate.setObjectName("cbRewindRate")
        self.cbRewindRate.addItem("")
        self.cbRewindRate.addItem("")
        self.horizontalLayout.addWidget(self.cbRewindRate)
        self.labelDuration = QtWidgets.QLabel(self.groupBoxVideo)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.labelDuration.sizePolicy().hasHeightForWidth())
        self.labelDuration.setSizePolicy(sizePolicy)
        self.labelDuration.setMaximumSize(QtCore.QSize(16777215, 21))
        self.labelDuration.setObjectName("labelDuration")
        self.horizontalLayout.addWidget(self.labelDuration)
        self.verticalLayoutMain.addLayout(self.horizontalLayout)
        self.layoutMain.addWidget(self.groupBoxVideo)
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setObjectName("groupBox")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.groupBox)
        self.verticalLayout_2.setContentsMargins(2, 2, 2, 2)
        self.verticalLayout_2.setSpacing(3)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.groupBox_2 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_2.setObjectName("groupBox_2")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.groupBox_2)
        self.verticalLayout_3.setContentsMargins(4, 3, 4, 3)
        self.verticalLayout_3.setSpacing(3)
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.btnCheckAll = QtWidgets.QPushButton(self.groupBox_2)
        self.btnCheckAll.setEnabled(False)
        self.btnCheckAll.setMaximumSize(QtCore.QSize(1000, 16777215))
        self.btnCheckAll.setObjectName("btnCheckAll")
        self.verticalLayout_3.addWidget(self.btnCheckAll)
        self.btnUncheckAll = QtWidgets.QPushButton(self.groupBox_2)
        self.btnUncheckAll.setEnabled(False)
```

```
        self.btnUncheckAll.setMaximumSize(QtCore.QSize(100, 16777215))
        self.btnUncheckAll.setObjectName("btnUncheckAll")
        self.verticalLayout_3.addWidget(self.btnUncheckAll)
        spacerItem = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.verticalLayout_3.addItem(spacerItem)
        self.btnApply = QtWidgets.QPushButton(self.groupBox_2)
        self.btnApply.setEnabled(False)
        self.btnApply.setMaximumSize(QtCore.QSize(1000, 16777215))
        self.btnApply.setObjectName("btnApply")
        self.verticalLayout_3.addWidget(self.btnApply)
        self.verticalLayout_2.addWidget(self.groupBox_2)
        self.groupBox_3 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_3.setObjectName("groupBox_3")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.groupBox_3)
        self.verticalLayout_4.setContentsMargins(4, 3, 4, 3)
        self.verticalLayout_4.setSpacing(3)
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.btnAdd = QtWidgets.QPushButton(self.groupBox_3)
        self.btnAdd.setObjectName("btnAdd")
        self.verticalLayout_4.addWidget(self.btnAdd)
        self.btnTrack = QtWidgets.QPushButton(self.groupBox_3)
        self.btnTrack.setObjectName("btnTrack")
        self.verticalLayout_4.addWidget(self.btnTrack)
        self.btnClothes = QtWidgets.QPushButton(self.groupBox_3)
        self.btnClothes.setObjectName("btnClothes")
        self.verticalLayout_4.addWidget(self.btnClothes)
        self.btnDeleteFiles = QtWidgets.QPushButton(self.groupBox_3)
        self.btnDeleteFiles.setObjectName("btnDeleteFiles")
        self.verticalLayout_4.addWidget(self.btnDeleteFiles)
        spacerItem1 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.verticalLayout_4.addItem(spacerItem1)
        self.btnOpen = QtWidgets.QPushButton(self.groupBox_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.btnOpen.sizePolicy().hasHeightForWidth())
        self.btnOpen.setSizePolicy(sizePolicy)
        self.btnOpen.setAutoFillBackground(False)
        self.btnOpen.setObjectName("btnOpen")
        self.verticalLayout_4.addWidget(self.btnOpen)
        self.verticalLayout_2.addWidget(self.groupBox_3)
        self.groupBox_4 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_4.setObjectName("groupBox_4")
        self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.groupBox_4)
        self.verticalLayout_5.setContentsMargins(4, 3, 4, 3)
        self.verticalLayout_5.setSpacing(3)
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.btnSplit = QtWidgets.QPushButton(self.groupBox_4)
        self.btnSplit.setObjectName("btnSplit")
        self.verticalLayout_5.addWidget(self.btnSplit)
        self.btnConcat = QtWidgets.QPushButton(self.groupBox_4)
        self.btnConcat.setEnabled(True)
```

```
        self.btnConcat.setObjectName("btnConcat")
        self.verticalLayout_5.addWidget(self.btnConcat)
        self.btnDeleteEdit = QtWidgets.QPushButton(self.groupBox_4)
        self.btnDeleteEdit.setObjectName("btnDeleteEdit")
        self.verticalLayout_5.addWidget(self.btnDeleteEdit)
        spacerItem2 = QtWidgets.QSpacerItem(20, 10, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Fixed)
        self.verticalLayout_5.addItem(spacerItem2)
        self.btnSave = QtWidgets.QPushButton(self.groupBox_4)
        self.btnSave.setObjectName("btnSave")
        self.verticalLayout_5.addWidget(self.btnSave)
        self.btnReset = QtWidgets.QPushButton(self.groupBox_4)
        self.btnReset.setObjectName("btnReset")
        self.verticalLayout_5.addWidget(self.btnReset)
        self.verticalLayout_2.addWidget(self.groupBox_4)
        self.groupBox_5 = QtWidgets.QGroupBox(self.groupBox)
        self.groupBox_5.setObjectName("groupBox_5")
        self.verticalLayout_9 = QtWidgets.QVBoxLayout(self.groupBox_5)
        self.verticalLayout_9.setContentsMargins(4, 3, 4, 3)
        self.verticalLayout_9.setSpacing(3)
        self.verticalLayout_9.setObjectName("verticalLayout_9")
        self.btnWithout = QtWidgets.QPushButton(self.groupBox_5)
        self.btnWithout.setObjectName("btnWithout")
        self.verticalLayout_9.addWidget(self.btnWithout)
        self.btnFined = QtWidgets.QPushButton(self.groupBox_5)
        self.btnFined.setObjectName("btnFined")
        self.verticalLayout_9.addWidget(self.btnFined)
        self.verticalLayout_2.addWidget(self.groupBox_5)
        spacerItem3 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
        self.verticalLayout_2.addItem(spacerItem3)
        self.layoutMain.addWidget(self.groupBox)
        self.groupBoxTracks = QtWidgets.QGroupBox(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.groupBoxTracks.sizePolicy().hasHeightForWidth())
        self.groupBoxTracks.setSizePolicy(sizePolicy)
        self.groupBoxTracks.setMaximumSize(QtCore.QSize(170, 16777215))
        self.groupBoxTracks.setObjectName("groupBoxTracks")
        self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.groupBoxTracks)
        self.verticalLayout_6.setContentsMargins(2, 2, 2, 2)
        self.verticalLayout_6.setSpacing(2)
        self.verticalLayout_6.setObjectName("verticalLayout_6")
        self.lineEdit = QtWidgets.QLineEdit(self.groupBoxTracks)
        self.lineEdit.setMaximumSize(QtCore.QSize(16777215, 0))
        self.lineEdit.setObjectName("lineEdit")
        self.verticalLayout_6.addWidget(self.lineEdit)
        self.scrollAreaTracks = QtWidgets.QScrollArea(self.groupBoxTracks)
        self.scrollAreaTracks.setAutoFillBackground(True)
        self.scrollAreaTracks.setStyleSheet("background-color: rgb(255, 255, 255);")
        self.scrollAreaTracks.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.scrollAreaTracks.setFrameShadow(QtWidgets.QFrame.Plain)
```

