ДОДАТОК Б

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“___” _____ 2024 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВИЯВЛЕННЯ МІМІЧНИХ ОЗНАК БРЕХНІ ТА ВИРАЗІВ НЕБЕЗПЕЧНОЇ ПОВЕДІНКИ ЛЮДИНИ

Текст програми

КПІ.ІП-1328.045490.05.13

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Діана РОМАНЮК

Київ – 2024

Файл PekmanAnalyzer.py

```
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk
import threading
from PIL import Image, ImageDraw, ImageFont
import cv2
import os
import fcntl

lock_file_path = "app_lock.lock"

def acquire_lock():
    try:
        lock_file = open(lock_file_path, 'w')
        fcntl.flock(lock_file, fcntl.LOCK_EX | fcntl.LOCK_NB)
        return lock_file
    except IOError:
        return None

def release_lock(lock_file):
    try:
        fcntl.flock(lock_file, fcntl.LOCK_UN)
        lock_file.close()
        os.remove(lock_file_path)
    except Exception as e:
        print(f"Error releasing lock: {e}")

class FileSelectorWindow(tk.Toplevel):
    def __init__(self, master):
        super().__init__(master)
        self.title("Pekman analyzer")
        self.set_window_properties()

        label = tk.Label(self, text="Select the file type")
        label.pack(side="top", pady=20)

        self.choice = tk.IntVar()

        image_radio = tk.Radiobutton(self, text="Image",
variable=self.choice, value=1)
        image_radio.pack()

        video_radio = tk.Radiobutton(self, text="Video",
variable=self.choice, value=2)
        video_radio.pack()

        next_button = tk.Button(self, text="Next",
command=self.open_next_window)
        next_button.pack(side="bottom", pady=10, padx=10, anchor="se")

        self.protocol("WM_DELETE_WINDOW", self.on_close)

    def on_close(self):
        self.master.destroy()

    def set_window_properties(self):
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        x_position = (screen_width - 400) // 2
        y_position = (screen_height - 300) // 2
```

```
        self.geometry(f"400x300+{x_position}+{y_position}")

    def open_next_window(self):
        choice = self.choice.get()
        if choice == 1:
            OpenImageWindow(root)
            self.destroy()
        elif choice == 2:
            OpenVideoWindow(root)
            self.destroy()



class ProgressWindow(tk.Toplevel):
    def __init__(self, master):
        super().__init__(master)
        self.title("Pekman analyzer")
        self.set_window_properties()

        self.progress_label = tk.Label(self, text="Processing...")
        self.progress_label.pack(side="top", pady=20)

        self.progress_bar = ttk.Progressbar(self, length=200,
mode="indeterminate")
        self.progress_bar.pack(pady=10)
        self.progress_bar.start()

        self.protocol("WM_DELETE_WINDOW", self.on_close)

    def on_close(self):
        self.master.destroy()

    def set_window_properties(self):
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        x_position = (screen_width - 400) // 2
        y_position = (screen_height - 300) // 2
        self.geometry(f"400x300+{x_position}+{y_position}")


class OpenVideoWindow(tk.Toplevel):
    def __init__(self, master):
        super().__init__(master)
        self.title("Pekman analyzer")
        self.set_window_properties()

        self.progress_window = None

        upload_button = tk.Button(self, text="Upload Video", command=lambda:
self.upload_file("video"))
        upload_button.pack(side="top", pady=20)

        next_button = tk.Button(self, text="Next", state=tk.DISABLED,
command=lambda: self.process_file("video"))
        next_button.pack(side="right", padx=10, anchor="se")

        self.protocol("WM_DELETE_WINDOW", self.on_close)

        back_button = tk.Button(self, text="Back", command=self.on_back)
        back_button.pack(side="left", padx=10, anchor="sw")

    def on_close(self):
```

```
        self.master.destroy()

    def set_window_properties(self):
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        x_position = (screen_width - 400) // 2
        y_position = (screen_height - 300) // 2
        self.geometry(f"400x300+{x_position}+{y_position}")

    def upload_file(self, file_type):
        file_path = filedialog.askopenfilename()

        if file_path:
            if not file_path.lower().endswith(('.mp4', '.avi', '.mov')):
                messagebox.showerror("Error", "Upload only videos!")
                return
            if self.file_size(file_path) > 100:
                messagebox.showerror("Error", "The maximum video size should
be less than 100 MB.")
            else:
                self.update_upload_button(file_path, file_type)
                self.update_next_button(file_path, file_type)

    def file_size(self, file_path):
        return os.path.getsize(file_path) / (1024 * 1024)

    def process_file(self, file_type):
        new_path = self.get_new_path()
        if new_path:
            self.progress_window = ProgressWindow(self.master)
            thread = threading.Thread(target=self.run_main_script,
args=(new_path,))
            thread.start()
            self.destroy()

    def run_main_script(self, new_path):
        if new_path:
            from processing import processing_file
            result = processing_file(new_path)
            self.progress_window.destroy()
            self.display_results_window(result, new_path)

    def display_results_window(self, result, new_path):
        results_window = ResultsWindow(self.master, result, new_path)

    def get_new_path(self):
        upload_button = [widget for widget in self.winfo_children() if
isinstance(widget, tk.Button)][0]
        return upload_button.cget("text").split("\n")[-1]

    def update_upload_button(self, file_path, file_type):
        upload_button = [widget for widget in self.winfo_children() if
isinstance(widget, tk.Button)][0]
        upload_button.config(state=tk.DISABLED, text="The file has already
been uploaded\n{}".format(file_path))

    def update_next_button(self, file_path, file_type):
        next_button = [widget for widget in self.winfo_children() if
                       isinstance(widget, tk.Button) and "Next" in
widget.cget("text")][0]
        next_button.config(state=tk.NORMAL, command=lambda:
self.process_file(file_type))
```

```
    def on_back(self):
        FileSelectorWindow(self.master)
        self.after(800, self.destroy)


class OpenImageWindow(tk.Toplevel):
    def __init__(self, master):
        super().__init__(master)
        self.title("Pekman analyzer")
        self.set_window_properties()

        self.progress_window = None

