SAPIENZA
UNIVERSITÀ DI ROMA

GROUP 21

Statistical Learning

Final Report

Authors:
Deniz Yilmaz (ID: 2108621)
Onur Ozan Sunger (ID: 2113119)
Umut Altun (ID: 2101934)
Selin Topaloglu (ID: 2311462)
Mayis Atayev (ID: 2104359)

Professor:
Pierpaolo Brutti

Date: July 7, 2024

Abstract

This project involves developing a real-time emotion detection system integrated with Spotify to enhance user experience by playing emotion-specific playlists. The system utilizes a Raspberry Pi for deployment, employing TensorFlow Lite for efficient emotion detection and Spotipy for Spotify API interaction. This report covers the data collection, model training, implementation, results, and future improvements.

Contents

1 Introduction

The integration of emotion detection with multimedia services has the potential to revolutionize user interactions. This project aims to develop a system capable of detecting emotions in real-time using a webcam and subsequently playing Spotify playlists that match the detected emotions. This report details the development and deployment of this system, highlighting the methodologies, technologies, and results.

2 Objectives

- Develop a system for real-time emotion detection using a webcam.
- Integrate the emotion detection system with Spotify to play playlists based on detected emotions.

3 Tools and Technologies

3.1 TensorFlow

Used for model training and conversion to TensorFlow Lite.

3.2 OpenCV

Utilized for real-time face detection.

3.3 TensorFlow Lite (TFLite)

Enables efficient model inference on the Raspberry Pi.

3.4 Spotipy

Python library for Spotify Web API interaction.

3.5 Raspberry Pi

Hardware platform for deployment.

4 Data Collection

4.1 Dataset

FER-2013 dataset with 7 emotion classes: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral.

4.2 Preprocessing

Images were normalized, and labels were converted to one-hot encoding for model training.

4.3 Code Snippet: Data Loading and Preprocessing

```
import pandas as pd
import numpy as np
from tensorflow.keras.utils import to_categorical

def load_data(csv_path):
    df = pd.read_csv(csv_path, header=None)
    data = df.values
    X = data[:, 1:].astype(np.float32) / 255.0  # Normalizing pixel values
    y = to_categorical(data[:, 0], num_classes=7)  # Converting labels to
    ↪ one-hot encoding
    X = X.reshape(-1, 48, 48, 1)  # Reshape for CNN input
    return X, y

X_train, y_train = load_data('train_data.csv')
X_val, y_val = load_data('validation_data.csv')
```

5 Model Architecture

5.1 Layers

- Convolutional layers with BatchNormalization and Dropout.
- Dense layers with Dropout.
- Softmax output layer.

5.2 Training

- ***Optimizer*** **:** Stochastic Gradient Descent (SGD) with momentum.
- ***Callbacks*** **:** ReduceLROnPlateau, EarlyStopping.
- Achieved approximately **65%** accuracy.

5.3 Code Snippet: Model Architecture

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1), padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),
    Flatten(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(7, activation='softmax')
])

optimizer = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-6)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

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

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    batch_size=64,
    callbacks=[reduce_lr, early_stopping]
)

model.save('simplified_emotion_model_from_csv.h5')
```

6 Model Simplification and Conversion

6.1 Model Simplification:

Reduced complexity to improve performance on Raspberry Pi by focusing on reducing layers and parameters.

6.2 Conversion to TensorFlow Lite:

- Loaded the Keras model and converted it using the TFLite Converter.
- Enabled optimizations to reduce size and increase efficiency.
- Addressed compatibility issues and memory constraints.
- Result: Successfully converted model ready for deployment on Raspberry Pi.

6.3 Code Snippet: Conversion to TensorFlow Lite

```python
import tensorflow as tf

# Convert to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]  # Enable optimizations to reduce size
tflite_model = converter.convert()

# Save the converted model
with open('simplified_emotion_model_from_csv.tflite', 'wb') as f:
    f.write(tflite_model)
```

7 Implementation on Raspberry Pi

7.1 Environment Setup:

- Installed necessary libraries: OpenCV, numpy, tflite_runtime.
- Configured Raspberry Pi for efficient model inference.

7.2 Script Overview

- Face Detection: Used OpenCV's Haar Cascade for real-time face detection, focusing on the closest face to the camera.
- Emotion Detection: Preprocessed grayscale face images and used TFLite model for emotion inference.
- Real-time Processing: Displayed colored video feed with emotion labels, ensuring minimal latency.
- Output: Visual indicators for detected emotions on live video.

7.3 Code Snippet: Raspberry Pi Implementation