```
        self.scrollAreaTracks.setWidgetResizable(True)
        self.scrollAreaTracks.setObjectName("scrollAreaTracks")
        self.scrollAreaWidgetContentsTracks = QtWidgets.QWidget()
        self.scrollAreaWidgetContentsTracks.setGeometry(QtCore.QRect(0, 0, 131, 601))
        self.scrollAreaWidgetContentsTracks.setAutoFillBackground(False)

self.scrollAreaWidgetContentsTracks.setObjectName("scrollAreaWidgetContentsTracks")
        self.verticalLayoutListTracks =
QtWidgets.QVBoxLayout(self.scrollAreaWidgetContentsTracks)
        self.verticalLayoutListTracks.setObjectName("verticalLayoutListTracks")
        self.scrollAreaTracks.setWidget(self.scrollAreaWidgetContentsTracks)
        self.verticalLayout_6.addWidget(self.scrollAreaTracks)
        self.layoutMain.addWidget(self.groupBoxTracks)
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setSizeConstraint(QtWidgets.QLayout.SetDefaultConstraint)
        self.verticalLayout.setSpacing(2)
        self.verticalLayout.setObjectName("verticalLayout")
        self.groupBoxFiles = QtWidgets.QGroupBox(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(1)

sizePolicy.setHeightForWidth(self.groupBoxFiles.sizePolicy().hasHeightForWidth())
        self.groupBoxFiles.setSizePolicy(sizePolicy)
        self.groupBoxFiles.setMaximumSize(QtCore.QSize(150, 16777215))
        self.groupBoxFiles.setObjectName("groupBoxFiles")
        self.verticalLayout_8 = QtWidgets.QVBoxLayout(self.groupBoxFiles)
        self.verticalLayout_8.setContentsMargins(2, 2, 2, 2)
        self.verticalLayout_8.setObjectName("verticalLayout_8")
        self.listWidgetFiles = QtWidgets.QListWidget(self.groupBoxFiles)
        self.listWidgetFiles.setObjectName("listWidgetFiles")
        self.verticalLayout_8.addWidget(self.listWidgetFiles)
        self.verticalLayout.addWidget(self.groupBoxFiles)
        self.groupBoxClothes = QtWidgets.QGroupBox(self.centralwidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(2)

sizePolicy.setHeightForWidth(self.groupBoxClothes.sizePolicy().hasHeightForWidth())
        self.groupBoxClothes.setSizePolicy(sizePolicy)
        self.groupBoxClothes.setMaximumSize(QtCore.QSize(150, 16777215))
        self.groupBoxClothes.setObjectName("groupBoxClothes")
        self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.groupBoxClothes)
        self.verticalLayout_7.setContentsMargins(2, 2, 2, 2)
        self.verticalLayout_7.setObjectName("verticalLayout_7")
        self.scrollAreaClothes = QtWidgets.QScrollArea(self.groupBoxClothes)
        self.scrollAreaClothes.setStyleSheet("background-color: rgb(255, 255, 255);")
        self.scrollAreaClothes.setWidgetResizable(True)
        self.scrollAreaClothes.setObjectName("scrollAreaClothes")
        self.scrollAreaWidgetContentsClothes = QtWidgets.QWidget()
        self.scrollAreaWidgetContentsClothes.setGeometry(QtCore.QRect(0, 0, 142,
392))
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
self.scrollAreaWidgetContentsClothes.setObjectName("scrollAreaWidgetContentsClothes")
        self.verticalLayoutClothes =
QtWidgets.QVBoxLayout(self.scrollAreaWidgetContentsClothes)
        self.verticalLayoutClothes.setObjectName("verticalLayoutClothes")
        self.scrollAreaClothes.setWidget(self.scrollAreaWidgetContentsClothes)
        self.verticalLayout_7.addWidget(self.scrollAreaClothes)
        self.verticalLayout.addWidget(self.groupBoxClothes)
        self.layoutMain.addLayout(self.verticalLayout)
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusBar = QtWidgets.QStatusBar(MainWindow)
        self.statusBar.setObjectName("statusBar")
        MainWindow.setStatusBar(self.statusBar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.groupBoxVideo.setTitle(_translate("MainWindow", "VideoPlayer"))
        self.cbPlaySpeed.setItemText(0, _translate("MainWindow", "1"))
        self.cbPlaySpeed.setItemText(1, _translate("MainWindow", "0.2"))
        self.cbRewindRate.setItemText(0, _translate("MainWindow", "Sec"))
        self.cbRewindRate.setItemText(1, _translate("MainWindow", "Frame"))
        self.labelDuration.setText(_translate("MainWindow", "00:00/00:00"))
        self.groupBox.setTitle(_translate("MainWindow", "Buttons"))
        self.groupBox_2.setTitle(_translate("MainWindow", "Tracks"))
        self.btnCheckAll.setText(_translate("MainWindow", "Check"))
        self.btnUncheckAll.setText(_translate("MainWindow", "Uncheck"))
        self.btnApply.setText(_translate("MainWindow", "Apply"))
        self.groupBox_3.setTitle(_translate("MainWindow", "Files"))
        self.btnAdd.setText(_translate("MainWindow", "Add"))
        self.btnTrack.setText(_translate("MainWindow", "Track"))
        self.btnClothes.setText(_translate("MainWindow", "Clothes"))
        self.btnDeleteFiles.setText(_translate("MainWindow", "Delete"))
        self.btnOpen.setText(_translate("MainWindow", "View"))
        self.groupBox_4.setTitle(_translate("MainWindow", "Editing"))
        self.btnSplit.setText(_translate("MainWindow", "Split"))
        self.btnConcat.setText(_translate("MainWindow", "Concat"))
        self.btnDeleteEdit.setText(_translate("MainWindow", "Delete"))
        self.btnSave.setText(_translate("MainWindow", "Save"))
        self.btnReset.setText(_translate("MainWindow", "Reset"))
        self.groupBox_5.setTitle(_translate("MainWindow", "Clothes"))
        self.btnWithout.setText(_translate("MainWindow", "Without"))
        self.btnFined.setText(_translate("MainWindow", "Finde"))
        self.groupBoxTracks.setTitle(_translate("MainWindow", "Tracks"))
        self.groupBoxFiles.setTitle(_translate("MainWindow", "Files"))
        self.groupBoxClothes.setTitle(_translate("MainWindow", "Clothes"))
```

4. FilesList.py

