Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

\_\_\_\_\_\_\_\_ Євгеній ВОВК

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 р.

**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РОЗПІЗНАВАНЬ ОБЛИЧ В ВІДЕОПОТОЦІ**

**Текст програми**

КПІ.ІT-0324.045440.03.12

“ПОГОДЖЕНО”

Керівник роботи:

\_\_\_\_\_\_\_\_\_\_\_\_ Євгеній ВОВК

|  |  |
| --- | --- |
| Консультант: | Виконавець: |
| \_\_\_\_\_\_\_\_\_\_ Максим ГОЛОВЧЕНКО | \_\_\_\_\_\_\_\_\_\_ Марина ЦУКАНОВА |

Київ – 2022

**Файл main.py**

**import** tkinter  
**from** tkinter **import** ttk, messagebox  
**from** tkinter **import** \*  
  
**import** cv2  
**import** mysql.connector  
  
**from** yt\_manager **import** YoutubeManager  
**from** db\_manager **import** DBManager  
**from** video\_analysis **import** VideoAnalysis  
**from** table\_manager **import** TableManager  
**from** PIL **import** Image, ImageTk  
  
  
**class** WindowsManager:  
  
 **def** \_\_init\_\_(self, root, frame):  
 self.root = root  
 self.frame = frame  
  
 **def** youtube\_window(self, root, frame):  
 frame.grid\_forget()  
 youtube\_window\_frame = Frame(root)  
 youtube\_window\_frame.grid(row=0, column=0)  
  
 yt\_label = Label(youtube\_window\_frame, text=**'Enter Youtube link'**, font=(**"Arial"**, 12))  
 yt\_label.grid(row=0, column=0)  
 yt\_entry = Entry(youtube\_window\_frame, width=105, font=(**"Arial"**, 12))  
 yt\_entry.grid(row=1, column=0, padx=10, pady=10)  
 analyse\_button = Button(youtube\_window\_frame, text=**"Analyse video"**,  
 command=**lambda**: yt\_man.parse\_yt\_data(yt\_entry), font=(**"Arial"**, 12))  
 analyse\_button.grid(row=2, column=0, padx=10, pady=10)  
  
 done\_button = Button(youtube\_window\_frame, text=**"To my videos"**,  
 command=**lambda**: self.my\_videos(root, youtube\_window\_frame), font=(**"Arial"**, 12))  
 done\_button.grid(row=3, column=0, padx=10, pady=10)  
  
 **return** youtube\_window\_frame  
  
 **def** my\_videos(self, root, frame):  
 frame.grid\_forget()  
 my\_videos\_frame = Frame(root)  
 my\_videos\_frame.grid(row=0, column=0)  
  
 style = ttk.Style()  
 style.configure(**"Treeview"**, background=**"#D3D3D3"**, foreground=**"black"**,  
 rowheight=25, fieldbackground=**"#D3D3D3"**)  
 tree\_frame = Frame(my\_videos\_frame)  
 tree\_frame.pack(pady=25, padx=25)  
  
 tree\_scroll = Scrollbar(tree\_frame)  
 tree\_scroll.pack(side=RIGHT, fill=Y)  
res\_tree = ttk.Treeview(tree\_frame, yscrollcommand=tree\_scroll.set, selectmode=**"browse"**)  
 res\_tree.pack()  
 tree\_scroll.config(command=res\_tree.yview)  
  
 res\_tree[**'columns'**] = (**"ID"**, **"Video Name"**, **"Channel Name"**, **'Youtube Link'**)  
res\_tree.column(**"#0"**, width=0, stretch=NO)  
 res\_tree.column(**"ID"**, anchor=CENTER, width=50)  
 res\_tree.column(**"Video Name"**, anchor=CENTER, width=300)  
 res\_tree.column(**"Channel Name"**, anchor=CENTER, width=250)  
 res\_tree.column(**"Youtube Link"**, anchor=CENTER, width=350)  
res\_tree.heading(**"#0"**, text=**""**, anchor=W)  
 res\_tree.heading(**"ID"**, text=**"ID"**, anchor=CENTER)  
 res\_tree.heading(**"Video Name"**, text=**"Video Name"**, anchor=CENTER)  
 res\_tree.heading(**"Channel Name"**, text=**"Channel Name"**, anchor=CENTER)  
 res\_tree.heading(**"Youtube Link"**, text=**"Youtube Link"**, anchor=CENTER)  
  
 **def** select\_record(self):  
 **try**:  
 selected = res\_tree.focus()  
 values = res\_tree.item(selected, **'values'**)  
 watch\_button[**'state'**] = tkinter.NORMAL  
 delete\_button[**'state'**] = tkinter.NORMAL  
 gen\_res\_button[**'state'**] = tkinter.NORMAL  
 det\_res\_button[**'state'**] = tkinter.NORMAL  
 analyse\_ag\_btn[**'state'**] = tkinter.NORMAL  
 **return** values[0]  
  
 **except** IndexError:  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
 **except** mysql.connector.Error:  
 tb\_man.deselect(res\_tree)  
 watch\_button[**'state'**] = tkinter.DISABLED  
 delete\_button[**'state'**] = tkinter.DISABLED  
 gen\_res\_button[**'state'**] = tkinter.DISABLED  
 det\_res\_button[**'state'**] = tkinter.DISABLED  
 analyse\_ag\_btn[**'state'**] = tkinter.DISABLED  
 messagebox.showerror(**'showerror'**, **'DatabaseError'**)  
  
 button\_frame = LabelFrame(my\_videos\_frame, text=**"Commands"**, font=(**"Arial"**, 12))  
 button\_frame.pack(fill=**"x"**, expand=**"yes"**, padx=25, anchor=CENTER)  
  
 watch\_button = Button(button\_frame, text=**'Watch video'**,  
 command=**lambda**: tb\_man.see\_video\_result(res\_tree, select\_record(my\_videos\_frame)),  
 state=tkinter.DISABLED, font=(**"Arial"**, 12))  
 watch\_button.grid(row=0, column=0, padx=10, pady=10)  
  
 delete\_button = Button(button\_frame, text=**'Delete video'**,  
 command=**lambda**: tb\_man.remove\_one(res\_tree, select\_record(  
 my\_videos\_frame)), state=tkinter.DISABLED, font=(**"Arial"**, 12))  
 delete\_button.grid(row=0, column=1, padx=10, pady=10)  
  
