Real-Time Emotion Detection

Digital Image Processing project using CNN Algorithm and FER 2013 Dataset

Professor: Ivica Dimitrovski

Students: Genti Nuhiu, Vigan Demiri

15 June, 2024

Table of Contents

1. Introduction

Emotion detection is a crucial aspect of human-computer interaction, allowing machines to understand and respond appropriately to human emotions. This technology has wide-ranging applications, from enhancing user experience in customer service to improving safety in autonomous driving. By recognizing emotions through facial expressions, computers can better understand and interact with humans in a natural and intuitive way.

2. Project Overview

This project aims to build a real-time emotion detection system using a laptop camera. The system will leverage deep learning techniques, particularly Convolutional Neural Networks (CNNs), to classify emotions from facial expressions. The FER2013 dataset, which contains thousands of labeled facial images, will be used to train the model. The main components of the project include data preprocessing, model training, and real-time emotion detection.

Key steps of this project include:

- Data Preprocessing: Load and normalize the FER2013 dataset.
- Model Training: Build and train a CNN for emotion classification.
- Real-Time Detection: Use the trained model to detect emotions in real-time via a laptop camera.

3. Scope

The primary scope of this project encompasses the development of an emotion detection system that leverages deep learning techniques to classify emotions from facial expressions in real-time. The key elements of the scope include the following:

1. Dataset: The project utilizes the FER2013 dataset, which is a well-known dataset in the field of facial expression recognition. It comprises 35,887 grayscale images of human faces, each having a resolution of 48x48 pixels. The dataset is labeled with seven different emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. This diverse set of labeled images provides a robust foundation for training and evaluating the emotion classification model.
2. Model: The core of the project involves designing and training a Convolutional Neural Network (CNN) tailored for image classification tasks. CNNs are particularly effective for this type of work due to their ability to automatically and adaptively learn spatial hierarchies of features from input images, which is crucial for accurate facial expression recognition.
3. Tools: The project employs a combination of Python programming language and several powerful libraries. TensorFlow and Keras are used for building and training the deep

learning model, providing a comprehensive framework for neural network development. Additionally, OpenCV is utilized for real-time video processing, allowing the system to capture and analyze video feeds from a laptop camera.

4. Output: The ultimate goal of the project is to develop a system capable of detecting and classifying emotions in real-time from a video feed. This system will capture live video from a laptop camera, process each frame to detect faces, and then use the trained CNN to classify the detected facial expressions into one of the seven emotion categories.

4. Objectives

The objectives of this project are centered around three main goals:

1. Developing a Robust Preprocessing Pipeline for the FER2013 Dataset: Before feeding the data into the neural network, it is crucial to preprocess it appropriately. This involves reading and converting the pixel data into a usable format, normalizing the images to ensure consistent input, and one-hot encoding the emotion labels. A robust preprocessing pipeline ensures that the data is in the optimal state for training the model, leading to better performance and accuracy.
2. Designing and Training a High-Accuracy CNN: The project aims to design a CNN architecture that can effectively classify facial expressions with high accuracy. The CNN will be trained on the preprocessed FER2013 dataset, learning to recognize patterns and features associated with different emotions. The training process involves tuning hyperparameters, optimizing the network, and evaluating its performance using metrics such as accuracy, precision, recall, and F1-score.
3. Implementing a Real-Time Emotion Detection System: The final objective is to implement a system that utilizes the trained CNN to detect emotions in real-time. This involves integrating the model with OpenCV to capture and process live video frames from a laptop camera. The system will detect faces in each frame, preprocess the detected faces, and classify their emotions using the CNN. The classified emotions will be displayed on the screen, providing real-time feedback on the emotional states of individuals in the video feed.

By achieving these objectives, the project will result in a functional emotion detection system that combines the power of deep learning with real-time video processing, offering a wide range of applications in fields such as customer service, healthcare, education, marketing, entertainment, and security.

5. Applications

Emotion detection technology has numerous applications, including:

- Customer Service: Enhance customer experience by understanding their emotions during interactions.
- Healthcare: Monitor patients' emotional states for better mental health management.

- Education: Adapt educational content based on students' emotional responses.
- Marketing: Analyze consumer emotions to tailor marketing strategies.
- Entertainment: Create interactive content that responds to users' emotions.
- Security: Detect unusual emotional states that may indicate potential threats.