```
from PyQt5 import QtWidgets
from PyQt5.QtCore import Qt, QThread
from TrackerProcessor import SingleTrackerProcessor
from ClothesProcessor import SingleClothesProcessor
from FilesProcessing import FilesGetter

from functools import partial
from pathlib import Path
import os
import numpy as np
import shutil
import threading
import json

class FilesList:
    LOADED = 0
    TRACKED = 1
    FOUND = 2
    types = ["❌", "⚠", "✅"]

    def __init__(self, parent):
        self.video_folder = "data\\video\\"
        self.tracks_folder = "data\\tracks\\"
        self.clothes_folder = "data\\clothes\\"
        self.files = dict()
        self.parent = parent
        self.filesListWidget = self.parent.GetFilesList()
        self.readProcessedFiles()
        self.counter = 0
        self.threads = dict()

    def readProcessedFiles(self):
        videos = os.listdir(self.video_folder)
        for video in videos:
            file_name, file_type = video.rsplit('.', 1)
            if file_type != "mp4":
                continue

            paths = FilesGetter.finedAllFiles(self.video_folder + file_name)
            if paths[1] == "":
                self.addToList(file_name, self.LOADED, paths)
                continue
            tracks_file = open(paths[1], "r")
            tracks = self.parsTracks(tracks_file.readlines())
            tracks_file.close()
            if tracks.shape[0] == 0:
                self.addToList(file_name, self.LOADED, paths)
                continue

            if paths[2] == "":
                self.addToList(file_name, self.TRACKED, paths)
                continue
            # clothes_file = open(paths[2], "r")
```

```
            # clothes = self.parsTracks(clothes_file.readlines())
            # clothes_file.close()
            self.addToList(file_name, self.FOUND, paths)

    def addToList(self, file_name, type, paths):
        item = QtWidgets.QListWidgetItem()
        item.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)
        item.setText(f"{self.types[type]}{file_name}")
        self.filesListWidget.addItem(item)
        self.files[file_name] = [item, type, paths, True]

    def parsTracks(self, tracks):
        tracks_list = []
        for line in tracks:
            line = line.strip().split(",")
            if len(line) != 10:
                self.parent.DisplayMsg("⚠ Can't parse tracks file")
                tracks_list = []
                break
            try:
                line = list(map(int, map(float, line)))
            except Exception as ex:
                self.parent.DisplayMsg("⚠ Can't parse tracks file")
                tracks_list = []
                break
            tracks_list.append(line)

        return np.array(tracks_list)

    def addNewFile(self, path_from):
        my_thread = threading.Thread(target=self.copy, args=(path_from,))
        my_thread.start()

    def trackFile(self, file_name):
        file_name = file_name.strip("❌⚠✅")
        if file_name not in self.files:
            return
        if self.files[file_name][1] != self.LOADED:
            self.parent.DisplayMsg(f"⚠ File '{file_name}' already tracked!")
            return
        video_path = self.video_folder + file_name + ".mp4"
        tracks_path = self.tracks_folder + file_name + ".txt"
        processor = SingleTrackerProcessor(video_path, tracks_path)
        self.runTread(processor, file_name, self.TRACKED)

    def clothesFile(self, file_name):
        file_name = file_name.strip("❌⚠✅")
        if file_name not in self.files:
            return
        if self.files[file_name][1] == self.LOADED:
            self.parent.DisplayMsg(f"⚠ File '{file_name}' not tracked yet!")
            return
        if self.files[file_name][1] == self.FOUND:
            self.parent.DisplayMsg(f"⚠ File '{file_name}' already founded(clothes)
```

```
!")
            return
        video_path = self.video_folder + file_name + ".mp4"
        tracks_path = self.tracks_folder + file_name + ".txt"
        clothes_path = self.clothes_folder + file_name + ".txt"
        processor = SingleClothesProcessor(video_path, tracks_path, clothes_path)
        self.runTread(processor, file_name, self.FOUND)

    def runTread(self, obj, name, type):
        self.counter += 1
        cur_num = self.counter % 20
        item = self.files[name][0]
        item.setFlags(Qt.NoItemFlags)
        self.threads[cur_num] = (obj, QThread())
        self.threads[cur_num][0].moveToThread(self.threads[cur_num][1])
        self.threads[cur_num][0].finished.connect(partial(self.end_thread, cur_num))
        self.threads[cur_num][0].percent.connect(partial(self.percentChange, item))
        self.threads[cur_num][1].started.connect(self.threads[cur_num][0].process)
        self.threads[cur_num][1].finished.connect(partial(self.release_thread,
cur_num, item, name, type))
        self.threads[cur_num][1].start()


    def percentChange(self, item, percent):
       name = item.text().split("|", 1)[-1]
       item.setText(f"{percent}%|{name}")

    def end_thread(self, name):
        self.threads[name][1].quit()

    def release_thread(self, num, item, name, type):
        del self.threads[num]
        item.setText(f"{self.types[type]}{name}")
        item.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled)
        paths = FilesGetter.finedAllFiles(self.video_folder + name + ".mp4")
        self.files[name][1] = type
        self.files[name][2] = paths

    def endProcessing(self):
        for name, process in self.threads.items():
            process[0].setStopped()
            process[1].quit()
            process[1].wait()

    def copy(self, path_from):
        parts = list(Path(path_from).parts)
        file_name = parts[-1].rsplit('.', 1)[0]
        if file_name in self.files:
            self.parent.DisplayMsg(f"⚠ File '{file_name}' already added!")
            return
        path_to = os.path.join(self.video_folder, parts[-1])
        shutil.copy(path_from, path_to)
        paths = FilesGetter.finedAllFiles(path_to)
        self.addToList(file_name, self.LOADED, paths)
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
    def openFile(self, name):
        name = name.strip("✕⚠✓")
        ret = [np.array([]), "", dict(), ""]
        if name not in self.files:
            return ret
        self.parent.openVideo(self.video_folder + name + ".mp4")
        if self.files[name][1] in (self.TRACKED, self.FOUND):
            self.parent.setTracksBtns(True)
            tracks_file = open(self.files[name][2][1], "r")
            tracks = self.parsTracks(tracks_file.readlines())
            tracks_file.close()
            ret[0] = tracks
            ret[1] = self.files[name][2][1]
            if self.files[name][1] == self.FOUND:
                clothes_file = open(self.files[name][2][2], "r")
                clothes = json.load(clothes_file)
                clothes_file.close()
                self.parent.setClothesBtns(True)
                ret[2] = clothes
                ret[3] = self.files[name][2][2]

        return ret

    def deleteFile(self, name):
        #TODO fix a lot of bugs
        name = name.strip("✕⚠✓")
        if name not in self.files:
            return
        self.files[name][0] =
self.filesListWidget.takeItem(self.filesListWidget.row(self.files[name][0]))
        if self.files[name][2][0] != "":
            self.files[name][2][0] += ".mp4"
        for path in self.files[name][2][:-1]:
            if path != "":
                try:
                    os.remove(path)
                except:
                    print("")
        del self.files[name]

    def hideAll(self):
        for file in self.files:
            if self.files[file][3]:
                self.files[file][0] =
self.filesListWidget.takeItem(self.filesListWidget.row(self.files[file][0]))

    def displayWithClothes(self, checked_clothes):
        self.hideAll()
        for file in self.files:
            self.files[file][3] = False
            if self.files[file][1] == self.FOUND:
                clothes_file = open(self.files[file][2][2], "r")
                clothes = json.load(clothes_file)
                clothes_file.close()
                for track in clothes:
```

```
                    flagIn = any(map(lambda v: v in clothes[track], checked_clothes))
                    if flagIn:
                        self.files[file][3] = True
                        break
        self.showMatched()

    def showMatched(self):
        for file in self.files:
            if self.files[file][3]:
                self.filesListWidget.addItem(self.files[file][0])

    def matchAll(self):
        for file in self.files:
            self.files[file][3] = True
```

5. TracksList.py

