گزارش پروژه فاز دوم

محمد مهدی دقیقی

9926163

چکیده

این پروژه بر توسعه یک سیستم تشخیص احساسات با استفاده از شبکه‌های عصبی پیچشی (CNN) تمرکز دارد . سیستم برای طبقه‌بندی حالات چهره به هفت دسته :عصبانیت، انزجار، ترس، شادی، غم، شگفتی و حالت خنثی طراحی شده است .پروژه شامل پیش‌پردازش داده‌ها، ساخت مدل، آموزش و تشخیص احساسات به صورت بلادرنگ با استفاده از وب‌کم است .این گزارش جزئیات مجموعه داده‌ها، معماری مدلCNN ، روش‌های آموزش و پیاده‌سازی تشخیص احساسات بلادرنگ را شرح می‌دهد.

مقدمه

تشخیص احساسات از حالات چهره یک وظیفه مهم در کاربردهای مختلف مانند تعامل انسان و کامپیوتر، سیستم های امنیتی و مطالعات روانشناسی است .هدف این پروژه ایجاد یک سیستم تشخیص احساسات با استفاده از تکنیک‌های یادگیری عمیق، به‌ویژهCNN ، برای تحلیل حالات چهره و طبقه‌بندی آن‌ها به کلاس‌های احساسات از پیش تعریف شده است.

مجموع داده

مجموعه داده استفاده شده در این پروژه، مجموعه داده (FER-2013 تشخیص احساسات چهره ۲۰۱۳) است که شامل تصاویر سیاه و سفید از چهره‌ها با ابعاد ۴۸x۴۸ پیکسل است .هر تصویر با یکی از هفت دسته احساسات برچسب‌گذاری شده است.

در ادامه هر فایل به تفصیل توضیح داده شده است :

dataset.py:

```python
import pandas as pd  # Importing pandas for data manipulation
import numpy as np  # Importing numpy for numerical operations
from sklearn.model_selection import train_test_split  # Importing train_test_split for splitting data
from tensorflow.keras.utils import to_categorical  # Importing to_categorical for one-hot encoding
from tensorflow.keras.preprocessing.image import ImageDataGenerator  # Importing ImageDataGenerator for data


# augmentation

# Function to load the dataset from a CSV file
2 usages
def load_dataset(filename):
    data = pd.read_csv(filename)  # Reading the CSV file into a DataFrame
    pixels = data['pixels'].apply(lambda x: np.fromstring(x, sep=' '))  # Converting pixel strings to numpy arrays
    images = np.vstack(pixels.values).reshape(-1, 48, 48, 1).astype('float32')  # Reshaping and stacking the arrays
    # into images
    images /= 255.0  # Normalizing pixel values to the range [0, 1]
    emotions = to_categorical(data['emotion'])  # One-hot encoding the emotion labels
    return train_test_split( *arrays: images, emotions, test_size=0.2, random_state=42)  # Splitting data into training and
    # testing sets


# Function to create data generators for training and testing
2 usages
def get_data_generators(X_train, X_test, y_train, y_test):
    train_datagen = ImageDataGenerator(
        rotation_range=30,  # Randomly rotating images up to 30 degrees
        width_shift_range=0.2,  # Randomly shifting images horizontally by 20%
        height_shift_range=0.2,  # Randomly shifting images vertically by 20%
        shear_range=0.2,  # Randomly shearing images
        zoom_range=0.2,  # Randomly zooming images
        horizontal_flip=True,  # Randomly flipping images horizontally
        fill_mode='nearest'  # Filling empty pixels after transformations
    )
    test_datagen = ImageDataGenerator()  # No augmentation for test data

    train_generator = train_datagen.flow(X_train, y_train, batch_size=64)  # Creating the training data generator
    test_generator = test_datagen.flow(X_test, y_test, batch_size=64)  # Creating the test data generator

    return train_generator, test_generator  # Returning the data generators
```

model.py:

```python
from tensorflow.keras.models import Sequential  # Importing Sequential model type from Keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, \
    BatchNormalization  # Importing layers
from tensorflow.keras.optimizers import Adam  # Importing the Adam optimizer
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping  # Importing callbacks for training


# Function to build the CNN model
1 usage
def build_model():
    model = Sequential([  # Initializing the Sequential model
        Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)),  # First convolutional layer
        BatchNormalization(),  # Batch normalization layer
        MaxPooling2D(pool_size=(2, 2)),  # Max-pooling layer
        Conv2D(64, kernel_size=(3, 3), activation='relu'),  # Second convolutional layer
        BatchNormalization(),  # Batch normalization layer
        MaxPooling2D(pool_size=(2, 2)),  # Max-pooling layer
        Conv2D(128, kernel_size=(3, 3), activation='relu'),  # Third convolutional layer
        BatchNormalization(),  # Batch normalization layer
        MaxPooling2D(pool_size=(2, 2)),  # Max-pooling layer
        Flatten(),  # Flattening layer to convert 2D matrices to 1D vector
        Dense(512, activation='relu'),  # Fully connected dense layer with 512 neurons
        Dropout(0.5),  # Dropout layer for regularization
        Dense(7, activation='softmax')  # Output layer with 7 neurons for 7 emotion classes
    ])
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
                  metrics=['accuracy'])  # Compiling the model
    return model  # Returning the model
```

```python
# Function to train the model
1 usage
def train_model(model, train_generator, test_generator, epochs=50):
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-7)  # Callback to reduce
    # learning rate
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)  # Callback to stop
    # training early

    model.fit(
        train_generator,  # Training data generator
        epochs=epochs,  # Number of epochs
        validation_data=test_generator,  # Validation data generator
        callbacks=[reduce_lr, early_stopping]  # List of callbacks
    )
    model.save('models/emotion_recognition_model.keras')  # Saving the trained model


if __name__ == '__main__':
    from dataset import load_dataset, get_data_generators  # Importing functions from dataset.py

    X_train, X_test, y_train, y_test = load_dataset('../data/fer2013.csv')  # Loading the dataset
    train_generator, test_generator = get_data_generators(X_train, X_test, y_train, y_test)  # Getting data generators
    model = build_model()  # Building the model
    train_model(model, train_generator, test_generator, epochs=100)  # Training the model
```

detect.py:

```python
from mtcnn import MTCNN  # Importing MTCNN for face detection

detector = MTCNN()  # Initializing the MTCNN detector


# Function to detect faces in an image
2 usages
def detect_faces(image):
    results = detector.detect_faces(image)  # Detecting faces
    return results  # Returning the detected faces
```

real_time.py:

```python
import cv2  # Importing OpenCV for video capture and image processing
import numpy as np  # Importing numpy for numerical operations
from tensorflow.keras.models import load_model  # Importing load_model to load the trained model
from detect import detect_faces  # Importing detect_faces function from detect.py

# Dictionary mapping emotion indices to emoji paths and emotion texts
EMOTION_MAP = {
    0: ('emojis/angry.png', 'Angry'),
    1: ('emojis/disgust.png', 'Disgust'),
    2: ('emojis/fear.png', 'Fear'),
    3: ('emojis/happy.png', 'Happy'),
    4: ('emojis/sad.png', 'Sad'),
    5: ('emojis/surprise.png', 'Surprise'),
    6: ('emojis/neutral.png', 'Neutral')
}


# Function to overlay emoji and text on the frame
1 usage
def overlay_emoji_and_text(frame, emoji_path, emotion_text, x, y, width, height):
    emoji_img = cv2.imread(emoji_path, cv2.IMREAD_UNCHANGED)  # Reading the emoji image
    emoji_img = cv2.resize(emoji_img, dsize: (width, height))  # Resizing the emoji image
    try:
        for c in range(0, 3):
            frame[y:y + height, x:x + width, c] = emoji_img[:, :, c] * (emoji_img[:, :, 3] / 255.0) + frame[
                                                                                                      y:y + height,
                                                                                                      x:x + width,
                                                                                                      c] * (
                                                  1.0 - emoji_img[:, :,
                                                        3] / 255.0)  # Overlaying the emoji image
    except Exception as e:
        print("Failed to overlay emoji:", e)  # Printing an error message if overlay fails

    font = cv2.FONT_HERSHEY_SIMPLEX  # Setting the font for text
    cv2.putText(frame, emotion_text, org: (x, y - 10), font, fontScale: 0.5, color: (255, 255, 255), thickness: 2,
                cv2.LINE_AA)  # Adding text to the frame

    return frame  # Returning the frame with overlay
```

```python
# Function to overlay emoji and text on the frame
1 usage
def overlay_emoji_and_text(frame, emoji_path, emotion_text, x, y, width, height):
    emoji_img = cv2.imread(emoji_path, cv2.IMREAD_UNCHANGED)  # Reading the emoji image
    emoji_img = cv2.resize(emoji_img, dsize: (width, height))  # Resizing the emoji image
    try:
        for c in range(0, 3):
            frame[y:y + height, x:x + width, c] = emoji_img[:, :, c] * (emoji_img[:, :, 3] / 255.0) + frame[
                                                                                                    y:y + height,
                                                                                                    x:x + width,
                                                                                                    c] * (
                                                  1.0 - emoji_img[:, :,
                                                        3] / 255.0)  # Overlaying the emoji image
    except Exception as e:
        print("Failed to overlay emoji:", e)  # Printing an error message if overlay fails

    font = cv2.FONT_HERSHEY_SIMPLEX  # Setting the font for text
    cv2.putText(frame, emotion_text, org: (x, y - 10), font, fontScale: 0.5, color: (255, 255, 255), thickness: 2,
                cv2.LINE_AA)  # Adding text to the frame

    return frame  # Returning the frame with overlay
```

```python
# Function to perform real-time emotion recognition
1 usage
def real_time_recognition(model_path):
    model = load_model(model_path)  # Loading the trained model
    cap = cv2.VideoCapture(0)  # Capturing video from the webcam

    while True:
        ret, frame = cap.read()  # Reading a frame from the webcam
        if not ret:
            break  # Exiting if the frame is not read successfully

        faces = detect_faces(frame)  # Detecting faces in the frame
        for face in faces:
            x, y, width, height = face['box']  # Getting the coordinates of the face bounding box
            cv2.rectangle(frame, (x, y), (x + width, y + height), (0, 255, 0), 2)  # Drawing a rectangle around the face
            roi = cv2.cvtColor(frame[y:y + height, x:x + width],
                               cv2.COLOR_BGR2GRAY)  # Converting the face region to grayscale
            roi = cv2.resize(roi, dsize: (48, 48))  # Resizing the face region to 48x48 pixels
            roi = roi.astype('float32') / 255.0  # Normalizing the face region
            roi = np.expand_dims(np.expand_dims(roi, -1), axis: 0)  # Expanding dimensions to match model input shape
            prediction = model.predict(roi)  # Predicting the emotion
            max_index = np.argmax(prediction[0])  # Getting the index of the highest probability emotion
            emoji_path, emotion_text = EMOTION_MAP[max_index]  # Getting the emoji path and emotion text
            frame = overlay_emoji_and_text(frame, emoji_path, emotion_text, x, y, width,
                                           height)  # Overlaying emoji and text

        cv2.imshow( winname: "Real-time Emotion Recognition", frame)  # Displaying the frame
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break  # Exiting if 'q' key is pressed

    cap.release()  # Releasing the video capture
    cv2.destroyAllWindows()  # Destroying all OpenCV windows


if __name__ == "__main__":
    real_time_recognition('models/emotion_recognition_model.keras')  # Running the real-time recognition function
```

تشخیص احساسات بلادرنگ

برای امکان تشخیص احساسات به صورت بلادرنگ، یک اسکریپت جداگانه (real_time.py) پیاده‌سازی شده است .این اسکریپت فریم‌های ویدئویی را از وب‌کم می‌گیرد، چهره‌ها را تشخیص می‌دهد، چهره‌های تشخیص داده شده را پیش‌پردازش می‌کند، احساسات آن‌ها را با استفاده از مدل آموزش دیده پیش‌بینی می‌کند و ایموجی و متن احساسات مربوطه را بر روی فریم‌های ویدئویی نمایش می‌دهد.

تشخیص چهره

تشخیص چهره با استفاده از کتابخانه MTCNN انجام می‌شود که قابلیت تشخیص چهره قوی دارد .چهره‌های تشخیص داده شده سپس برش داده شده و برای پیش‌بینی احساسات پیش‌پردازش می‌شوند.

نتیجه‌گیری

این پروژه به طور موفقیت‌آمیز یک سیستم تشخیص احساسات با استفاده از CNN پیاده‌سازی می‌کند .این سیستم قادر به تشخیص احساسات از حالات چهره به صورت بلادرنگ است و بازخورد بصری فوری از طریق ایموجی‌ها و متن نمایش می‌دهد .بهبودهای آینده می‌تواند شامل استفاده از مدل‌های پیچیده‌تر، ادغام مجموعه داده‌های بیشتر برای بهبود تعمیم و بهینه‌سازی عملکرد بلادرنگ سیستم باشد.