6. Understanding the FER2013 Dataset

The FER2013 dataset is a publicly available dataset used for facial expression recognition. It consists of 35,887 grayscale images of faces, each 48x48 pixels, labeled with one of seven emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. The dataset is divided into training and test sets, with the training set containing 28,709 images and the test set containing 7,178 images.

Data Format: Each image in the dataset is represented as a string of pixel values, which need to be converted into numpy arrays for processing.

7. Data Preprocessing

Data preprocessing is a crucial step in any machine learning project. For this project, preprocessing involves:

1. Loading the dataset: Reading the CSV file and extracting pixel values and labels.
2. Converting pixel values: Converting the pixel string to a numpy array and reshaping it into a 48x48 matrix.
3. Normalization: Scaling pixel values to the range [0, 1] by dividing by 255.
4. One-Hot Encoding: Converting emotion labels to one-hot encoded vectors.

7.1 Preprocessing Code

```
# data_preprocessing.py


import pandas as pd

import numpy as np


def load_and_preprocess_data(file_path):

    data = pd.read_csv(file_path)


    pixels = data['pixels'].tolist()

    faces = []

    for pixel_sequence in pixels:

        face = [int(pixel) for pixel in pixel_sequence.split(' ')]
```

```
        face = np.asarray(face).reshape(48, 48)
        faces.append(face)

    faces = np.asarray(faces)
    faces = np.expand_dims(faces, -1)  # Add channel dimension
    faces = faces / 255.0  # Normalize pixel values

    emotions = pd.get_dummies(data['emotion']).values
    return faces, emotions

if __name__ == "__main__":
    faces, emotions = load_and_preprocess_data('fer2013.csv')
    np.save('faces.npy', faces)
    np.save('emotions.npy', emotions)
```

8. Model Architecture

For this project, we use a Convolutional Neural Network (CNN) to classify emotions from facial expressions. CNNs are particularly effective for image classification tasks because they can automatically learn spatial hierarchies of features from input images.

8.1 CNN Layers

1. Convolutional Layers: Extract features from input images using convolutional filters. These layers apply a set of learnable filters to the input images, creating feature maps that capture various aspects of the images, such as edges, textures, and patterns.
2. Pooling Layers: Reduce the spatial dimensions of the feature maps, retaining the most important information. This down-sampling process helps to reduce the computational complexity and overfitting.
3. Fully Connected Layers: Perform classification based on the features extracted by the convolutional layers. These layers are densely connected, allowing the model to learn complex representations of the data.
4. Dropout Layers: Prevent overfitting by randomly setting a fraction of input units to 0 during training. Dropout helps to improve the generalization of the model by preventing it from relying too heavily on any particular set of features.

8.2 Model Architecture Code

```
import numpy as np
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from sklearn.model_selection import train_test_split


def create_model():
    model = Sequential()


    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.25))


    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.25))


    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.25))


    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7, activation='softmax'))  # 7 classes for 7 emotions


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


    return model


if __name__ == "__main__":
    faces = np.load('faces.npy')
    emotions = np.load('emotions.npy')
```

```
    X_train, X_val, y_train, y_val = train_test_split(faces, emotions,
test_size=0.2, random_state=42)


    model = create_model()

    history = model.fit(X_train, y_train, epochs=30, batch_size=64,
validation_data=(X_val, y_val))


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

9. Training the Model

Training the model involves several key steps:

1. Splitting the data: The dataset is divided into training and validation sets. This allows us to monitor the model's performance on unseen data during training.
2. Compiling the model: The model is compiled with the Adam optimizer and categorical cross-entropy loss function. The Adam optimizer is a popular choice for deep learning models because it adjusts the learning rate dynamically, making it more efficient.
3. Fitting the model: The model is trained over several epochs, with the training progress monitored using validation data. During each epoch, the model adjusts its weights based on the training data, gradually improving its performance.

9.1 Model Training Code

```
if __name__ == "__main__":

    faces = np.load('faces.npy')

    emotions = np.load('emotions.npy')


    X_train, X_val, y_train, y_val = train_test_split(faces, emotions,
test_size=0.2, random_state=42)


    model = create_model()

    history = model.fit(X_train, y_train, epochs=30, batch_size=64,
validation_data=(X_val, y_val))


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

10. Real-Time Emotion Detection

Real-time emotion detection involves capturing video frames from the laptop camera, detecting faces, and classifying emotions using the trained model.