```
from PyQt5 import QtWidgets
from PyQt5.QtCore import Qt
import numpy as np
from Visualizer import Visualizer


class TrackList:
    def __init__(self, parent):
        self.ch_tracks = dict()
        self.parent = parent
        self.parsed = False
        self.listWidget = self.parent.GetTracksList()
        self.Visualizer = Visualizer.getInstance()
        self.last_id = -1e3
        self.np_tracks = np.zeros(0)
        self.clothes = dict()
        self.tracks_path = ""
        self.clothes_path = ""

    def setInformation(self, tracks, clothes, tracks_path, clothes_path):
        self.clothes_path = clothes_path
        self.clothes = clothes
        self.setTracks(tracks, tracks_path)

    def setTracks(self, tracks, path):
        self.tracks_path = path
        self.np_tracks = np.copy(tracks)
        self.copy_np_tracks = np.copy(self.np_tracks)
        tracks_ids = np.unique(self.np_tracks[:, 1])
        self.last_id = tracks_ids[-1]
        self.Visualizer.setTracksInfo(self.np_tracks)
        self.displayTrackList()

    def displayTrackList(self):
        tracks_ids = np.unique(self.np_tracks[:, 1])
        for track in tracks_ids:
```

```
            clothes = self.getTrackClothes(track)
            flagIn = any(map(lambda v: v in clothes, self.parent.checked_clothes))
            if not self.parent.Fined or flagIn:
                self.add_chbox(f"{str(track)} ({', '.join(clothes)})", track)
        self.listWidget.addStretch()

    def getTrackClothes(self, track):
        track = str(track)
        ret = []
        if track in self.clothes:
            ret = self.clothes[track]
        return ret

    def add_chbox(self, text, track):
        wid = self.parent.addCHeckBoxTrack(text)
        self.ch_tracks[track] = wid

    def add_chbox_new(self, text, track):
        self.clearTracks(False)
        wid = self.parent.addCHeckBoxTrack(text)
        self.ch_tracks[track] = wid
        self.listWidget.addStretch()

    def clearTracks(self, all=True):
        for i in reversed(range(self.listWidget.count())):
            item = self.listWidget.itemAt(i)
            if item.spacerItem():
                self.listWidget.removeItem(item)
            elif self.listWidget.itemAt(i).widget() and all:
                self.listWidget.itemAt(i).widget().setParent(None)
            else:
                pass

    def getCheckedTracks(self):
        checked = []
        for id, ch_track in self.ch_tracks.items():
            if ch_track.isChecked():
                checked.append(id)
        return checked

    def checkAll(self):
        for _, ch_track in self.ch_tracks.items():
            ch_track.setCheckState(True)

    def uncheckAll(self):
        for _, ch_track in self.ch_tracks.items():
            ch_track.setCheckState(False)

    def applyTracks(self):
        self.Visualizer.setTracks(self.getCheckedTracks())

    def makeSplit(self, frame):
        checked = self.getCheckedTracks()
        if len(checked) != 1:
            self.parent.DisplayMsg("For split you should check one and only one
```

```
track!")
            return
        cur_track = self.np_tracks[self.np_tracks[:, 1] == checked[0]]
        prev = np.sum(cur_track[:, 0] <= frame)
        post = np.sum(cur_track[:, 0] > frame)
        if prev == 0:
            self.parent.DisplayMsg("First part of track is empty")
            return
        if post == 0:
            self.parent.DisplayMsg("Second part of track is empty")
            return

        print("Frame to split: ", frame)
        mask = (self.np_tracks[:, 0] > frame) & (self.np_tracks[:, 1] == checked[0])
        self.np_tracks[mask, 1] = self.last_id + 1
        self.last_id += 1
        self.add_chbox_new(self.last_id, self.last_id)
        self.Visualizer.setTracksInfo(self.np_tracks)

    def makeConcat(self):
        checked = self.getCheckedTracks()
        if len(checked) != 2:
            self.parent.DisplayMsg("For concat you should check two and only two
track!")
            return

        first_track = self.np_tracks[self.np_tracks[:, 1] == checked[0]]
        second_track = self.np_tracks[self.np_tracks[:, 1] == checked[1]]
        first_frames = set(np.unique(first_track[:, 0]))
        second_frames = set(np.unique(second_track[:, 0]))

        if len(first_frames & second_frames) > 4:
            self.parent.DisplayMsg("Tracks can overlap in no more than 4 frames")
            return

        mask = (self.np_tracks[:, 1] == checked[1])
        self.np_tracks[mask, 1] = checked[0]
        self.ch_tracks[checked[1]].setParent(None)
        del self.ch_tracks[checked[1]]
        self.Visualizer.setTracksInfo(self.np_tracks)

    def deleteTracks(self):
        checked = self.getCheckedTracks()
        mask = np.array([True] * self.np_tracks.shape[0])

        for track in checked:
            mask = mask & (self.np_tracks[:, 1] == track)
            self.ch_tracks[track].setParent(None)
            del self.ch_tracks[track]

        self.np_tracks = self.np_tracks[mask, :]
        self.Visualizer.setTracksInfo(self.np_tracks)

    def resetChanges(self):
        self.np_tracks = np.copy(self.copy_np_tracks)
```

```
            self.clearTracks()
            self.displayTrackList()

        def saveChanges(self):
            tracks_file = open(self.tracks_path, "w")
            for line in self.np_tracks.shape[0]:
                print('%d,%d,%.2f,%.2f,%.2f,%.2f,1,-1,-1,-1' % (
                    line[0], line[1], line[2], line[3], line[4], line[5]),
                      file=tracks_file)
            tracks_file.close()
```

6. ClothesList.py

```
class ClothesList:
    def __init__(self, parent):
        self.parent = parent
        self.cats_path = "D:\Programming\CourseWork_3\code\data\categories.txt"
        self.listWidget = self.parent.GetClothesList()
        self.clothes_list = dict()
        self.displayClothesList()

    def displayClothesList(self):
        categories = self._getCategoriesList()
        for clothes in categories:
            self.add_chbox(clothes)
        self.listWidget.addStretch()

    def add_chbox(self, clothes):
        wid = self.parent.addCHeckBoxClothes(clothes)
        self.clothes_list[clothes] = wid

    def getCheckedClothes(self):
        checked = []
        for id, clothes in self.clothes_list.items():
            if clothes.isChecked():
                checked.append(id)
        return checked

    def _getCategoriesList(self):
        cats_file = open(self.cats_path, 'r').readlines()
        categories = []

        for ln in cats_file:
            cur = list(filter(None, ln[:-1].split(' ')))
            if cur[1] in ("1", "2", "3"):
                categories.append(cur[0])

        return categories
```

7. ClothesProcessor.py

```
from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot
from PIL import Image
from collections import Counter
import torchvision

import torch
import numpy as np
import cv2
import json
import os


from pprint import pprint



class SingleClothesProcessor(QObject):
    finished = pyqtSignal()
    percent = pyqtSignal(int)


    def __init__(self, path_video, path_tracks, clothes_path):
        super().__init__()
        self.path_video = path_video
        self.path_tracks = path_tracks
        self.clothes_path = clothes_path
        self.model_path = 'D:\Programming\CourseWork_3\code\data\model_new_all.pt'
        self.cats_path = "D:\Programming\CourseWork_3\code\data\categories.txt"
        self.types = {
            "1": "top",
            "2": "bottom",
            "3": "all",
            "4": "res"
        }
        self.stopped = False