 gen\_res\_button = Button(button\_frame, text=**'To general results'**,  
 command=**lambda**: self.general\_res\_page(root, my\_videos\_frame,  
 video\_id=select\_record(my\_videos\_frame)),  
 state=tkinter.DISABLED, font=(**"Arial"**, 12))  
 gen\_res\_button.grid(row=0, column=2, padx=10, pady=10)  
  
 det\_res\_button = Button(button\_frame, text=**'To detailed results'**,  
 command=**lambda**: self.detailed\_res\_page(root, my\_videos\_frame,  
 video\_id=select\_record(my\_videos\_frame)),  
 state=tkinter.DISABLED, font=(**"Arial"**, 12))  
 det\_res\_button.grid(row=0, column=3, padx=10, pady=10)  
  
 to\_main\_button = Button(button\_frame, text=**'To analysis page'**,  
 command=**lambda**: self.youtube\_window(root, my\_videos\_frame), font=(**"Arial"**, 12))  
 to\_main\_button.grid(row=0, column=4, padx=10, pady=10)  
  
 analyse\_ag\_btn = Button(button\_frame, text=**'Analyse again'**,  
 command=**lambda**: vid\_analysis.repeat\_analysis(video\_id=select\_record(my\_videos\_frame)),  
 state=tkinter.DISABLED, font=(**"Arial"**, 12))  
 analyse\_ag\_btn.grid(row=0, column=5)  
  
 res\_tree.bind(**"<ButtonRelease-1>"**, select\_record)  
 tb\_man.query\_videos\_database(res\_tree)  
 root.mainloop()  
  
 **def** general\_res\_page(self, root, frame, video\_id):  
 frame.grid\_forget()  
 general\_res\_frame = Frame(root)  
 general\_res\_frame.grid(row=0, column=0)  
 general\_label = Label(general\_res\_frame, text=**'General results of video №'** + str(video\_id), font=(**"Arial"**, 12))  
 general\_label.grid(row=0, column=1)  
 to\_main\_button = Button(general\_res\_frame, text=**'To my videos'**,  
 command=**lambda**: self.my\_videos(root, general\_res\_frame),  
 font=(**"Arial"**, 12))  
 to\_main\_button.grid(row=1, column=1)  
  
 data = db\_man.get\_gen\_res\_data(video\_id)  
 male\_count = data[0][2]  
 female\_count = data[0][3]  
 gender\_img = data[0][11]  
  
 angry\_count = data[0][4]  
 scared\_count = data[0][5]  
 happy\_count = data[0][6]  
 sad\_count = data[0][7]  
 surprised\_count = data[0][8]  
 neutral\_count = data[0][9]  
 emotion\_img = data[0][14]  
  
 avg\_age = data[0][10]  
 age\_count\_img = data[0][12]  
 age\_box\_img = data[0][13]  
  
 image\_folder = **'C:/Users/serg/PycharmProjects/cpe\_desktop\_coursework/stats\_folder/'** gender\_frame = LabelFrame(general\_res\_frame, text=**'Gender stats'**, font=(**"Arial"**, 12))  
 male\_label = Label(gender\_frame, text=**'Male had '** + str(male\_count) + **' occurrences.'**, font=(**"Arial"**, 12))  
 male\_label.grid(row=0, column=0)  
 female\_label = Label(gender\_frame, text=**'Female had '** + str(female\_count) + **' occurrences.'**, font=(**"Arial"**, 12))  
 female\_label.grid(row=1, column=0)  
 g\_img = Image.open(image\_folder + gender\_img)  
 g\_img = g\_img.resize((300, 300))  
 g\_img = ImageTk.PhotoImage(g\_img)  
 gender\_img = Label(gender\_frame, image=g\_img)  
 gender\_img.grid(row=2, column=0)  
  
 age\_frame = LabelFrame(general\_res\_frame, text=**'Age stats'**, font=(**"Arial"**, 12))  
 age\_label = Label(age\_frame, text=**'Average age is '** + str(avg\_age), font=(**"Arial"**, 12))  
 age\_label.grid(row=0, column=0, sticky=**'ew'**)  
 a\_b\_img = Image.open(image\_folder + age\_box\_img)  
 a\_b\_img = a\_b\_img.resize((300, 300))  
 a\_b\_img = ImageTk.PhotoImage(a\_b\_img)  
 age\_box\_img = Label(age\_frame, image=a\_b\_img)  
 age\_box\_img.grid(row=1, column=0)  
 a\_c\_img = Image.open(image\_folder + age\_count\_img)  
 a\_c\_img = a\_c\_img.resize((300, 300))  
 a\_c\_img = ImageTk.PhotoImage(a\_c\_img)  
 age\_count\_img = Label(age\_frame, image=a\_c\_img)  
 age\_count\_img.grid(row=1, column=1)  
  
 emotion\_frame = LabelFrame(general\_res\_frame, text=**'Emotion stats'**, font=(**"Arial"**, 12))  
 angry\_label = Label(emotion\_frame, text=**'Angry people had '** + str(angry\_count) + **' occurrences.'**,  
 font=(**"Arial"**, 12))  
 angry\_label.grid(row=0, column=0)  
 scared\_label = Label(emotion\_frame, text=**'Scared people had '** + str(scared\_count) + **' occurrences.'**,  
 font=(**"Arial"**, 12))  
 scared\_label.grid(row=1, column=0)  
 happy\_label = Label(emotion\_frame, text=**'Happy people had '** + str(happy\_count) + **' occurrences.'**,  
 font=(**"Arial"**, 12))  
 happy\_label.grid(row=2, column=0)  
 sad\_label = Label(emotion\_frame, text=**'Sad people had '** + str(sad\_count) + **' occurrences.'**, font=(**"Arial"**, 12))  
 sad\_label.grid(row=0, column=1)  
 surprised\_label = Label(emotion\_frame, text=**'Surprised people had '** + str(surprised\_count) + **' occurrences.'**,  
 font=(**"Arial"**, 12))  
 surprised\_label.grid(row=1, column=1)  
 neutral\_label = Label(emotion\_frame, text=**'Neutral people had '** + str(neutral\_count) + **' occurrences.'**,  
 font=(**"Arial"**, 12))  
 neutral\_label.grid(row=2, column=1)  
 em\_img = Image.open(image\_folder + emotion\_img)  
 em\_img = em\_img.resize((300, 300))  
 em\_img = ImageTk.PhotoImage(em\_img)  
 emotion\_img = Label(emotion\_frame, image=em\_img)  
 emotion\_img.grid(row=3, column=0)  
  