10.1 Steps for Real-Time Detection

1. Capture Video Frames: Use OpenCV to capture frames from the laptop camera.
2. Face Detection: Detect faces in each frame using Haar cascades. Haar cascades are a popular technique for real-time face detection due to their efficiency and accuracy.
3. Emotion Classification: Preprocess the detected face and predict the emotion using the trained CNN.
4. Display Results: Draw bounding boxes around detected faces and display the predicted emotion.

10.2 Real-Time Detection Code

```
import numpy as np

import cv2

from tensorflow.keras.models import load_model


emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise',
'Neutral']


def detect_emotions(frame, face_cascade, model):

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

    faces = face_cascade.detectMultiScale(gray, 1.3, 5)


    for (x, y, w, h) in faces:

        face = gray[y:y+h, x:x+w]

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

        face = face / 255.0

        face = np.reshape(face, (1, 48, 48, 1))


        prediction = model.predict(face)

        emotion = emotion_labels[np.argmax(prediction)]


        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

```
        cv2.putText(frame, emotion, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
(255, 0, 0), 2)


    return frame


if __name__ == "__main__":
    model = load_model('emotion_detection_model.h5')
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')


    cap = cv2.VideoCapture(0)


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


        frame = detect_emotions(frame, face_cascade, model)
        cv2.imshow('Emotion Detector', frame)


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


    cap.release()
    cv2.destroyAllWindows()
```

11. Code Explanation

11.1 Data Preprocessing (data_preprocessing.py)

- load_and_preprocess_data: This function reads the CSV file, converts pixel strings to numpy arrays, normalizes the pixel values, and one-hot encodes the emotion labels.
- main block: Loads and preprocesses the data, then saves the processed data to numpy files.

11.2 Model Training (train_model.py)

- create_model: Defines the CNN architecture with three convolutional layers, pooling layers, dropout layers, and fully connected layers.
- main block: Loads the preprocessed data, splits it into training and validation sets, compiles and trains the model, and saves the trained model to a file.

11.3 Real-Time Detection (real_time_emotion_detection.py)

- detect_emotions: Captures frames from the laptop camera, detects faces, preprocesses the faces, predicts emotions using the trained model, and displays the results.
- main block: Loads the trained model and Haar cascade for face detection, captures video frames, processes each frame to detect emotions, and displays the results in real-time.

12. Model Evaluation and Improvement

12.1 Evaluation

Evaluate the model using the following metrics:

- Accuracy: The proportion of correctly predicted emotions.
- Precision: The proportion of true positive predictions among all positive predictions.
- Recall: The proportion of true positive predictions among all actual positives.
- F1-Score: The harmonic mean of precision and recall.
- Confusion Matrix: A matrix that shows the number of correct and incorrect predictions for each emotion, providing insight into the model's performance on each class.

12.2 Improvement Techniques

- Data Augmentation: Increase the diversity of training data by applying random transformations (e.g., rotations, flips, zooms) to the images. This helps the model generalize better to new data.
- Hyperparameter Tuning: Experiment with different architectures, learning rates, batch sizes, and other hyperparameters to optimize model performance.
- Transfer Learning: Use pre-trained models (e.g., VGG16, ResNet) and fine-tune them for emotion detection. Transfer learning leverages the knowledge learned from large datasets to improve performance on smaller datasets.
- Ensemble Methods: Combine the predictions of multiple models to improve accuracy and robustness.

13. Challenges

- Imbalanced Data: Some emotions are underrepresented in the FER2013 dataset, making it difficult for the model to learn to recognize them accurately.
- Real-Time Performance: Ensuring the model performs well in real-time with minimal latency is crucial for practical applications.

- Generalization: Ensuring the model generalizes well to different lighting conditions, backgrounds, and face orientations is challenging.

14. Considerations

- Ethical Implications: Addressing privacy concerns and ensuring the technology is used responsibly. It's important to obtain consent before using emotion detection technology and to be transparent about how the data is used.
- Accuracy vs. Speed: Balancing model complexity with real-time performance requirements. More complex models may achieve higher accuracy but can be slower, making them less suitable for real-time applications.

15. Future Work

- More Robust Models: Experiment with more advanced architectures and techniques to improve accuracy and robustness.
- Multi-Modal Emotion Detection: Combine facial expression analysis with other modalities such as voice and text to improve emotion detection accuracy.
- Deployment: Deploy the model on edge devices (e.g., smartphones, IoT devices) or in a web application to make the technology more accessible.
- User Feedback: Incorporate user feedback to continually improve the system's accuracy and usability.

16. Conclusion

This project demonstrates the application of deep learning for real-time emotion detection using a laptop camera. By leveraging the FER2013 dataset and a Convolutional Neural Network (CNN), we have developed a system capable of accurately classifying emotions from facial expressions. While the current model achieves satisfactory performance, there is room for improvement in terms of accuracy, robustness, and generalization. Future work can explore more sophisticated models and multi-modal approaches to enhance emotion detection capabilities.

17. References

- FER2013 Dataset: https://www.kaggle.com/datasets/deadskull7/fer2013
- OpenCV Documentation: https://docs.opencv.org/4.x/
- TensorFlow Documentation: https://www.tensorflow.org/api_docs
- One Hot Encoding in Machine Learning: https://www.geeksforgeeks.org/ml-one-hot-encoding/
- Convolutional Neural Network (CNN) in Machine Learning: https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/

- Python GUI Programming: https://www.tutorialspoint.com/python/python_gui_programming.htm
- Face Detection with Python using OpenCV: https://www.datacamp.com/tutorial/face-detection-python-opencv

18. Appendix: Full Code Listings

18.1 Data Preprocessing (data_preprocessing.py)