    @pyqtSlot()
    def process(self):
        self.percent.emit(0)

        model = torchvision.models.resnext101_32x8d(pretrained=False)
        model.fc = torch.nn.Linear(in_features=2048, out_features=1024, bias=True)
        model = torch.nn.Sequential(
            model,
            torch.nn.Linear(in_features=1024, out_features=50, bias=True)
        ).cuda()
        model.load_state_dict(torch.load(self.model_path))
        model.cuda()
        model.eval()

        self.percent.emit(5)

        video = cv2.VideoCapture(self.path_video)
```

```
            sm = torch.nn.Softmax(dim=1)

            clothes_dict = dict()
            categories = self.getCategories()

            counter = 0
            frames_count = video.get(cv2.CAP_PROP_FRAME_COUNT)
            current_percent = 0

            tracks_file = open(self.path_tracks, "r")
            tracks = self.parsTracks(tracks_file.readlines())
            tracks_file.close()

            success, frame = video.read()
            while success and not self.stopped:
                cur_tracks = tracks[tracks[:, 0] == counter]

                for track in cur_tracks:
                    x1, x2 = max(0, track[2]), max(track[2] + track[4], frame.shape[1])
                    y1, y2 = max(0, track[3]), max(track[3] + track[5], frame.shape[0])

                    img = frame[y1:y2, x1:x2]
                    img = self.getProperImg(img)

                    img = torch.tensor(img, dtype=torch.float32,
device='cuda:0').reshape(1, 224, 224, 3).permute(0, 3, 1, 2).contiguous()
                    img = img / 256.0 - 0.5


                    probs = sm(model(img))
                    probs, idxs = probs.sort(descending=True)
                    probs = probs.detach().cpu().numpy()[0]
                    idxs = idxs.detach().cpu().numpy()[0]

                    track_id = str(track[1])
                    if track_id not in clothes_dict:
                        clothes_dict[track_id] = dict()
                        clothes_dict[track_id][str(-1)] = 0
                        for _, type in self.types.items():
                            clothes_dict[track_id][type] = Counter()

                    for i in range(6):
                        cat = categories[idxs[i]][0]
                        type = categories[idxs[i]][1]
                        clothes_dict[track_id][type][cat] += probs[i]
                    clothes_dict[track_id][str(-1)] += 1

                counter += 1
                if current_percent != int((counter / frames_count) * 95):
                    current_percent = int((counter / frames_count) * 95)
                self.percent.emit(current_percent + 5)

                success, frame = video.read()

            video.release()
```

```
            if self.stopped:
                os.remove(self.clothes_path)
            else:
                dict_out = dict()
                res = ["top", "bottom", "all"]
                for track, types in clothes_dict.items():
                    dict_out[track] = []
                    cur_types = dict(zip(res, list(zip([0] * len(res), [""] *
len(res)))))

                    for type in res:
                        for clothes, percent in types[type].items():
                            if percent > cur_types[type][0]:
                                cur_types[type] = [percent, clothes]
                    if cur_types["all"][0] * 2 > cur_types["bottom"][0] +
cur_types["top"][0]:
                        dict_out[track] = [cur_types["all"][1]]
                    else:
                        dict_out[track] = [cur_types["top"][1], cur_types["bottom"][1]]

                file_out = open(self.clothes_path, "w")
                file_out.write(json.dumps(dict_out))
                file_out.close()
                self.finished.emit()


        @staticmethod
        def getProperImg(img_arr):
            TARGET_SIZE = (224, 224)

            result = np.ones((TARGET_SIZE[0], TARGET_SIZE[1], 3), dtype=np.uint8) * 25
            try:
                img = Image.fromarray(img_arr)
            except:
                print(img.shape)
                print(img)
            img.thumbnail(TARGET_SIZE)
            offset = (np.array(TARGET_SIZE) - np.array(img.size)) // 2
            result[offset[1]: offset[1] + img.size[1], offset[0]: offset[0] +
img.size[0]] = np.array(img)
            return result

        @staticmethod
        def parsTracks(tracks):
            tracks_list = []
            for line in tracks:
                line = line.strip().split(",")
                line = list(map(int, map(float, line)))
                tracks_list.append(line)

            return np.array(tracks_list)

        def getCategories(self):
            cats_file = open(self.cats_path, 'r').readlines()
            categories = []
```

```
        for ln in cats_file:
            cur = list(filter(None, ln[:-1].split(' ')))
            categories.append([cur[0], self.types[cur[1]]])

        return categories

    def setStopped(self):
        self.stopped = True
```

8. TracksProcessor.py

```
from deep_sort_2.deep.feature_extractor import Extractor
from deep_sort_2.sort.nn_matching import NearestNeighborDistanceMetric
from deep_sort_2.sort.preprocessing import non_max_suppression
from deep_sort_2.sort.detection import Detection
from deep_sort_2.sort.tracker import Tracker

from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg

from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot

import numpy as np
import cv2
import os
import shutil
from pathlib import Path
import time


class SingleTrackerProcessor(QObject):
    finished = pyqtSignal()
    percent = pyqtSignal(int)

    min_confidence = 0.3
    nn_budget = 100
    max_cosine_distance = 0.2
    nms_max_overlap = 0.1
    n_init = 3
    max_iou_distance = 0.7
    max_age = 70

    def __init__(self, video_path, tracks_path):
        super().__init__()
        self.video_path = video_path
        self.tracks_path = tracks_path
        self.detector_name = "COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"
        self.tracker_path = \
"D:\Programming\CourseWork_3\code\deep_sort_2\deep\checkpoint\ckpt.t7"
        self.video = None
        self.stopped = False
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
    @pyqtSlot()
    def process(self):
        self.percent.emit(0)
        cfg = get_cfg()
        cfg.merge_from_file(model_zoo.get_config_file(self.detector_name))
        cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
        cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(self.detector_name)
        predictor = DefaultPredictor(cfg)
        metric = NearestNeighborDistanceMetric("cosine", self.max_cosine_distance,
self.nn_budget)
        tracker = Tracker(metric, max_iou_distance=self.max_iou_distance,
max_age=self.max_age, n_init=self.n_init)
        extractor = Extractor(self.tracker_path, use_cuda=True)

        self.percent.emit(5)

        out_file = open(self.tracks_path, 'w')
        self.video = cv2.VideoCapture(self.video_path)
        counter = 0
        frames_count = self.video.get(cv2.CAP_PROP_FRAME_COUNT)
        current_percent = 0

        det_time = 0
        trac_time = 0
        timeAll1 = time.time()

        success, frame = self.video.read()
        while success and not self.stopped:
            counter += 1
            if current_percent != int((counter / frames_count) * 95):
                current_percent = int((counter / frames_count) * 95)
            self.percent.emit(current_percent + 5)

            time1 = time.time()
            outputs = predictor(frame)
            time2 = time.time()
            preds = self.getBboxs(outputs["instances"].to("cpu"))

            features = self.get_features(preds[:, :4].astype(np.int32), frame,
extractor)
            bbox_tlwh = self.xyxy_to_xywh(preds[:, :4])
            detections = [Detection(bbox_tlwh[i], conf, features[i]) for i, conf in
enumerate(preds[:, 4]) if
                          conf > self.min_confidence]