 **def** show\_frame(gender, age, emotion):  
 **if** gender == 1:  
 age\_frame.grid\_forget()  
 emotion\_frame.grid\_forget()  
 gender\_frame.grid(row=3, column=1)  
 **elif** age == 1:  
 emotion\_frame.grid\_forget()  
 gender\_frame.grid\_forget()  
 age\_frame.grid(row=3, column=1)  
 **else**:  
 age\_frame.grid\_forget()  
 gender\_frame.grid\_forget()  
 emotion\_frame.grid(row=3, column=1)  
  
 gender\_button = Button(general\_res\_frame, text=**'Show gender stats'**,  
 command=**lambda**: show\_frame(1, 0, 0), font=(**"Arial"**, 12))  
 gender\_button.grid(row=2, column=0)  
 age\_button = Button(general\_res\_frame, text=**'Show age stats'**, command=**lambda**: show\_frame(0, 1, 0),  
 font=(**"Arial"**, 12))  
 age\_button.grid(row=2, column=1)  
 emotion\_button = Button(general\_res\_frame, text=**'Show emotion stats'**, command=**lambda**: show\_frame(0, 0, 1),  
 font=(**"Arial"**, 12))  
 emotion\_button.grid(row=2, column=2)  
  
 detailed\_button = Button(general\_res\_frame, text=**'To detailed results'**,  
 command=**lambda**: self.detailed\_res\_page(root, general\_res\_frame, video\_id),  
 font=(**"Arial"**, 12))  
 detailed\_button.grid(row=4, column=1)  
  
 root.mainloop()  
  
 **def** detailed\_res\_page(self, root, frame, video\_id):  
 frame.grid\_forget()  
 detailed\_res\_frame = Frame(root)  
 detailed\_res\_frame.grid(row=0, column=0)  
  
 style = ttk.Style()  
 style.configure(**"Treeview"**, background=**"#D3D3D3"**, foreground=**"black"**,  
 rowheight=25, fieldbackground=**"#D3D3D3"**)  
 res\_tree\_frame = Frame(detailed\_res\_frame)  
 res\_tree\_frame.pack(pady=25, padx=50)  
  
 res\_tree\_scroll = Scrollbar(res\_tree\_frame)  
 res\_tree\_scroll.pack(side=RIGHT, fill=Y)  
det\_res\_tree = ttk.Treeview(res\_tree\_frame, yscrollcommand=res\_tree\_scroll.set, selectmode=**"browse"**)  
 det\_res\_tree.pack()  
 res\_tree\_scroll.config(command=det\_res\_tree.yview)  
  
 det\_res\_tree[**'columns'**] = (**"ID"**, **"Video ID"**, **"Gender"**, **'Age'**, **'Emotion'**, **'Photo path'**)  
det\_res\_tree.column(**"#0"**, width=0, stretch=NO)  
 det\_res\_tree.column(**"ID"**, anchor=CENTER, width=70)  
 det\_res\_tree.column(**"Video ID"**, anchor=CENTER, width=70)  
 det\_res\_tree.column(**"Gender"**, anchor=CENTER, width=100)  
 det\_res\_tree.column(**"Age"**, anchor=CENTER, width=100)  
 det\_res\_tree.column(**"Emotion"**, anchor=CENTER, width=100)  
 det\_res\_tree.column(**"Photo path"**, anchor=CENTER, width=300)  
det\_res\_tree.heading(**"#0"**, text=**""**, anchor=W)  
 det\_res\_tree.heading(**"ID"**, text=**"ID"**, anchor=CENTER)  
 det\_res\_tree.heading(**"Video ID"**, text=**"Video ID"**, anchor=CENTER)  
 det\_res\_tree.heading(**"Gender"**, text=**"Gender"**, anchor=CENTER)  
 det\_res\_tree.heading(**"Age"**, text=**"Age"**, anchor=CENTER)  
 det\_res\_tree.heading(**"Emotion"**, text=**"Emotion"**, anchor=CENTER)  
 det\_res\_tree.heading(**"Photo path"**, text=**"Photo path"**, anchor=CENTER)  
  
 **def** select\_record(self):  
 **try**:  
 selected = det\_res\_tree.focus()  
 values = det\_res\_tree.item(selected, **'values'**)  
 photo\_button[**'state'**] = tkinter.NORMAL  
 **return** values[0]  
 **except** IndexError:  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
 **except** mysql.connector.Error:  
 tb\_man.deselect(det\_res\_tree)  
 photo\_button[**'state'**] = tkinter.DISABLED  
 messagebox.showerror(**'showerror'**, **'DatabaseError'**)  
  
 button\_frame = LabelFrame(detailed\_res\_frame, text=**"Commands"**, font=(**"Arial"**, 12))  
 button\_frame.pack(fill=**"x"**, expand=**"yes"**, padx=20)  
  
 photo\_button = Button(button\_frame, text=**'View photo'**, command=**lambda**: tb\_man.view\_photo(det\_res\_tree),  
 state=tkinter.DISABLED, font=(**"Arial"**, 12))  
 photo\_button.grid(row=0, column=0, padx=10, pady=10)  
  
 gen\_res\_button = Button(button\_frame, text=**'To general results'**,  
 command=**lambda**: self.general\_res\_page(root, detailed\_res\_frame, video\_id=video\_id),  
 font=(**"Arial"**, 12))  
 gen\_res\_button.grid(row=0, column=2, padx=10, pady=10)  
  
 my\_videos\_btn = Button(button\_frame, text=**'To my videos'**,  
 command=**lambda**: self.my\_videos(root, detailed\_res\_frame), font=(**"Arial"**, 12))  
 my\_videos\_btn.grid(row=0, column=3, padx=10, pady=10)  
  
 det\_res\_tree.bind(**"<ButtonRelease-1>"**, select\_record)  
 tb\_man.query\_results\_database(det\_res\_tree, video\_id)  
 root.mainloop()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 db\_man = DBManager()  
 vid\_analysis = VideoAnalysis()  
 yt\_man = YoutubeManager()  
 tb\_man = TableManager()  
  
 root = Tk()  
 root.title(**'Video analyser'**)  
 root.geometry(**"1000x550"**)  
 frame = Frame(root)  
 win = WindowsManager(root, frame)  
 win.my\_videos(root, frame)  
 root.mainloop()

**Файл yt\_manager.py**

**import** tkinter  
  
**from** pytube **import** YouTube  
**import** requests  
**import** json  
**from** urllib.parse **import** urlparse, parse\_qs  
**from** db\_manager **import** DBManager  
**from** video\_analysis **import** VideoAnalysis  
**from** tkinter **import** messagebox  
**import** os  
  
db\_man = DBManager()  
vid\_analysis = VideoAnalysis()  
  
  
**class** YoutubeManager:  
 **def** \_\_init\_\_(self):  
 self.folder\_path = **'/downloaded\_videos'** self.api\_key = **'AIzaSyDLtCySc1gSpyguuzcl16ij-4x7QwJp0R4'  
  
 def** parse\_yt\_data(self, yt\_entry):  
 **try**:  
 video\_link = yt\_entry.get()  
 **if** video\_link == **''**:  
 messagebox.showerror(**'showerror'**, **'Enter Youtube link'**)  
 **else**:  
 video\_yt\_id = self.get\_video\_id(video\_link)  
 **if** video\_yt\_id **is None**:  
 messagebox.showerror(**'showerror'**, **'Incorrect link. Try again!'**)  
 yt\_entry.delete(0, tkinter.END)  
 **else**:  
 video\_id = db\_man.check\_video\_ex(video\_yt\_id)  
 **if** video\_id **is False**:  
 video\_path = self.download\_video(video\_link)  
 video\_name, channel\_id = self.get\_video\_data(video\_yt\_id, self.api\_key)  
 channel\_name = self.get\_channel\_stats(channel\_id, self.api\_key)  
 print(video\_path, video\_yt\_id, video\_name, channel\_name, channel\_id)  
 db\_man.add\_entry\_videos(video\_yt\_id, video\_name, channel\_id, channel\_name, video\_link, video\_path)  
 vid\_analysis.start\_analysing(video\_path)  
 messagebox.showinfo(**'showinfo'**, **'Analysis complete!'**)  
 **else**:  
 messagebox.showinfo(**'showinfo'**, **'This video has already been analysed!'**)  
 **except** requests.exceptions.RequestException:  
 print(**'Could not get id'**)  
  
 **def** download\_video(self, video\_link):  
 **try**:  
 youtubeObject = YouTube(video\_link)  
 youtubeObject = youtubeObject.streams.get\_highest\_resolution()  
 downloaded\_video = youtubeObject.download(self.folder\_path)  
 messagebox.showinfo(**'showinfo'**, **"Download is completed successfully"**)  
 downloaded\_path = os.path.abspath(downloaded\_video)  
 **return** downloaded\_path  
 **except** KeyError:  
 print(**"An error has occurred"**)  
 **return None  
  
 def** get\_video\_id(self, link):  
 query = urlparse(link)  
 **if** query.hostname == **'youtu.be'**:  
 video\_id = query.path[1:]  
 **return** video\_id  
 **if** query.hostname **in** (**'www.youtube.com'**, **'youtube.com'**):  
 **if** query.path == **'/watch'**:  
 p = parse\_qs(query.query)  
 video\_id = p[**'v'**][0]  
 **return** video\_id  
 **if** query.path[:7] == **'/embed/'**:  
 video\_id = query.path.split(**'/'**)[2]  
 **return** video\_id  
 **if** query.path[:3] == **'/v/'**:  
 video\_id = query.path.split(**'/'**)[2]  
 **return** video\_id  
  
 **return None  
  
 def** get\_video\_data(self, video\_id, api\_key):  
 snippet\_url = **f"https://www.googleapis.com/youtube/v3/videos?part=snippet&id={**video\_id**}&key={**api\_key**}"** print(snippet\_url)  
 json\_url\_snippet = requests.get(snippet\_url)  
 data = json.loads(json\_url\_snippet.text)  
 **try**:  
 video\_name = data[**'items'**][0][**'snippet'**][**'title'**]  
 channel\_id = data[**'items'**][0][**'snippet'**][**'channelId'**]  
  
 **return** video\_name, channel\_id  
 **except** KeyError:  
 print(**f'Error! Could not get data.'**)  
  
 **def** get\_channel\_stats(self, channel\_id, api\_key):  
 snippet\_url = **f"https://www.googleapis.com/youtube/v3/channels?part=snippet&id={**channel\_id**}&key={**api\_key**}"** print(snippet\_url)  
 json\_url\_snippet = requests.get(snippet\_url)  
 data = json.loads(json\_url\_snippet.text)  
 **try**:  
 channel\_name = data[**'items'**][0][**'snippet'**][**'title'**]  
 **return** channel\_name  
 **except** KeyError:  
 print(**f'Error! Could not get data.'**)

**Файл video\_analysis.py**

**import** cv2  
**import** numpy **as** np  
**from** datetime **import** datetime  
**from** keras.models **import** load\_model  
  
**from** trained\_models.wide\_resnet **import** WideResNet  
**from** results\_analyser **import** ResultsAnalysis  
**from** db\_manager **import** DBManager  
  
db\_man = DBManager()  
res\_analysis = ResultsAnalysis()  
  
  
**class** VideoAnalysis:  
 **def** get\_labels(self, dataset\_name):  
 **if** dataset\_name == **'emotions'**:  
 **return** {0: **'Angry'**, 1: **'Disgusted'**, 2: **'Scared'**, 3: **'Happy'**,  
 4: **'Sad'**, 5: **'Surprised'**, 6: **'Neutral'**}  
 **elif** dataset\_name == **'gender'**:  
 **return** {0: **'Female'**, 1: **'Male'**}  
 **else**:  
 **raise** Exception(**'Invalid dataset name'**)  
  
 **def** apply\_offsets(self, face\_coordinates, offsets):  
 x, y, width, height = face\_coordinates  
 x\_off, y\_off = offsets  
 **return** x - x\_off, x + width + x\_off, y - y\_off, y + height + y\_off  
  
 **def** draw\_text(self, coordinates, image\_array, text, color, x\_offset=0, y\_offset=0,  
 font\_scale=2, thickness=2):  
 x, y = coordinates[:2]  
 cv2.putText(image\_array, text, (x + x\_offset, y + y\_offset),  
 cv2.FONT\_HERSHEY\_SIMPLEX,  
 font\_scale, color, thickness, cv2.LINE\_AA)  
  
 **def** preprocess\_input(self, x, v2=**True**):  
 x = x.astype(**'float32'**)  
 x = x / 255.0  
 **if** v2:  
 x = x - 0.5  
 x = x \* 2.0  
 **return** x  
  
 **def** start\_analysing(self, video\_path):  
 video\_capture = cv2.VideoCapture(video\_path)  
 *# find id of video based on path* video\_id = db\_man.get\_video\_id\_from\_path(video\_path)  
 **if** video\_id **is not None**:  
 print(video\_id)  
  
 model\_folder\_path = **'trained\_models/'** result\_crop\_photo\_f = **'results\_crop/'** result\_full\_f = **'results\_full/'** detection\_model\_path = model\_folder\_path + **'haarcascade\_frontalface\_default.xml'** emotion\_model\_path = model\_folder\_path + **'emotion\_model.hdf5'** weights\_file = model\_folder\_path + **'weights.28-3.73.hdf5'** emotion\_labels = self.get\_labels(**'emotions'**)  
 gender\_labels = self.get\_labels(**'gender'**)  
 emotion\_offsets = (20, 40)  
  
 face\_detection = cv2.CascadeClassifier(detection\_model\_path)  
 emotion\_classifier = load\_model(emotion\_model\_path, compile=**False**)  
 emotion\_target\_size = emotion\_classifier.input\_shape[1:3]  
 model = WideResNet(64, depth=16, k=8)()  
 model.load\_weights(weights\_file)  
  
 cv2.namedWindow(**'window\_frame'**)  
  
 process\_this\_frame = **True  
 while True**:  
 **if** process\_this\_frame:  
 image = video\_capture.read()[1]  
 **try**:  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 gray\_image = np.squeeze(gray\_image)  
 gray\_image = gray\_image.astype(**'uint8'**)  
 **except** cv2.error:  
 **continue** faces = face\_detection.detectMultiScale(gray\_image, 1.3, 5)  
 **for** face\_coordinates **in** faces:  
 x1, x2, y1, y2 = self.apply\_offsets(face\_coordinates, emotion\_offsets)  
 x, y, w, h = face\_coordinates  
 gray\_face = gray\_image[y1:y2 + 1, x1:x2 + 1]  
 **try**:  
 gray\_face = cv2.resize(gray\_face, emotion\_target\_size)  
 **except** cv2.error:  
 **continue** margin = 0.4  
 img\_h, img\_w, \_ = np.shape(rgb\_image)  
 xw1 = max(int(x1 - margin \* w), 0)  
 yw1 = max(int(y1 - margin \* h), 0)  
 xw2 = min(int(x2 + margin \* w), img\_w - 1)  
 yw2 = min(int(y2 + margin \* h), img\_h - 1)  
 img\_h, img\_w, \_ = np.shape(rgb\_image)  
 rgb\_face = rgb\_image[yw1:yw2, xw1:xw2, :]  
 faces = np.empty((1, 64, 64, 3))  
 faces[0, :, :, :] = cv2.resize(rgb\_face, (64, 64))  
  
 *# emotion classification* gray\_face = self.preprocess\_input(gray\_face, **True**)  
 gray\_face = np.expand\_dims(gray\_face, 0)  
 emotion\_label\_arg = np.argmax(emotion\_classifier.predict(gray\_face))  
 *# age and gender classification* age\_gender\_pred = model.predict(faces)  
 predicted\_gender = age\_gender\_pred[0]  
 ages = np.arange(0, 101).reshape(101, 1)  
 predicted\_age = age\_gender\_pred[1].dot(ages).flatten()  
  
 gender = 0 **if** predicted\_gender[0][0] > 0.4 **else** 1  
 gender = gender\_labels[gender]  
 age = int(predicted\_age[0])  
 emotion = emotion\_labels[emotion\_label\_arg]  
 datetime\_now = datetime.now()  
 timestamp = datetime\_now.timestamp()  
 cv2.rectangle(rgb\_image, (x, y), (x + w, y + h), 2)  
  
 crop\_img = rgb\_image[y:y + h, x:x + w]  
 crop\_img = cv2.cvtColor(crop\_img, cv2.COLOR\_RGB2BGR)  
  
 label = **"{}, {}, {}"**.format(age, gender, emotion)  
  
 fullname = str(str(gender.lower()) + **'\_'** + str(age) + **'\_'** + str(emotion.lower()) + **'\_'** + str(  
 int(timestamp)) + **'.jpg'**)  
 print(fullname)  
 fullname\_4\_crop = str(  
 str(gender.lower()) + **'\_'** + str(age) + **'\_'** + str(emotion.lower()) + **'\_'** + str(  
 int(timestamp)) + **'\_cr.jpg'**)  
  
 db\_man.add\_entry\_results(video\_id, gender, age, emotion, fullname\_4\_crop)  
  
 color = (255, 0, 0)  
 self.draw\_text(face\_coordinates, rgb\_image, label, color, 0, -20, 1, 2)  
 cv2.rectangle(rgb\_image, (x, y), (x + w, y + h), color, 2)  
  
 bgr\_image = cv2.cvtColor(rgb\_image, cv2.COLOR\_RGB2BGR)  
 cv2.imwrite(result\_crop\_photo\_f + fullname\_4\_crop, crop\_img)  
 cv2.imwrite(result\_full\_f + fullname, bgr\_image)  
 cv2.imshow(**'window\_frame'**, bgr\_image)  
  
 **if** cv2.waitKey(1) & 0xFF == ord(**'q'**):  
 video\_capture.release()  
 res\_analysis.analyse\_results(video\_id)  
 cv2.destroyAllWindows()  
  
 process\_this\_frame = **not** process\_this\_frame  
  
 **def** repeat\_analysis(self, video\_id):  
 video\_path = db\_man.get\_path\_from\_video\_id(video\_id)  
 print(video\_path)  
 db\_man.clear\_results(video\_id)  
 self.start\_analysing(video\_path)

**Файл emotion\_detection\_model\_training.py**

**import** matplotlib.pyplot **as** plt  
**from** keras.models **import** Sequential  
**from** keras.layers **import** Flatten, Dropout, Dense, Conv2D, BatchNormalization, Activation, MaxPooling2D  
**from** keras.optimizers **import** Adam  
**from** keras.preprocessing.image **import** ImageDataGenerator  
**from** keras.callbacks **import** ModelCheckpoint, EarlyStopping, ReduceLROnPlateau  
  
picture\_size = 48  
folder\_path = **'/datasets/emotions/images/'**Batch\_size = 128  
datagen\_train = ImageDataGenerator()  
datagen\_test = ImageDataGenerator()  
  
train\_set = datagen\_train.flow\_from\_directory(folder\_path + **"train"**,  
 target\_size=(picture\_size, picture\_size),  
 color\_mode=**'grayscale'**,  
 batch\_size=Batch\_size,  
 class\_mode=**'categorical'**,  
 shuffle=**True**)  
test\_set = datagen\_test.flow\_from\_directory(folder\_path + **"test"**,  
 target\_size=(picture\_size, picture\_size),  
 color\_mode=**'grayscale'**,  
 batch\_size=Batch\_size,  
 class\_mode=**'categorical'**,  
 shuffle=**True**)  
  
no\_of\_classes = 7  
model = Sequential()  
  
model.add(Conv2D(64, (3, 3), padding=**'same'**, input\_shape=(48, 48, 1)))  
model.add(BatchNormalization())  
model.add(Activation(**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.25))  
  
model.add(Conv2D(128, (5, 5), padding=**'same'**))  
model.add(BatchNormalization())  
model.add(Activation(**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.25))  
  
model.add(Conv2D(512, (3, 3), padding=**'same'**))  
model.add(BatchNormalization())  
model.add(Activation(**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.25))  
  
model.add(Conv2D(512, (3, 3), padding=**'same'**))  
model.add(BatchNormalization())  
model.add(Activation(**'relu'**))  
model.add(MaxPooling2D(pool\_size=(2, 2)))  
model.add(Dropout(0.25))  
model.add(Flatten())  
  
model.add(Dense(256))  
model.add(BatchNormalization())  
model.add(Activation(**'relu'**))  
model.add(Dropout(0.25))  
  
model.add(Dense(512))  
model.add(BatchNormalization())  
model.add(Activation(**'relu'**))  
model.add(Dropout(0.25))  
  
model.add(Dense(no\_of\_classes, activation=**'softmax'**))  
  
opt = Adam(lr=0.0001)  
model.compile(optimizer=opt, loss=**'categorical\_crossentropy'**, metrics=[**'accuracy'**])  
model.summary()  
  
checkpoint = ModelCheckpoint(  
 filepath=**"C:/Users/serg/PycharmProjects/CPE\_Coursework\_Tsukanova/trained\_models/emotions\_model.hdf5"**,  
 monitor=**'val\_accuracy'**, verbose=1, save\_best\_only=**True**, mode=**'max'**)  
early\_stopping = EarlyStopping(monitor=**'val\_loss'**, min\_delta=0, patience=3, verbose=1, restore\_best\_weights=**True**)  
reduce\_learningrate = ReduceLROnPlateau(monitor=**'val\_loss'**, factor=0.2, patience=3, verbose=1, min\_delta=0.0001)  
callbacks\_list = [checkpoint, early\_stopping, reduce\_learningrate]  
epochs = 48  
  
history = model.fit\_generator(generator=train\_set,  
 steps\_per\_epoch=train\_set.n // train\_set.batch\_size,  
 epochs=epochs,  
 validation\_data=test\_set,  
 validation\_steps=test\_set.n // test\_set.batch\_size,  
 callbacks=callbacks\_list)

**Файл results\_analyser.py**

**import** pandas **as** pd  
**import** seaborn **as** sns  
**import** matplotlib.pyplot **as** plt  
**from** db\_manager **import** DBManager  
  
db\_man = DBManager()  
  
  
**class** ResultsAnalysis:  
 **def** analyse\_results(self, video\_id):  
 data = db\_man.get\_detailed\_data(video\_id)  
 df = pd.DataFrame(data)  
 video\_id = int(video\_id)  
 df.columns = [**'Result ID'**, **'Video ID'**, **'Gender'**, **'Age'**, **'Emotion'**, **'Path'**]  
 df = df.drop(**'Video ID'**, axis=1)  
 df = df.drop(**'Path'**, axis=1)  
  
 stats\_folder = **'stats\_folder/'** gender = df[**'Gender'**].value\_counts()  
 print(**'Gender occ. :\n'**, gender)  
 sns.countplot(x=**'Gender'**, data=df)  
 plt.title(**'Number of Male and Female occurrences'**)  
 gender\_path = str(video\_id) + **'\_gender.png'** plt.savefig(stats\_folder + gender\_path)  
 plt.show()  
  
 genders = [**'Male'**, **'Female'**]  
 g\_counts = [0, 0]  
 **for** i **in** range(len(genders)):  
 **if** genders[i] **not in** df.values:  
 g\_counts[i] = 0  
 **else**:  
 g\_counts[i] = df[**'Gender'**].value\_counts()[genders[i]]  
 i += 1  
  
 print(g\_counts)  
 male\_count = int(g\_counts[0])  
 female\_count = int(g\_counts[1])  
  
  
 emotion = df[**'Emotion'**].value\_counts()  
 print(**'Emotion occ. :\n'**, emotion)  
 sns.countplot(x=**'Emotion'**, data=df)  
 plt.title(**'Number of each emotion occurrences'**)  
 emotion\_path = str(video\_id) + **'\_emotion.png'** plt.savefig(stats\_folder + emotion\_path)  
 plt.show()  
  
 emotions = [**'Angry'**, **'Scared'**, **'Happy'**, **'Sad'**, **'Surprised'**, **'Neutral'**]  
 counts = [0, 0, 0, 0, 0, 0]  
 **for** i **in** range(len(emotions)):  
 **if** emotions[i] **not in** df.values:  
 counts[i] = 0  
 **else**:  
 counts[i] = df[**'Emotion'**].value\_counts()[emotions[i]]  
 i += 1  
  
 print(counts)  
 angry\_count = int(counts[0])  
 scared\_count = int(counts[1])  
 happy\_count = int(counts[2])  
 sad\_count = int(counts[3])  
 surprised\_count = int(counts[4])  
 neutral\_count = int(counts[5])  
  
 age\_mean = df[**'Age'**].mean().item()  
 print(**'Mean age:'**, age\_mean)  
 sns.countplot(x=**'Age'**, data=df)  
 plt.title(**'Number of each age occurrences'**)  
 age\_count\_path = str(video\_id) + **'\_age\_count.png'** plt.savefig(stats\_folder + age\_count\_path)  
 plt.show()  
 sns.boxplot(x=**'Age'**, data=df)  
 plt.title(**'Age boxplot'**)  
 age\_box\_path = str(video\_id) + **'\_age\_box.png'** plt.savefig(stats\_folder + age\_box\_path)  
 plt.show()  
 db\_man.add\_gen\_data(video\_id, male\_count, female\_count, angry\_count, scared\_count, happy\_count,  
 sad\_count, surprised\_count, neutral\_count, age\_mean, gender\_path, age\_count\_path,  
 age\_box\_path,  
 emotion\_path)

**Файл table\_manager.py**

**from** tkinter **import** messagebox  
**import** cv2  
**from** db\_manager **import** DBManager  
**from** yt\_manager **import** YoutubeManager  
  
yt\_man = YoutubeManager()  
db\_man = DBManager()  
  
  
**class** TableManager:  
  
 **def** deselect(self, tree):  
 selected = tree.focus()  
 tree.selection\_remove(selected)  
  
 **def** query\_videos\_database(self, tree):  
 **for** record **in** tree.get\_children():  
 tree.delete(record)  
  
 records = db\_man.videos\_for\_tree()  
 count = 0  
  
 **for** record **in** records:  
 **if** count % 2 == 0:  
 tree.insert(parent=**''**, index=**'end'**, iid=count, text=**''**,  
 values=(record[0], record[1], record[2], record[3]))  
 **else**:  
 tree.insert(parent=**''**, index=**'end'**, iid=count, text=**''**,  
 values=(record[0], record[1], record[2], record[3]))  
 count += 1  
  
 **def** query\_results\_database(self, tree, video\_id):  
 **for** record **in** tree.get\_children():  
 tree.delete(record)  
  
 records = db\_man.results\_for\_tree(video\_id)  
 count = 0  
  
 **for** record **in** records:  
 **if** count % 2 == 0:  
 tree.insert(parent=**''**, index=**'end'**, iid=count, text=**''**,  
 values=(record[0], record[1], record[2], record[3], record[4], record[5]))  
 **else**:  
 tree.insert(parent=**''**, index=**'end'**, iid=count, text=**''**,  
 values=(record[0], record[1], record[2], record[3], record[4], record[5]))  
 count += 1  
  
 **def** see\_video\_result(self, tree, video\_id):  
 **try**:  
 video\_path = db\_man.get\_path\_from\_video\_id(video\_id)  
 cap = cv2.VideoCapture(video\_path)  
  
 **if** video\_path == **''**:  
 messagebox.showerror(**'showerror'**, **'Empty address. Try again'**)  
 **elif not** cap.isOpened():  
 messagebox.showinfo(**'showinfo'**, **'Downloading video. Wait a minute!'**)  
 video\_link = db\_man.get\_link\_from\_id(video\_id)  
 yt\_man.download\_video(video\_link)  
 self.see\_video\_result(tree, video\_id)  
 **else**:  
 **while** cap.isOpened():  
 ret, frame = cap.read()  
 **if** ret:  
 cv2.imshow(**'Frame'**, frame)  
 **if** cv2.waitKey(25) & 0xFF == ord(**'q'**):  
 **break  
 else**:  
 **break** cap.release()  
 **except** IndexError:  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
 **except** cv2.error:  
 messagebox.showerror(**'showerror'**, **'OpenCV error'**)  
  
 self.deselect(tree)  
  
 **def** remove\_one(self, tree, video\_id):  
 **try**:  
 x = tree.selection()[0]  
 tree.delete(x)  
 db\_man.delete\_entry(video\_id)  
 self.deselect(tree)  
 messagebox.showinfo(**'showinfo'**, **'Record successfully deleted'**)  
 **except** IndexError:  
 self.deselect(tree)  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
  
 **def** view\_photo(self, tree):  
 **try**:  
 folder\_path = **'results\_crop/'** *#change route if needed* selected = tree.focus()  
 values = tree.item(selected, **'values'**)  
 img\_path = folder\_path + values[5]  
  
 img = cv2.imread(img\_path, cv2.IMREAD\_COLOR)  
 **if** values[5] == **''**:  
 messagebox.showerror(**'showerror'**, **'Empty address. Try again'**)  
 **elif** img **is None**:  
 messagebox.showerror(**'showerror'**, **'Could not file photo'**)  
 **else**:  
 scale\_percent = 50  
 width = int(img.shape[1] \* scale\_percent / 100)  
 height = int(img.shape[0] \* scale\_percent / 100)  
 dim = (width, height)  
  
 name = values[5]  
 resized\_img = cv2.resize(img, dim)  
 cv2.imshow(name, resized\_img)  
 cv2.waitKey(0)  
  
 **except** IndexError:  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
 **except** cv2.error:  
 messagebox.showerror(**'showerror'**, **'OpenCV error'**)  
 self.deselect(tree)

**Файл db\_manager.py**

**from** tkinter **import** messagebox  
**import** mysql.connector  
**import** os  
  
  
**class** DBManager:  
  
 **def** check\_video\_ex(self, video\_yt\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**'''SELECT \* FROM videos WHERE video\_yt\_id = (%s) '''**, [video\_yt\_id])  
 video\_id = cursor.fetchone()  
 **if** video\_id **is not None**:  
 cursor.close()  
 db.close()  
 print(**'Already exists'**)  
 **return True  
 else**:  
 print(**'New entry'**)  
 **return False  
 except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** check\_channel\_ex(self, channel\_yt\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**'''SELECT \* FROM channels WHERE channel\_yt\_id = (%s) '''**, [channel\_yt\_id])  
 channel\_id = cursor.fetchone()  
 **if** channel\_id **is not None**:  
 channel\_id = channel\_id[0]  
 cursor.close()  
 db.close()  
 print(**'Channel already exists'**)  
 **return** channel\_id  
 **else**:  
 print(**'New channel entry'**)  
 **return None  
 except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** add\_entry\_channels(self, channel\_yt\_id, channel\_name):  
 **try**:  
 channel\_id = self.check\_channel\_ex(channel\_yt\_id)  
 **if** channel\_id **is None**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**''' INSERT INTO channels (channel\_yt\_id, channel\_id) VALUES (%s,%s)'''**,  
 (channel\_yt\_id, channel\_name))  
 print(**'New channel added'**)  
 db.commit()  
 channel\_id = self.check\_channel\_ex(channel\_yt\_id)  
 cursor.close()  
 db.close()  
 **return** channel\_id  
 **else**:  
 **return** channel\_id  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** add\_entry\_videos(self, video\_yt\_id, video\_name, channel\_yt\_id, channel\_name, video\_yt\_link, video\_path):  
 **try**:  
 channel\_id = self.add\_entry\_channels(channel\_yt\_id, channel\_name)  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**''' INSERT INTO videos (video\_yt\_id, video\_name, channel\_id, video\_yt\_link, video\_path)  
 VALUES (%s,%s, %s,%s,%s)'''**,  
 (video\_yt\_id, video\_name, channel\_id, video\_yt\_link, video\_path))  
 print(**'Value added'**)  
 db.commit()  
 cursor.close()  
 db.close()  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** get\_gen\_res\_data(self, video\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**'''SELECT \* FROM final\_results WHERE res\_vid\_id = (%s) '''**, [video\_id])  
 data = cursor.fetchall()  
 cursor.close()  
 db.close()  
 **return** data  
 **except** IndexError:  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** get\_detailed\_data(self, video\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor()  
 cursor.execute(**'''SELECT \* FROM results WHERE video\_id\_fk = %s'''**, [video\_id])  
 data = cursor.fetchall()  
 cursor.close()  
 db.close()  
 **return** data  
 **except** IndexError:  
 messagebox.showerror(**'showerror'**, **'Choose an entry'**)  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** add\_gen\_data(self, video\_id, male\_count, female\_count, angry\_count, scared\_count, happy\_count,  
 sad\_count, surprised\_count, neutral\_count, age\_mean, gender\_path, age\_count\_path,  
 age\_box\_path, emotion\_path):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor()  
 cursor.execute(**''' INSERT INTO final\_results (res\_vid\_id, male\_count, female\_count, angry\_count, scared\_count, happy\_count,  
 sad\_count, surprised\_count, neutral\_count,avg\_age, gender\_path, age\_count\_path, age\_box\_path, emotion\_path)  
 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)'''**,  
 (video\_id, male\_count, female\_count, angry\_count, scared\_count, happy\_count,  
 sad\_count, surprised\_count, neutral\_count, age\_mean, gender\_path, age\_count\_path, age\_box\_path,  
 emotion\_path))  
 db.commit()  
 cursor.close()  
 db.close()  
 print(**'General Data added to table'**)  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** videos\_for\_tree(self):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(  
 **"SELECT video\_id, video\_name, channel\_name, video\_yt\_link FROM videos INNER JOIN channels ON videos.channel\_id = channels.channel\_id"**)  
 records = cursor.fetchall()  
 cursor.close()  
 db.close()  
 **return** records  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** results\_for\_tree(self, video\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**"SELECT \* FROM results WHERE video\_id\_fk = %s LIMIT 200"**, [video\_id])  
 records = cursor.fetchall()  
 cursor.close()  
 db.close()  
 **return** records  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** get\_path\_from\_video\_id(self, video\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**''' SELECT video\_path FROM videos WHERE video\_id = %s'''**, [video\_id])  
 video\_path = cursor.fetchone()  
 **if** video\_path **is not None**:  
 video\_path = video\_path[0]  
 **else**:  
 video\_path = **''  
 return** video\_path  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** get\_video\_id\_from\_path(self, video\_path):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**'''SELECT video\_id FROM videos WHERE video\_path = (%s) '''**, [video\_path])  
 video\_id = cursor.fetchone()  
 video\_id = video\_id[0]  
 print(**'Analysing video with ID:'** + str(video\_id))  
 cursor.close()  
 db.close()  
 **return** video\_id  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
 **return None  
  
 def** delete\_entry(self, video\_id):  
 video\_path = self.get\_path\_from\_video\_id(video\_id)  
 **try**:  
 os.remove(video\_path)  
 print(**'Deleted file'**)  
 **except** FileNotFoundError:  
 print(**'File was not found'**)  
  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor(buffered=**True**)  
 cursor.execute(**"DELETE from videos WHERE video\_id = %s"**, [video\_id])  
 db.commit()  
 db.close()  
 print(**'Entry successfully deleted.'**)  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** add\_entry\_results(self, video\_id, gender, age, emotion, fullname\_4\_crop):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor()  
 cursor.execute(  
 **'''INSERT INTO results (video\_id\_fk, res\_gender, res\_age, res\_emotion, res\_address) VALUES (%s, %s, %s, %s, %s)'''**,  
 (video\_id, gender, age, emotion, fullname\_4\_crop))  
 db.commit()  
 cursor.close()  
 db.close()  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** clear\_results(self, video\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor()  
 cursor.execute(**'''DELETE FROM results WHERE video\_id\_fk = %s'''**, [video\_id])  
 db.commit()  
 cursor.execute(**'''DELETE FROM final\_results WHERE res\_vid\_id = %s'''**, [video\_id])  
 db.commit()  
 cursor.close()  
 db.close()  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)  
  
 **def** get\_link\_from\_id(self, video\_id):  
 **try**:  
 db = mysql.connector.connect(host=**'localhost'**, user=**'root'**, password=**''**, db=**'cpe\_coursework'**)  
 cursor = db.cursor()  
 cursor.execute(  
 **'''SELECT video\_yt\_link FROM videos WHERE video\_id = %s'''**, [video\_id])  
 video\_link = cursor.fetchone()[0]  
 cursor.close()  
 db.close()  
 **return** video\_link  
 **except** mysql.connector.Error:  
 print(**'MySql.Connector.Error'**)