```
import pandas as pd
import numpy as np

def load_and_preprocess_data(file_path):
    data = pd.read_csv(file_path)

    pixels = data['pixels'].tolist()
    faces = []
    for pixel_sequence in pixels:
        face = [int(pixel) for pixel in pixel_sequence.split(' ')]
        face = np.asarray(face).reshape(48, 48)
        faces.append(face)

    faces = np.asarray(faces)
    faces = np.expand_dims(faces, -1)  # Add channel dimension
    faces = faces / 255.0  # Normalize pixel values

    emotions = pd.get_dummies(data['emotion']).values
    return faces, emotions

if __name__ == "__main__":
    faces, emotions = load_and_preprocess_data('fer2013.csv')
    np.save('faces.npy', faces)
    np.save('emotions.npy', emotions)
```

18.2 Model Training (train_model.py)

```
import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout

from sklearn.model_selection import train_test_split


def create_model():

    model = Sequential()


    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))

    model.add(MaxPooling2D((2, 2)))

    model.add(Dropout(0.25))


    model.add(Conv2D(64, (3, 3), activation='relu'))

    model.add(MaxPooling2D((2, 2)))

    model.add(Dropout(0.25))


    model.add(Conv2D(128, (3, 3), activation='relu'))

    model.add(MaxPooling2D((2, 2)))

    model.add(Dropout(0.25))


    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.5))

    model.add(Dense(7, activation='softmax'))  # 7 classes for 7 emotions


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


    return model


if __name__ == "__main__":
```

```
    faces = np.load('faces.npy')
    emotions = np.load('emotions.npy')

    X_train, X_val, y_train, y_val = train_test_split(faces, emotions,
test_size=0.2, random_state=42)

    model = create_model()
    history = model.fit(X_train, y_train, epochs=30, batch_size=64,
validation_data=(X_val, y_val))

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

18.3 Real-Time Emotion Detection (real_time_emotion_detection.py)

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

emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise',
'Neutral']

def detect_emotions(frame, face_cascade, model):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:
        face = gray[y:y+h, x:x+w]
        face = cv2.resize(face, (48, 48))
        face = face / 255.0
        face = np.reshape(face, (1, 48, 48, 1))

        prediction = model.predict(face)
        emotion = emotion_labels[np.argmax(prediction)]

        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

```
        cv2.putText(frame, emotion, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
(255, 0, 0), 2)


    return frame


if __name__ == "__main__":
    model = load_model('emotion_detection_model.h5')
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')


    cap = cv2.VideoCapture(0)


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


        frame = detect_emotions(frame, face_cascade, model)
        cv2.imshow('Emotion Detector', frame)


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


    cap.release()
    cv2.destroyAllWindows()
```

This expanded documentation provides a comprehensive guide to the project, covering all aspects from the initial concept to potential future enhancements. By following the outlined steps and using the provided code, you can develop an effective emotion detection system using a laptop camera.