            boxes = np.array([d.tlwh for d in detections])
            scores = np.array([d.confidence for d in detections])
            indices = non_max_suppression(boxes, self.nms_max_overlap, scores)
            detections = [detections[i] for i in indices]

            tracker.predict()
            tracker.update(detections)
            time3 = time.time()
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
                det_time += (time2 - time1)
                trac_time += (time3 - time2)

                for track in tracker.tracks:
                    if not track.is_confirmed() or track.time_since_update > 1:
                        continue

                    bbox = track.to_tlbr().astype(np.int32)

                    print('%d,%d,%.2f,%.2f,%.2f,%.2f,1,-1,-1,-1' % (
                        counter, track.track_id, bbox[0], bbox[1], bbox[2] - bbox[0],
bbox[3] - bbox[1]),
                          file=out_file)

                success, frame = self.video.read()


            timeAll2 = time.time()

            print(det_time, trac_time, det_time + trac_time)
            print(timeAll2 - timeAll1)
            out_file.close()
            self.video.release()
            if self.stopped:
                os.remove(self.tracks_path)
            else:
                self.finished.emit()

        @staticmethod
        def getBboxs(output):
            bboxs = output.pred_boxes[output.pred_classes == 0].tensor.numpy()
            scores = output.scores[output.pred_classes == 0].numpy()
            return np.concatenate((bboxs, scores.reshape(-1, 1)), axis=1)

        @staticmethod
        def get_features(bbox_xyxy, ori_img, extractor):
            im_crops = []
            for box in bbox_xyxy:
                x1, y1, x2, y2 = box
                im = ori_img[y1:y2, x1:x2]
                im_crops.append(im)
            if im_crops:
                features = extractor(im_crops)
            else:
                features = np.array([])
            return features

        @staticmethod
        def xyxy_to_xywh(bbox_xyxy):
            bbox_xywh = bbox_xyxy.copy()
            bbox_xywh[:, 2:] -= bbox_xywh[:, :2]
            return bbox_xywh

        def setStopped(self):
            self.stopped = True
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

9. VideoPlayerWidget.py

```
from PyQt5.QtGui import QImage, QPainter
from PyQt5 import QtWidgets, QtCore
from VideoPlayer import VideoPlayer
import time
import numpy as np


class VideoPlayerQWidget(QtWidgets.QWidget):
    STATUS_NOT_LOADED = -1
    STATUS_INIT = 0
    STATUS_PLAYING = 1
    STATUS_PAUSE = 2

    def __init__(self, parent=None):
        super().__init__(parent)
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.nextFrame)
        self.image = QImage()
        self.video = VideoPlayer()
        self.frame_counter = 0
        self.parent = parent
        self.status = self.STATUS_NOT_LOADED
        self.play_speed = 1.0
        self.updateRate = 24

    def get_qimage(self, image: np.ndarray):
        height, width, colors = image.shape
        bytesPerLine = 3 * width

        image = QImage(image.data,
                       width,
                       height,
                       bytesPerLine,
                       QImage.Format_RGB888)

        image = image.rgbSwapped()
        return image

    def getFrameCount(self):
        return self.video.getFrameCount()

    def getCurrentTime(self):
        return self.frame_counter // self.updateRate

    def getCurrentTime_frame(self):
        return self.frame_counter

    def setFrame(self, frame):
        self.timer.stop()
        self.timer.start(1000)
        self.frame_counter = frame
```

```
        self.video.setFrame(frame)
        self.parent.updateDurationInfo(frame // 24)
        if self.status == self.STATUS_PLAYING:
            self.timer.start(int((1000 // 24) / self.play_speed))
        elif self.status in (self.STATUS_PAUSE, self.STATUS_INIT):
            self.timer.stop()

    def setVideo(self, path):
        self.frame_counter = 0
        self.video.initialize(path)
        self.status = self.STATUS_INIT

    def play(self):
        if self.status == self.STATUS_INIT:
            self.video.play()
        self.timer.start(int((1000 // 24) / self.play_speed))
        self.status = self.STATUS_PLAYING

    def pause(self):
        self.timer.stop()
        self.status = self.STATUS_PAUSE

    def nextFrame(self):
        ret, frame = self.video.get_next_frame()
        if not ret:
            self.pause()
            self.parent.set_btnPlay()
        self.frame_counter += 1
        try:
            img = self.get_qimage(frame)
            self.image = img.scaled(self.size(), QtCore.Qt.KeepAspectRatio)
            self.update()
            if self.frame_counter % self.updateRate == 0:
                self.parent.set_slider_time(self.frame_counter // self.updateRate)
        except:
            pass

    def stopPlaying(self):
        if self.status != self.STATUS_NOT_LOADED:
            self.release()


    def paintEvent(self, event):
        painter = QPainter(self)
        x_pos = (self.width() - self.image.width()) // 2
        y_pos = (self.height() - self.image.height()) // 2
        painter.drawImage(x_pos, y_pos, self.image)

    def setPlaySpeed(self, speed):
        if self.play_speed != speed:
            self.play_speed = speed
            if self.status == self.STATUS_PLAYING:
                self.timer.start(int((1000 // 24) / self.play_speed))

    def setUpdateRate(self, rate):
```

```
        self.updateRate = rate


    def release(self):
        self.video.release()
```

10. VideoPlayer.py

```
import numpy as np
import cv2
from VideoStream import FileVideoStream


class VideoPlayer:
    def __init__(self):
        self.path = ""
        self.stream = None
        self.started = False
        self.last_frame = np.zeros((1,1,3))

    def initialize(self, path):
        self.stream = FileVideoStream(path, 30)

    def play(self):
        self.stream.start()
        self.started = True

    def get_next_frame(self):
        if self.stream.more():
            self.last_frame = self.stream.read()
            return True, self.last_frame
        self.stream.pause()
        return False, 0

    def getFrameCount(self):
        return self.stream.stream.get(cv2.CAP_PROP_FRAME_COUNT)

    def setFrame(self, frame):
        self.stream.setFrame(frame)

    def release(self):
        if self.started:
            self.stream.stop()
```