```python
import cv2
import numpy as np
import tensorflow as tf
from tflite_runtime.interpreter import Interpreter
import spotipy
from spotipy.oauth2 import SpotifyOAuth

# Load TFLite model and allocate tensors
interpreter = Interpreter(model_path="simplified_emotion_model_from_csv.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Initialize Spotify API
sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id='your_client_id',
                                               client_secret='your_client_secret',
                                               redirect_uri='your_redirect_uri',
                                               scope='user-modify-playback-state'))

# Function to process the image and predict emotion
def predict_emotion(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(gray, (48, 48))
    input_data = np.expand_dims(np.expand_dims(resized, axis=0), axis=3).astype(np.float32)
    interpreter.set_tensor(input_details[0]['index'], input_data)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])
```

```python
    return np.argmax(output_data)

# Setup video capture
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    emotion = predict_emotion(frame)
    # Mapping emotion to Spotify playlist
    if emotion == 0:  # Angry
        playlist_uri = "spotify:playlist:angry_playlist_uri"
    elif emotion == 1:  # Disgust
        playlist_uri = "spotify:playlist:disgust_playlist_uri"
    # Add other emotions mappings...

    # Play playlist
    sp.start_playback(context_uri=playlist_uri)

    cv2.imshow('Emotion Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()

```

8 Spotify Integration

8.1 Objective:

Enhance user experience by playing songs that match the detected emotion.

8.2 Setup:

- Created a Spotify developer account.
- Generated Client ID and Client Secret.
- Configured authentication with redirect URI.

8.3 Script Functionality:

- Authenticated with Spotify API.

- Shuffled and played playlists based on detected emotions.
- Ensured continuous playback of a song for at least 15 seconds, even if emotions change.

8.4 Emotion Playlists:

- Fear: [Playlist URI]
- Disgust: [Playlist URI]
- Happy: [Playlist URI]
- Angry: [Playlist URI]
- Sad: [Playlist URI]
- Neutral: [Playlist URI]

9 Implementation Details on Raspberry Pi

9.1 Hardware Setup:

- Raspberry Pi with a webcam for live emotion detection.
- Connected to the internet for accessing Spotify API.

9.2 Software Environment:

- Installed necessary libraries: OpenCV for image processing, TensorFlow Lite for model inference, and Spotipy for Spotify API interaction.
- Configured and activated Python virtual environment.

9.3 Python Script:

- Loaded the pre-trained TFLite model.
- Captured live video feed from the webcam.
- Detected faces and preprocessed them for emotion prediction.
- Controlled Spotify playback based on detected emotions.

10 Results and Performance

10.1 Model Accuracy:

- Achieved approximately 65% accuracy on the validation set.
- Performed well in recognizing emotions such as anger, happiness, and surprise.
- Difficulties in accurately predicting emotions like disgust and fear.

10.2 Live Testing:

- Successfully integrated with OpenCV for real-time face detection.
- Smooth interaction with the Spotify API to play emotion-specific playlists.

10.3 Challenges:

- Handling overlapping emotions.
- Ensuring real-time processing on the Raspberry Pi.

11 Future Improvements

11.1 Model Enhancements:

- Data Augmentation: Enhance the training dataset with augmented images to improve model robustness.
- Advanced Architectures: Explore more advanced deep learning architectures like ResNet or EfficientNet.
- Transfer Learning: Utilize pre-trained models on larger emotion datasets to improve accuracy, particularly for underperforming classes.

11.2 Real-time Performance:

- Optimization: Optimize the model for faster inference on Raspberry Pi, possibly using TensorFlow Lite with further quantization.
- Multithreading: Implement multithreading for parallel processing of face detection and emotion recognition to improve real-time performance.

12 Conclusion

This project successfully developed and deployed a real-time emotion detection system integrated with Spotify on a Raspberry Pi. The system efficiently detects emotions and plays corresponding Spotify playlists, enhancing user experience. Future improvements aim to increase model accuracy and real-time performance, potentially expanding the system's applications and effectiveness.

References

1 Facial Emotion Recognition Dataset, Kaggle.

2 Jones et al., "Deep Learning for Real-Time Image Processing on Embedded Devices,"2020.

3 Smith et al., "Emotion Recognition using Facial Expressions," 2021.