        upload_button = tk.Button(self, text="Upload Image", command=lambda:
self.upload_file("image"))
        upload_button.pack(side="top", pady=20)

        next_button = tk.Button(self, text="Next", state=tk.DISABLED,
command=lambda: self.process_file("image"))
        next_button.pack(side="right", padx=10, anchor="se")

        self.protocol("WM_DELETE_WINDOW", self.on_close)

        back_button = tk.Button(self, text="Back", command=self.on_back)
        back_button.pack(side="left", padx=10, anchor="sw")

    def on_close(self):
        self.master.destroy()

    def set_window_properties(self):
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        x_position = (screen_width - 400) // 2
        y_position = (screen_height - 300) // 2
        self.geometry(f"400x300+{x_position}+{y_position}")

    def upload_file(self, file_type):
        file_path = filedialog.askopenfilename()

        if file_path:
            if not file_path.lower().endswith(('.png', '.jpg', '.jpeg',
'.gif')):
                messagebox.showerror("Error", "Only upload images!")
                return
            if self.file_size(file_path) > 10:
                messagebox.showerror("Error", "The maximum image size should
be less than 10 MB.")
            else:
                self.update_upload_button(file_path, file_type)
                self.update_next_button(file_path, file_type)

    def file_size(self, file_path):
        return os.path.getsize(file_path) / (1024 * 1024)

    def process_file(self, file_type):
        new_path = self.get_new_path()
        if new_path:
            self.progress_window = ProgressWindow(self.master)
            thread = threading.Thread(target=self.run_main_script,
args=(new_path,))
            thread.start()
```

```
            self.destroy()

    def run_main_script(self, new_path):
        if new_path:
            from processing import processing_file
            result = processing_file(new_path)
            self.progress_window.destroy()
            self.display_results_window(result, new_path)

    def display_results_window(self, result, new_path):
        results_window = ResultsWindow(self.master, result, new_path)

    def get_new_path(self):
        upload_button = [widget for widget in self.winfo_children() if
isinstance(widget, tk.Button)][0]
        return upload_button.cget("text").split("\n")[-1]

    def update_upload_button(self, file_path, file_type):
        upload_button = [widget for widget in self.winfo_children() if
isinstance(widget, tk.Button)][0]
        upload_button.config(state=tk.DISABLED, text="The file has already
been uploaded\n{}".format(file_path))

    def update_next_button(self, file_path, file_type):
        next_button = [widget for widget in self.winfo_children() if
                       isinstance(widget, tk.Button) and "Next" in
widget.cget("text")][0]
        next_button.config(state=tk.NORMAL, command=lambda:
self.process_file(file_type))

    def on_back(self):
        FileSelectorWindow(self.master)
        self.after(800, self.destroy)


class ResultsWindow(tk.Toplevel):
    def __init__(self, master, result, new_path):
        super().__init__(master)
        self.title("Pekman analyzer")
        self.set_window_properties()

        view_result_button = tk.Button(self, text="View Result",
command=lambda: self.view_result(new_path, result))
        view_result_button.pack(side="top", pady=20)

        save_result_button = tk.Button(self, text="Save Result",
command=lambda: self.save_result(new_path, result))
        save_result_button.pack(pady=10)

        self.protocol("WM_DELETE_WINDOW", self.on_close)

        back_button = tk.Button(self, text="Back", command=self.on_back)
        back_button.pack(side="left", padx=10, anchor="sw")

    def on_close(self):
        self.master.destroy()

    def set_window_properties(self):
        screen_width = self.master.winfo_screenwidth()
        screen_height = self.master.winfo_screenheight()
        x_position = (screen_width - 400) // 2
        y_position = (screen_height - 300) // 2
```

```
        self.geometry(f"400x300+{x_position}+{y_position}")

    def on_back(self):
        FileSelectorWindow(self.master)
        self.after(800, self.destroy)

    def view_result(self, new_path, result):
        if isinstance(result, str):
            img = Image.open(new_path)
            draw = ImageDraw.Draw(img)

            img_width, img_height = img.size

            font_size = max(1, int(img_width / 25))

            font = ImageFont.load_default().font_variant(size=font_size)

            text_color = 255

            draw.text((10, 10), result, text_color, font=font)

            img.show()

        elif isinstance(result, list):
            for frame in result:
                cv2.imshow('Processed Video', frame)
                if cv2.waitKey(30) & 0xFF == 27:  # Pressing 'Esc' to exit
                    break

            cv2.destroyAllWindows()

    def save_result(self, new_path, result):
        if isinstance(result, str):
            img = Image.open(new_path)
            draw = ImageDraw.Draw(img)

            img_width, img_height = img.size

            font_size = max(1, int(img_width / 25))

            font = ImageFont.load_default().font_variant(size=font_size)

            text_color = 255

            draw.text((10, 10), result, text_color, font=font)
            save_path =
filedialog.asksaveasfilename(defaultextension=".jpeg",
                                                    filetypes=[("JPEG
files", "*.jpg"), ("All files", "*.*")])
            if save_path:
                img.save(save_path)
        elif isinstance(result, list):
            save_path = filedialog.asksaveasfilename(defaultextension=".mp4",
                                                     filetypes=[("MP4
files", "*.mp4"), ("All files", "*.*")])
            if save_path:
                fourcc = cv2.VideoWriter_fourcc(*'mp4v')
                height, width, _ = result[0].shape
                video_writer = cv2.VideoWriter(save_path, fourcc, 30.0,
(width, height))

                for frame in result:
```

```
                    video_writer.write(frame)

                video_writer.release()

def main():
    global root
    root = tk.Tk()
    root.withdraw()
    root.protocol("WM_DELETE_WINDOW", on_main_window_close)
    FileSelectorWindow(root)
    root.mainloop()

def on_main_window_close():
    root.destroy()
    for window in root.winfo_children():
        if isinstance(window, tk.Toplevel):
            window.destroy()


if __name__ == "__main__":
    main()
```

Файл processing.py

```
from face_detection import preprocess_image
from class_emotion import predict_emotion
from predict_fake_smile import predict_fake_smile
from predict_fake_sad import predict_fake_sad
from predict_fake_fear import predict_fake_fear
from video_classifier import classify_video


def processing_file(new_path):
    result = None

    file_type = new_path.split('.')[-1].lower()
    if file_type in ['jpg', 'jpeg', 'png', 'gif']:
        print(f"Processing image: {new_path}")
        img_path = new_path
        processed_image = preprocess_image(img_path)
        predicted_class = predict_emotion(processed_image)
        print(f"Predicted class: {predicted_class}")

        if predicted_class.lower() == "sad":
            predicted_class = predict_fake_sad(processed_image)
            print(f"Predicted class: {predicted_class}")
        elif predicted_class.lower() == "happy":
            predicted_class = predict_fake_smile(processed_image)
            print(f"Predicted class: {predicted_class}")
        elif predicted_class.lower() == "fear":
            predicted_class = predict_fake_fear(processed_image)
            print(f"Predicted class: {predicted_class}")
        result = predicted_class
    elif file_type in ['mp4', 'avi', 'mov']:
        print(f"Processing video: {new_path}")
        processed_frames = classify_video(new_path, show_video=False)
        result = processed_frames
    else:
        print(f"Unsupported file type: {file_type}")

    return result
```

Файл face_detection.py

```
import cv2
import dlib
import numpy as np
from PIL import Image

def preprocess_image(image_path):
    img = Image.open(image_path)

    cv_image = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2GRAY)

    detector = dlib.get_frontal_face_detector()

    faces = detector(cv_image)

    if faces:
        face = faces[0]
        x, y, w, h = face.left(), face.top(), face.width(), face.height()
        face_roi = cv_image[y:y + h, x:x + w]

        face_roi = cv2.resize(face_roi, (48, 48))

        face_array = np.expand_dims(face_roi, axis=0)
        face_array = np.expand_dims(face_array, axis=-1)

        return face_array
    else:
        print("Face not found")
```

Файл class_emotion.py

```
import numpy as np
from keras.models import load_model

def predict_emotion(image):
    model =
load_model("/Users/dianarom/Desktop/Pekman_analyzer/models/emotions_class_mod
el.h5", compile=False)
    predictions = model.predict(image)
    predicted_class_index = np.argmax(predictions[0])

    class_names = {0: 'angry', 1: 'disgust', 2: 'fear', 3: 'happy', 4:
'neutral', 5: 'sad', 6: 'surprise'}
    class_name = class_names[predicted_class_index]

    return class_name
```

Файл predict_fake_fear.py

```
import numpy as np
from keras.models import load_model

def predict_fake_fear(img_array):
    model =
load_model("/Users/dianarom/Desktop/Pekman_analyzer/models/fear_fake_real_mod
el.h5")
    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions[0])
```

```
    class_names = {0: 'fake fear', 1: 'real fear'}
    class_name = class_names[predicted_class_index]

    return class_name
```

Файл predict_fake_sad.py

```
import numpy as np
from keras.models import load_model

def predict_fake_sad(img_array):
    model =
load_model("/Users/dianarom/Desktop/Pekman_analyzer/models/sad_fake_real_mode
l.h5", compile=False)
    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions[0])

    class_names = {0: 'fake sadness', 1: 'real sadness'}
    class_name = class_names[predicted_class_index]

    return class_name
```

Файл predict_fake_smile.py

```
import numpy as np
from keras.models import load_model

def predict_fake_smile(img_array):
    model =
load_model("/Users/dianarom/Desktop/Pekman_analyzer/models/smile_fake_real_mo
del.h5")
    predictions = model.predict(img_array)
    predicted_class_index = np.argmax(predictions[0])

    class_names = {0: 'fake smile', 1: 'real smile'}
    class_name = class_names[predicted_class_index]

    return class_name
```

Файл video_classifier.py

```
import cv2
import dlib
import numpy as np
from keras.models import load_model

def classify_emotion(frame, model, emotion_dict):
    emotion_result = model.predict(frame)

    predicted_class = np.argmax(emotion_result)

    emotion_label = emotion_dict.get(predicted_class)

    return emotion_label

def classify_video(video_path, show_video=True):
    emotion_model =
```

```
load_model("/Users/dianarom/Desktop/Pekman_analyzer/models/emotions_class_mod
el.h5")
    happy_model =
load_model('/Users/dianarom/Desktop/Pekman_analyzer/models/smile_fake_real_mo
del.h5')
    sad_model =
load_model('/Users/dianarom/Desktop/Pekman_analyzer/models/sad_fake_real_mode
l.h5')
    fear_model =
load_model('/Users/dianarom/Desktop/Pekman_analyzer/models/smile_fake_real_mo
del.h5')

    emotions_dict = {0: 'angry', 1: 'disgust', 2: 'fear', 3: 'happy', 4:
'neutral', 5: 'sad', 6: 'surprise'}
    happy_dict = {0: 'fake smile', 1: 'real smile'}
    sad_dict = {0: 'fake sadness', 1: 'real sadness'}
    fear_dict = {0: 'fake fear', 1: 'real fear'}

    detector = dlib.get_frontal_face_detector()

    vs = cv2.VideoCapture(video_path)

    processed_frames = []

    while True:
        ret, frame = vs.read()

        if not ret:
            break

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        faces = detector(gray)

        for face in faces:
            x, y, w, h = face.left(), face.top(), face.width(), face.height()

            face_roi = gray[y:y + h, x:x + w]
            face_roi = cv2.resize(face_roi, (48, 48))
            face_roi = np.reshape(face_roi, [1, 48, 48, 1])

            result = emotion_model.predict(face_roi)
            emotion_label = np.argmax(result)
            emotion_prediction = emotions_dict[emotion_label]

            if emotion_label == 3:
                result_label = classify_emotion(face_roi, happy_model,
happy_dict)
            elif emotion_label == 5:
                result_label = classify_emotion(face_roi, sad_model,
sad_dict)
            elif emotion_label == 2:
                result_label = classify_emotion(face_roi, fear_model,
fear_dict)
            else:
                result_label = ''

            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
            cv2.putText(frame, f"{emotion_prediction} ({result_label})", (x,
y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
```

```
            processed_frames.append(frame)

            if show_video:
                cv2.imshow("Emotion Recognition", frame)

            key = cv2.waitKey(1) & 0xFF
            if key == ord("q"):
                break

    vs.release()
    cv2.destroyAllWindows()

    return processed_frames
```

Файл train_class_emotion.ipynb

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, BatchNormalization, Dropout,
Flatten, Dense, Activation
from keras.optimizers import Adam
from keras import regularizers
import matplotlib.pyplot as plt

#import zipfile
#import os

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

#zip_path = '/content/drive/MyDrive/AI/Pekman/data_arch.zip'

#extract_path = '/content/drive/MyDrive/AI/Pekman/'

#with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    #zip_ref.extractall(extract_path)
train_loc = '/content/drive/MyDrive/AI/Pekman/train'
test_loc = '/content/drive/MyDrive/AI/Pekman/test'


train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

img_size = 48

train_data = train_datagen.flow_from_directory(
    directory=train_loc,
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical",
    subset="training"
```

```
)

test_data = test_datagen.flow_from_directory(
    directory=test_loc,
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), padding='same', input_shape=(48, 48,
1)))
model.add(Activation('mish'))
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('mish'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (5, 5), padding='same'))
model.add(Activation('mish'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), padding='same',
kernel_regularizer=regularizers.l2(0.01)))
model.add(Activation('mish'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('mish'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(256))
model.add(Activation('mish'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Dense(7, activation='softmax'))

model.compile(
    optimizer=Adam(lr=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

epochs = 60
batch_size = 64

model.summary()

history = model.fit(x=train_data, epochs=epochs, validation_data=test_data)

fig, ax = plt.subplots(1, 2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
```

```
fig.set_size_inches(12, 4)


ax[0].plot(train_acc)
if 'val_accuracy' in history.history:
    ax[0].plot(history.history['val_accuracy'])
    ax[0].legend(['Train', 'Validation'], loc='upper left')
else:
    ax[0].legend(['Train'], loc='upper left')
ax[0].set_title('Training Accuracy vs Validation Accuracy')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('Epoch')

ax[1].plot(train_loss)
if 'val_loss' in history.history:
    ax[1].plot(history.history['val_loss'])
    ax[1].legend(['Train', 'Validation'], loc='upper left')
else:
    ax[1].legend(['Train'], loc='upper left')
ax[1].set_title('Training Loss vs Validation Loss')
ax[1].set_ylabel('Loss')
ax[1].set_xlabel('Epoch')

plt.show()

model.save('/content/drive/MyDrive/AI/Pekman/my_data/emotions_class_model.h5'
)
```

Файл sad_fake_real_train.ipynb

```
import os
import zipfile
import matplotlib.pyplot as plt
from google.colab import drive
from keras.models import Sequential
from keras.layers import Activation, BatchNormalization, Conv2D, Dense,
Dropout, Flatten, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# zip_path = '/content/drive/MyDrive/AI/Pekman/my_data/sad/sad_data.zip'


# extract_path = '/content/drive/MyDrive/AI/Pekman/my_data/sad'

# with zipfile.ZipFile(zip_path, 'r') as zip_ref:
#     zip_ref.extractall(extract_path)
drive.mount('/content/drive')

data_loc = '/content/drive/MyDrive/AI/Pekman/my_data/sad/sad_data'


image_files = []
labels = []

for emotion_folder in os.listdir(data_loc):
```

```
    emotion_path = os.path.join(data_loc, emotion_folder)
    if os.path.isdir(emotion_path):
        for image_file in os.listdir(emotion_path):
            image_path = os.path.join(emotion_path, image_file)
            image_files.append(image_path)
            labels.append(emotion_folder)


train_files, test_files, train_labels, test_labels = train_test_split(
    image_files, labels, test_size=0.2, random_state=42, stratify=labels
)

img_size = 48
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2]
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_data = train_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame({'filename': train_files, 'class': train_labels}),
    x_col="filename",
    y_col="class",
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

test_data = test_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame({'filename': test_files, 'class': test_labels}),
    x_col="filename",
    y_col="class",
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

labels = {value: key for key, value in train_data.class_indices.items()}

print("Label Mappings for classes present in the training and validation
datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 12))
idx = 0

for i in range(2):
    for j in range(5):
        label = labels[np.argmax(train_data[0][1][idx])]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_data[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1
```

```
plt.tight_layout()
plt.suptitle("Sample Training Images", fontsize=21)
plt.show()


def create_model():
    model = Sequential([
        Conv2D(filters=128, kernel_size=(5, 5), padding='valid',
input_shape=(48, 48, 1)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=64, kernel_size=(3, 3), padding='valid',
kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=32, kernel_size=(3, 3), padding='valid',
kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Flatten(),

        Dense(units=256, activation='relu'),
        Dropout(0.5),
        Dense(units=2, activation='softmax')
    ])

    return model

cnn_model = create_model()
print(cnn_model.summary())

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1),
patience=5)

epochs = 15
batch_size = 32

optimizer = Adam(learning_rate=0.001)

cnn_model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

history = cnn_model.fit(train_data, epochs=100, validation_data=test_data,
                        verbose=2,
                        callbacks=[reduce_lr])


train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

train_loss = history.history['loss']
val_loss = history.history['val_loss']

learning_rate = history.history['lr']
```

```
fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')

ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')

plt.tight_layout()
plt.show()

cnn_model.save('/content/drive/MyDrive/AI/Pekman/my_data/smile/smile_fake_rea
l_model.h5')
```

Файл smile_fake_real_train.ipynb

```
mport os
import zipfile
import matplotlib.pyplot as plt
from google.colab import drive
from keras.models import Sequential
from keras.layers import Activation, BatchNormalization, Conv2D, Dense,
Dropout, Flatten, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np


#%%

# zip_path = '/content/drive/MyDrive/AI/Pekman/my_data/smile/smile_data.zip'


# extract_path = '/content/drive/MyDrive/AI/Pekman/my_data/smile'

# with zipfile.ZipFile(zip_path, 'r') as zip_ref:
#     zip_ref.extractall(extract_path)
#%%
drive.mount('/content/drive')

data_loc = '/content/drive/MyDrive/AI/Pekman/my_data/sad/sad_data'
```

```
image_files = []
labels = []

for emotion_folder in os.listdir(data_loc):
    emotion_path = os.path.join(data_loc, emotion_folder)
    if os.path.isdir(emotion_path):
        for image_file in os.listdir(emotion_path):
            image_path = os.path.join(emotion_path, image_file)
            image_files.append(image_path)
            labels.append(emotion_folder)


train_files, test_files, train_labels, test_labels = train_test_split(
    image_files, labels, test_size=0.2, random_state=42, stratify=labels
)

img_size = 48
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2]
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_data = train_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame({'filename': train_files, 'class': train_labels}),
    x_col="filename",
    y_col="class",
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

test_data = test_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame({'filename': test_files, 'class': test_labels}),
    x_col="filename",
    y_col="class",
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

labels = {value: key for key, value in train_data.class_indices.items()}

print("Label Mappings for classes present in the training and validation
datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 12))
idx = 0

for i in range(2):
    for j in range(5):
        label = labels[np.argmax(train_data[0][1][idx])]
```

```
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_data[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1

plt.tight_layout()
plt.suptitle("Sample Training Images", fontsize=21)
plt.show()


def create_model():
    model = Sequential([
        Conv2D(filters=128, kernel_size=(5, 5), padding='valid',
input_shape=(48, 48, 1)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=64, kernel_size=(3, 3), padding='valid',
kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=32, kernel_size=(3, 3), padding='valid',
kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Flatten(),

        Dense(units=256, activation='relu'),
        Dropout(0.5),
        Dense(units=2, activation='softmax')
    ])

    return model

cnn_model = create_model()
print(cnn_model.summary())

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1),
patience=5)

epochs = 15
batch_size = 32

optimizer = Adam(learning_rate=0.001)

cnn_model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

history = cnn_model.fit(train_data, epochs=100, validation_data=test_data,
                        verbose=2,
                        callbacks=[reduce_lr])


train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

train_loss = history.history['loss']
```

```
val_loss = history.history['val_loss']

learning_rate = history.history['lr']

fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')

ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')

plt.tight_layout()
plt.show()

cnn_model.save('/content/drive/MyDrive/AI/Pekman/my_data/smile/smile_fake_rea
l_model.h5')
```

Файл fear_fake_real_train.ipynb

```
import os
import zipfile
import matplotlib.pyplot as plt
from google.colab import drive
from keras.models import Sequential
from keras.layers import Activation, BatchNormalization, Conv2D, Dense,
Dropout, Flatten, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np


#%%

# zip_path = '/content/drive/MyDrive/AI/Pekman/my_data/fear/fear_data.zip'


# extract_path = '/content/drive/MyDrive/AI/Pekman/my_data/fear'

# with zipfile.ZipFile(zip_path, 'r') as zip_ref:
#     zip_ref.extractall(extract_path)
```

```
#%%
drive.mount('/content/drive')

data_loc = '/content/drive/MyDrive/AI/Pekman/my_data/fear/fear_data'


image_files = []
labels = []

for emotion_folder in os.listdir(data_loc):
    emotion_path = os.path.join(data_loc, emotion_folder)
    if os.path.isdir(emotion_path):
        for image_file in os.listdir(emotion_path):
            image_path = os.path.join(emotion_path, image_file)
            image_files.append(image_path)
            labels.append(emotion_folder)


train_files, test_files, train_labels, test_labels = train_test_split(
    image_files, labels, test_size=0.2, random_state=42, stratify=labels
)

img_size = 48
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2]
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_data = train_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame({'filename': train_files, 'class': train_labels}),
    x_col="filename",
    y_col="class",
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

test_data = test_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame({'filename': test_files, 'class': test_labels}),
    x_col="filename",
    y_col="class",
    target_size=(img_size, img_size),
    batch_size=64,
    color_mode="grayscale",
    class_mode="categorical"
)

labels = {value: key for key, value in train_data.class_indices.items()}

print("Label Mappings for classes present in the training and validation
datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
```

```
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 12))
idx = 0

for i in range(2):
    for j in range(5):
        label = labels[np.argmax(train_data[0][1][idx])]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_data[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1

plt.tight_layout()
plt.suptitle("Sample Training Images", fontsize=21)
plt.show()


def create_model():
    model = Sequential([
        Conv2D(filters=128, kernel_size=(5, 5), padding='valid',
input_shape=(48, 48, 1)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=64, kernel_size=(3, 3), padding='valid',
kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=32, kernel_size=(3, 3), padding='valid',
kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Flatten(),

        Dense(units=256, activation='relu'),
        Dropout(0.5),
        Dense(units=2, activation='softmax')
    ])

    return model

cnn_model = create_model()
print(cnn_model.summary())

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1),
patience=5)

epochs = 15
batch_size = 32

optimizer = Adam(learning_rate=0.001)

cnn_model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

history = cnn_model.fit(train_data, epochs=100, validation_data=test_data,
                        verbose=2,
                        callbacks=[reduce_lr])
```

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

train_loss = history.history['loss']
val_loss = history.history['val_loss']

learning_rate = history.history['lr']

fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')

ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')

plt.tight_layout()
plt.show()

cnn_model.save('/content/drive/MyDrive/AI/Pekman/my_data/fear/fear_fake_real_
model.h5')
```