11. VideoStream.py

```
import threading
import cv2
import time
from queue import Queue
from Visualizer import Visualizer


class FileVideoStream:
    def __init__(self, path, queue_size=128):
        self.stream = cv2.VideoCapture(path)
        self.stopped = False
        self.queue_size = queue_size
        self.path = path
        self.Q = Queue(maxsize=queue_size)
        self.thread = threading.Thread(target=self.update, args=())
        # self.thread.daemon = True
        self.allreaded = False
        self.pause_cond = threading.Condition(threading.Lock())
        self.paused = False
        self.Visualizer = Visualizer.getInstance()
        self.Visualizer.setStream(self)
        self.frame_id = 0

    def start(self):
        self.thread.start()
        return self

    def update(self):
        while True:
            with self.pause_cond:
                while self.paused:
                    self.pause_cond.wait()

                if self.stopped:
                    break
                if self.allreaded:
                    continue
                if not self.Q.full():
                    (grabbed, frame) = self.stream.read()
                    self.frame_id += 1
                    if not grabbed:
                        self.allreaded = True

                    frame = self.Visualizer.drowTracks(frame, self.frame_id + 1)

                    self.Q.put(frame)
                else:
                    time.sleep(0.2)

        self.stream.release()

    def read(self):
        return self.Q.get()
```

```
def running(self):
    return self.more() or not self.stopped

def more(self):
    tries = 0
    while self.Q.qsize() == 0 and not self.stopped and tries < 5:
        time.sleep(0.1)
        tries += 1

    return self.Q.qsize() > 0

def flushQueue(self):
    self.Q = Queue(maxsize=self.queue_size)

def setFrame(self, frame):
    self.pause()
    self.allreaded = False
    self.stream.set(cv2.CAP_PROP_POS_FRAMES, frame)
    self.flushQueue()
    self.frame_id = frame
    self.resume()


def resetFrames(self):
    self.frame_id -= self.Q.qsize()
    self.stream.set(cv2.CAP_PROP_POS_FRAMES, self.frame_id)
    self.flushQueue()

def stop(self):
    self.stopped = True
    if self.thread.is_alive():
        self.thread.join()

def pause(self):
    if self.paused:
        return
    self.paused = True
    self.pause_cond.acquire()

def resume(self):
    self.paused = False
    self.pause_cond.notify()
    self.pause_cond.release()
```

12. Visualizer.py

```
import numpy as np
import cv2


class Visualizer:
    __instance = None
    stream = None
    tracks_info = None
    tracks = None
    colours = np.random.randint(255, size=(64, 3))

    def __init__(self):
        if Visualizer.__instance is None:
            Visualizer.__instance = self
        else:
            raise Exception("This class is a singleton!")

    @staticmethod
    def getInstance():
        if Visualizer.__instance is None:
            Visualizer()
        return Visualizer.__instance

    def setStream(self, stream):
        self.stream = stream

    def setTracksInfo(self, tracks_info):
        self.tracks_info = tracks_info

    def setTracks(self, tracks):
        if self.stream:
            self.stream.pause()
            self.stream.resetFrames()
            self.tracks = set(tracks)
            self.stream.resume()

    def drowTracks(self, img, frame):
        if self.tracks is not None and self.tracks_info is not None:
            frame_tracks = self.tracks_info[self.tracks_info[:, 0] == frame]
            for track in frame_tracks:
                if track[1] in self.tracks:
                    colour = self.colours[track[1] % 64, :].tolist()
                    cv2.rectangle(img, (track[2], track[3]), (track[2] + track[4],
track[3] + track[5]), colour, 2)
                    cv2.putText(img, str(track[1]), (track[2], track[3]), 0, 5e-3 *
200, (0, 255, 0), 1)

        return img
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

13. Category_prediction.ipynb

```
#%%

import torchvision
import torch
import numpy as np
import os

BASE_PATH = '/content/Clothes/'

#%%

from google.colab import drive
drive.mount('/content/drive')

#%%

import shutil
shutil.copytree('/content/drive/My Drive/Clothes/', '/content/Clothes/')

#%%

import zipfile
with zipfile.ZipFile('/content/Clothes/img.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/Clothes/')

#%% md

# Dataset loading

#%%

cats_file = open(BASE_PATH + 'Anno/list_category_cloth.txt', 'r').readlines()
cats_img_file = open(BASE_PATH + 'Anno/list_category_img.txt', 'r').readlines()

#%%

path_to_idx = {}
idx_to_path = {}

cats_img_file = [list(filter(None, x.strip().split(' '))) for x in cats_img_file[2:]]

for i, img in enumerate(cats_img_file):
    path = img[0]
    idx_to_path[i] = path
    path_to_idx[path] = i

N_SAMPLES = len(idx_to_path)

#%%

categories = []

for ln in cats_file[2:]:
```

```
    cur = list(filter(None, ln[:-1].split(' ')))
    categories.append(cur[0])


cat_target = np.zeros((len(idx_to_path), len(categories)), dtype=np.uint8)
cat_list = {}

for img in cats_img_file:
    cur_cat = int(img[1]) - 1

    cat_target[path_to_idx[img[0]]][cur_cat] = 1
    if cur_cat in cat_list:
        cat_list[cur_cat].append(path_to_idx[img[0]])
    else:
        cat_list[cur_cat] = [path_to_idx[img[0]]]

#%%

train_idxs = []
val_idxs   = []

split_file = open(BASE_PATH + 'Eval/list_eval_partition.txt', 'r').readlines()

for ln in split_file[2:]:
    cur = list(filter(None, ln[:-1].split(' ')))
    cur_type = cur[1]
    if cur_type == 'train':
        train_idxs.append(path_to_idx[cur[0]])
    elif cur_type == 'val':
        val_idxs.append(path_to_idx[cur[0]])

train_idxs = np.array(train_idxs)
val_idxs = np.array(val_idxs)

#%% md

# Data loading

#%%

from skimage.transform import rescale
from PIL import Image

TARGET_SIZE = (224, 224)

def load_img(path):
    result = np.ones((TARGET_SIZE[0], TARGET_SIZE[1], 3), dtype=np.uint8) * 255
    img = Image.open(path)
    img.thumbnail(TARGET_SIZE)
    offset = (np.array(TARGET_SIZE) - np.array(img.size)) // 2
    result[offset[1]: offset[1] + img.size[1], offset[0]: offset[0] + img.size[0]] =
np.array(img)
    return result
```

```
#%%

def batch_generator(batch_size=32, dataset='train', shuffle=True):
    if dataset == 'train':
        idxs = train_idxs
    else:
        idxs = val_idxs

    perm = np.random.permutation(idxs) if shuffle else np.arange(len(idxs))
    start_idx = 0
    while True:
        if start_idx + batch_size > len(idxs):
            perm = np.random.permutation(idxs) if shuffle else np.arange(len(idxs))
            start_idx = 0
        imgs = [load_img(BASE_PATH + idx_to_path[perm[start_idx + i]]) for i in
range(batch_size)]
        imgs = np.stack(imgs) / 256.0 - 0.5
        imgs = torch.tensor(imgs, dtype=torch.float32, device='cuda:0').permute(0, 3,
1, 2).contiguous()

        cur_idxs = perm[start_idx: start_idx + batch_size]
        target = torch.tensor(cat_target[cur_idxs].argmax(1), dtype=torch.int64,
device='cuda:0').contiguous()
        start_idx += batch_size
        yield imgs, target

#%% md

# Architecture

#%%

model = torchvision.models.resnext101_32x8d(pretrained=True)
model.fc = torch.nn.Linear(in_features=2048, out_features=1024, bias=True)
model = torch.nn.Sequential(
    model,
    torch.nn.Linear(in_features=1024, out_features=50, bias=True)
).cuda()

#%%

for j, ch in enumerate(model.children()):
    if j == 0:
        for i, layer in enumerate(ch.children()):
            if i < 9:
                for param in layer.parameters():
                    param.requires_grad = False
            if i == 9:
                for param in layer.parameters():
                    param.requires_grad = True

    else:
        for param in ch.parameters():
            param.requires_grad = True
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

```
#%%

from torchsummary import summary
summary(model, (3, 224, 224))

#%%

import time
import copy

def train_model(model, criterion, optimizer, num_epochs=3, batch_size=12):
    since = time.time()

    history = {}

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    dataloaders = {
        'train' : batch_generator(batch_size=batch_size, dataset='train'),
        'val' : batch_generator(batch_size=batch_size, dataset='val')
    }

    for epoch in range(num_epochs):
        history[epoch] = {}
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0
            running_top_3 = 0.0
            running_top_5 = 0.0

            # Iterate over data.
            N_STEPS = len(train_idxs if phase == 'train' else val_idxs) // batch_size
            N_STEPS = N_STEPS // 4
            total_imgs = 0
            s = time.time()
            for step in range(N_STEPS):
                inputs, labels = next(dataloaders[phase])
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                    _, preds = torch.max(outputs, 1)

                    pred_order = torch.argsort(outputs, dim=1, descending=True)
```

```
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                cur_corrects = torch.sum(preds == labels.data)
                running_corrects += cur_corrects

                cur_top_3 = (pred_order[:, :3] == labels.view(-1, 1)).sum()
                cur_top_5 = (pred_order[:, :5] == labels.view(-1, 1)).sum()
                running_top_3 += cur_top_3
                running_top_5 += cur_top_5

                total_imgs += batch_size
                t = time.time()
                print('Step: %d/%d (%.2f ms per step). Loss: %.4f. Batch-accuracy:
%.2f%%. Batch-top-3: %.2f%%. Batch-top-5: %.2f%% ' % (
                    step, N_STEPS, (t-s)*1000.0 / (step + 1),
                    running_loss / total_imgs,
                    cur_corrects * 100.0 / batch_size,
                    cur_top_3 * 100.0 / batch_size,
                    cur_top_5 * 100.0 / batch_size
                ))

            cur_idxs = train_idxs if phase == 'train' else val_idxs
            epoch_loss = running_loss / total_imgs
            epoch_acc = running_corrects.double() / total_imgs
            epoch_top3 = running_top_3.double() / total_imgs
            epoch_top5 = running_top_5.double() / total_imgs

            history[epoch][phase] = {
                'loss' : epoch_loss,
                'accuracy' : epoch_acc,
                'top3-acc' : epoch_top3,
                'top5-acc' : epoch_top5
            }

            print()
            print('Epoch %d| %s| Loss: %.4f. Acc: %.4f' % (epoch, phase, epoch_loss,
epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

            torch.save(model,
f'D:\Programming\CourseWork_3\Clothes\\model{mod}_{epoch}.pth')
        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60,
time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))
```

```
    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, history

#%%

loss = torch.nn.CrossEntropyLoss()

params_to_update = []
for name,param in model.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)

optimizer_ft = torch.optim.SGD(params_to_update, lr=0.001, momentum=0.9)

result = train_model(model, loss, optimizer_ft, batch_size=32, num_epochs=4)

#%%

model = result[0]
torch.save(model.state_dict(), f"/content/drive/My Drive/Clothes/model_last.pt")

#%%

import tqdm
gen = batch_generator(batch_size=32, dataset='val')

N_STEPS = 1250

accuracy = 0.0
top3_acc = 0.0
top5_acc = 0.0

model.eval()
with torch.set_grad_enabled(False):
    for i in tqdm.tqdm(range(N_STEPS)):
        imgs, labels = next(gen)
        outs = model(imgs)
        pred_order = torch.argsort(outs, dim=1, descending=True)

        accuracy += (pred_order[:, 0] == labels).sum()
        top3_acc += (pred_order[:, :3] == labels.view(-1, 1)).sum()
        top5_acc += (pred_order[:, :5] == labels.view(-1, 1)).sum()

#%%

total_imgs = N_STEPS * 32
print('Accuracy: %.3f%%. Top3-accuracy: %.3f%%. Top5-accuracy: %.3f%% ' % (
    float(accuracy) * 100.0 / total_imgs,
    float(top3_acc) * 100.0 / total_imgs,
    float(top5_acc) * 100.0 / total_imgs),
)

#%% md
```

```
### Fine tuning of convolutional layers

#%%

torch.cuda.empty_cache()

#%%

model = torchvision.models.resnext101_32x8d(pretrained=False)
model.fc = torch.nn.Linear(in_features=2048, out_features=1024, bias=True)
model = torch.nn.Sequential(
    model,
    torch.nn.Linear(in_features=1024, out_features=50, bias=True)
).cuda()
model.load_state_dict(torch.load("/content/drive/My Drive/Clothes/model_last.pt"))

#%%

for param in model.parameters():
    param.requires_grad = True

#%%

from torchsummary import summary
summary(model, (3, 224, 224))

#%%

# torch.backends.cudnn.benchmark = True
# torch.backends.cudnn.deterministic = True
loss = torch.nn.CrossEntropyLoss()

params_to_update = []
for name,param in model.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)

optimizer_ft = torch.optim.SGD(params_to_update, lr=0.001, momentum=0.9)

result = train_model(model, loss, optimizer_ft, batch_size=32, num_epochs=4)

#%%

model = result[0]
torch.save(model.state_dict(), f"/content/drive/My Drive/Clothes/model_all.pt")

#%%

import tqdm
gen = batch_generator(batch_size=32, dataset='val', shuffle=False)

N_STEPS = 1250

accuracy = 0.0
```

```
top3_acc = 0.0
top5_acc = 0.0

model.eval()
preds_lst = []
labels_lst = []
with torch.set_grad_enabled(False):
    for i in tqdm.tqdm(range(N_STEPS)):
        imgs, labels = next(gen)
        labels_lst.append(labels.detach().cpu().numpy())
        outs = model(imgs)
        preds_lst.append(outs.argmax(dim=1).detach().cpu().numpy())
        pred_order = torch.argsort(outs, dim=1, descending=True)

        accuracy += (pred_order[:, 0] == labels).sum()
        top3_acc += (pred_order[:, :3] == labels.view(-1, 1)).sum()
        top5_acc += (pred_order[:, :5] == labels.view(-1, 1)).sum()

#%%

torch.save(model.state_dict(), 'deepfashion_cat_ep8.torch')

#%%

total_imgs = N_STEPS * 32
print('Accuracy: %.3f%%. Top3-accuracy: %.3f%%. Top5-accuracy: %.3f%% ' % (
    float(accuracy) * 100.0 / total_imgs,
    float(top3_acc) * 100.0 / total_imgs,
    float(top5_acc) * 100.0 / total_imgs)
)

#%% md

## Visual evaluation

#%%

torch.cuda.empty_cache()

#%%

model = torchvision.models.resnext101_32x8d(pretrained=False)
model.fc = torch.nn.Linear(in_features=2048, out_features=1024, bias=True)
model = torch.nn.Sequential(
    model,
    torch.nn.Linear(in_features=1024, out_features=50, bias=True)
).cuda()
model.load_state_dict(torch.load("/content/drive/My Drive/Clothes/model_all.pt"))

#%%

img = load_img('tmp.jpg')
plt.imshow(img)
img = torch.tensor(img, dtype=torch.float32, device='cuda:0').reshape(1, 224, 224,
3).permute(0, 3, 1, 2)
```

```
img = img / 256.0 - 0.5

sm = torch.nn.Softmax(dim=1)
probs = sm(model(img))
probs, idxs = probs.sort(descending=True)
probs = probs.detach().cpu().numpy()[0]
idxs = idxs.detach().cpu().numpy()[0]

for i in range(5):
    print('%d) %s %.2f%% ' % (i + 1, categories[idxs[i]], probs[i] * 100.0))
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.13-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата