EMOJIFIER: Facial Expressions to Emoji

Miriam Xu, Anh Pham

Abstract— **The Emojifier project aims to bridge digital communication and human emotions. Using computer vision and machine learning, the program detects facial expressions and maps them to corresponding emojis in real time. The implementation builds on an existing repository, adding modern machine libraries and expanding emotion recognition capabilities.**

I. Introduction

As humans, we constantly interpret facial expressions in daily interactions, whether during casual conversations, providing emotional support, or engaging in business negotiations. But what if computers could also be taught to interpret facial expressions?

The Emojifier project aims to develop an emotion recognition system leveraging computer vision and machine learning techniques to detect and analyze human facial expressions. By examining facial features in real-time, this system translates emotions into corresponding emoji representations, providing a creative and intuitive method for conveying emotional states.

Potential applications for this technology span various domains. In virtual communication platforms, emojis have long been used to represent or amplify emotions that cannot be fully conveyed through text. Typically, users browse through extensive emoji lists to select those that best represent their emotional state. An AI-driven system like Emojifier can enhance user experiences by automatically generating emojis, enriching text-based messages, improving emotional context, reducing effort, and minimizing miscommunication. In customer service and user experience research, it can assist in assessing emotional feedback during interactions. Additionally, facial emotion recognition technology can play a valuable role in therapeutic settings, especially for patients who may struggle to articulate complex emotions verbally, as their non-verbal cues provide vital insights for practitioners.

By bridging the gap between digital communication and human emotion, this system fosters greater empathy and understanding in virtual environments.

This project aims to adapt an existing Emojifier repository to modern machine learning libraries while expanding the range of emotions it can recognize. The project identifies commonly used emojis and their associated facial expressions. We recognize that, in real life, emotional expressions vary significantly among individuals. Therefore, an AI-based system must strive for adaptability and avoid rigidity in interpreting human emotions. Our implementation focuses on technical accuracy in facial expression recognition, but we also acknowledge the importance of considering the diversity of emotional expressions, including those of individuals on the autism spectrum, with PTSD, or whose expressions may not align with conventional patterns. While our scope is limited, we recognize that future systems that use facial emotion recognition and similar technologies could be designed to capture these nuances more effectively. This work acknowledges these limitations and highlights the need for more inclusive and nuanced approaches as the field evolves.

II. Related Work

Facial emotion recognition has been explored through various projects, each offering unique approaches and limitations. Below is an analysis of related repositories, followed by a discussion on why the current implementation was chosen as well as the techniques used to achieve the project's goals.

The list below organizes the comparisons to highlight the strengths and weaknesses of each repository, clarifying the rationale for selecting **Facial EmotionRecognition Emojifier (vijuSR)** as the foundation for this project, primarily due to its relative simplicity and documentation. The selected repository is the last repository listed in the table.

- **Face-to-Emoji (ajinkyabedekar)**
  - **Popularity (Stars/Forks)**: Low
  - **Documentation**: Poor
  - **Sample Output**: No
  - **Strengths**: Simple functionality
  - **Limitations**: Poor documentation, unclear models
  - **Notes**: Not suitable for adaptation due to lack of clarity.
- **FacialEmotionDetector (louie-jancevski)**
  - **Popularity (Stars/Forks)**: Moderate
  - **Documentation**: Clear
  - **Sample Output**: Yes
  - **Strengths**: Attractive web design, comprehensive emotion list
  - **Limitations**: Glitches, unstable predictions, complex dependencies
  - **Notes**: Offers ideas for emotion categories but introduces unnecessary complexity.
- **Facial EmotionRecognition with Deepface (manish-9245)**
  - **Popularity (Stars/Forks)**: High
  - **Documentation**: Minimal
  - **Sample Output**: No
  - **Strengths**: Popular, leverages OpenCV and Deepface
  - **Limitations**: No emojis, limited documentation, no sample output
  - **Notes**: Promising for future exploration but not ideal for current project due to complexity.

- **Real-time Facial Emotion Detection**
  - **Popularity (Stars/Forks)**: High
  - **Documentation**: Poor
  - **Sample Output**: No
  - **Strengths**: Focuses on real-time deep learning applications
  - **Limitations**: No emojis, no sample output, complex
  - **Notes**: Valuable learning resource but not directly applicable in current context.
- **Emotion Detection using Face Recognition (prathmesh444)**
  - **Popularity (Stars/Forks)**: Low
  - **Documentation**: Poor
  - **Sample Output**: No
  - **Strengths**: Unsure
  - **Limitations**: Unclear implementation and limited information
  - **Notes**: Provided no significant insights; not considered further.
- **Facial EmotionRecognition Emojifier (vijuSR)**
  - **Popularity (Stars/Forks)**: Moderate
  - **Documentation**: Excellent
  - **Sample Output**: Yes
  - **Strengths**: Well-documented, overlays emojis based on facial emotion
  - **Limitations**: Outdated libraries, limited emotion range
  - **Notes**: Chosen for adaptation due to manageable complexity and alignment with project goals.

1) Uniqueness of Current Implementation: This project updates the chosen repository to modern machine learning libraries and widens the range of detectable emotions. It improves on the existing design by fixing outdated dependencies and emphasizing adaptability to diverse emotional expressions. Learning from the strengths and limitations of other repositories, this project attempts to balance technical feasibility with practical usability to offer an interactive program with clear documentation in a short amount of time.

2) Techniques Utilized:

1) Modernization of Legacy Code: Updated libraries and frameworks to ensure compatibility and usage of the latest technology.
2) Computer Vision Integration: Leveraged OpenCV for facial feature detection and preprocessing.
3) Emotion-to-Emoji Mapping: Improved and expanded the mapping logic for a wider set of emotions.
4) Supporting files to help streamline the execution workflow.

III. Dataset & User Inputs

We created our own dataset of 2,000 images of two faces, classified under 10 expressions, with 200 images of each expression. The expressions include: normal, angry, sunglasses, judging, LOL, shocked, smile, tease, yummy, and kiss. The images for each expression are split into four categories: face 1 with glasses, face 1 with no glasses, face 2 with glasses, and face 2 with no glasses, with the exception being the sunglasses expression. The images are shot under the same lighting.

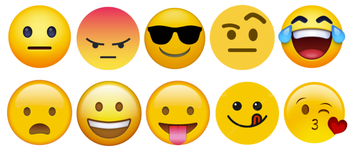

Fig. 1. Emojis used: normal, angry, sunglasses, judging, LOL, shocked, smile, tease, yummy, and kiss (left to right).

To run the program with its dependencies, the user can set up a virtual environment:

- python3 -m venv venv
- .\venv\Scripts\activate

Then, using pip, install the following:

- python3 -m pip install opencv-python
- python3 -m pip install tqdm
- python3 -m pip install tensorflow
- python3 -m pip install numpy

To run the program, use the command:

```
python3 src/predictor.py
```

Two live video streams will pop up, one grayscale and one in color. The color video stream will overlay the detected emoji on top of the face.

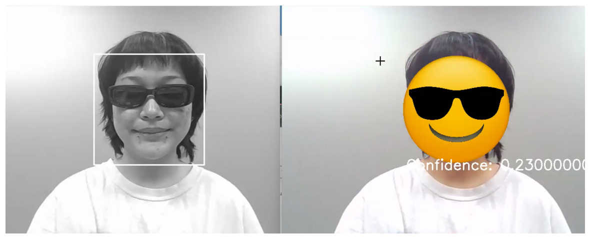

Fig. 2. Screenshot of running program.

IV. Methods

This section provides an in-depth analysis of the key elements in the Emojifier project, how these elements relate to computer vision techniques, and how the project was modernized to be compatible with current APIs and libraries.

The "Emojifier" project leverages several computer vision and deep learning techniques, including:

- Face Detection (via Haar Cascade Classifier)
- Image Preprocessing (grayscale conversion, resizing, normalization)
- Convolutional Neural Networks for emotion recognition
- Real-time Inference for predicting emotions and overlaying emojis on faces.

The project leverages OpenCV for real-time image processing and TensorFlow for training and inference in the emotion recognition model. Many of the methods used directly align with concepts we've explored throughout the course. Notably, feature extraction in facial feature recognition and image preprocessing techniques—such as grayscale conversion, resizing, and normalization—were central to almost every individual and group programming assignment this semester. In Section 1, we will discuss the key concepts and tools from computer vision, including core concepts learned in the course applied in the project. Additionally, the project heavily relies on external resources, including OpenCV and TensorFlow documentation, to modernize legacy code that previously used outdated versions of these libraries. This will be explored further in Section 2.

A. Elements of Computer Vision in the Project

The Emojifier project integrates several well-established techniques from computer vision and deep learning. Below are the main components of computer vision used in the project, with detailed elaboration on each:

1) Face Detection using Haar Cascade Classifier (Feature Extraction):

Concept from Course: In the course, we studied feature extraction as a critical step for detecting and identifying key patterns or regions of interest in images.

How it Applies: In the Emojifier project, the Haar Cascade Classifier is used to detect faces in real-time webcam feeds. The `face_capture.py` script uses OpenCV's `CascadeClassifier` with a pre-trained Haar Cascade to detect faces in the input video feed. This is the key operation for identifying and isolating facial features. The classifier works by analyzing the image for pre-defined Haar-like features, which are patterns of light and dark regions that correspond to facial features such as eyes, nose, and mouth. Once these features are identified, the classifier draws bounding boxes around the faces.

2) Image Preprocessing (Grayscale Conversion, Resizing, Normalization):

Concept from Course: In the course, we learned several image transformations and filters used to enhance image features or prepare them for further processing. These include grayscale conversion, resizing, and normalization. These transformations are often crucial steps in improving the performance of computer vision models by reducing noise, standardizing input sizes, and scaling pixel values.

How it Applies:

- **Grayscale Conversion:** In Emojifier, each video frame is converted to grayscale to simplify face detection. This reduces computational complexity since the model doesn't need to process color information.

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

- **Resizing:** The captured facial images are resized to a fixed 48x48 pixels before being fed into the neural network. This is an example of image transformation, where we standardize input dimensions to match the expected model input. This is similar to the work we have done when writing supporting functions in group programming assignments.

```
face_img_gray = cv2.resize(
    ↪ face_img_gray, (48, 48))
```

- **Normalization:** Pixel values are normalized to the range [0, 1] by dividing by 255. This image normalization improves the training stability.

```
img = gray_img_input.reshape(1, 48,
    ↪ 48, 1).astype(float) / 255
```

3) Convolutional Neural Networks (CNNs) for Emotion Recognition:

Concept from Course: While CNNs differ from traditional computer vision methods like image filtering or feature extraction, we learned about descriptor extraction techniques to see how models learn features from raw data.

How it Applies: The Emojifier project uses a CNN to classify emotions based on facial expressions. The CNN automatically learns features from raw pixel data, identifying patterns such as facial shapes, eye movements, and muscle contractions that represent different emotions. Descriptors are not manually defined; CNNs learn these features directly during training, but they all share the same basis that they are learning patterns from data.

4) Descriptor Matching and Facial Expression Recognition (Emotion Mapping):

Concept from Course: We explored descriptor matching in the course, which involves finding correspondences between key points in images. While the Emojifier project does not explicitly use techniques like SIFT, it indirectly leverages similar concepts when mapping predicted emotions to the corresponding emoji images.

How it Applies: Once an emotion is predicted, the corresponding emoji is selected using an emotion map. This ”descriptor” (the predicted emotion) is matched to an emoji, and the emoji is then overlaid onto the video feed. Although descriptor matching in this context is based on predicted labels rather than image features, it mirrors the basic idea of matching learned features to predefined templates, as we did with object recognition tasks in the course.

B. Modifications and Learnings to Adapt the Old Repo to Current APIs

1) Handling Library Version Conflicts and Dependency Issues:

Learning: The repo's original `requirements.txt` file was incompatible with newer versions of Python and its libraries. This was an important learning as we needed to manually install and configure the correct versions of libraries like OpenCV and TensorFlow.

Modification: We manually installed the required dependencies and updated the `requirements.txt` to include specific versions of libraries that were compatible with Python 3.9/3.10. Fixed issues such as module deprecation warnings, function signature changes, and library conflicts to ensure the code would run on modern libraries.

2) Fixing Compatibility Challenges with TensorFlow:

Learning: The original repository used a legacy version of TensorFlow (v1.x), which used a different method for defining models, handling sessions, and managing placeholders. We had to understand the key differences between TensorFlow 1.x and 2.x, especially in terms of eager execution and model building using the `tf.keras` API.

Modification:

It is worth noting here that we have to make a strategic choice here in terms of how to adapt this old repository to work with our compilers and our environment. One potential strategy for adapting the code and syncing machine learning libraries, compilers, and Python versions involves downgrading our current tools to ensure compatibility with the TensorFlow 1.x code from the original repository. However, this approach contradicts the principle of using the latest tools with enhanced features, as old tools are more likely to be unsupported in current and future code platforms. Additionally, it is not sustainable, as future developers may encounter difficulties with outdated, incompatible tools, which were the exact problems that we are currently facing and trying to solve. Therefore, the most practical solution is to update the source code to work with the latest libraries, which serves as the foundation for our modernizing of the old repository. Some of our main modifications include:

- **Rewriting the Model:** The old model was built using TensorFlow's lower-level API. We rewrote the model using the high-level `tf.keras` API, which is now the standard for building neural networks in TensorFlow 2.x.
- **Training and Inference:** Updated training loops and the model inference process to work with TensorFlow 2.x. This included replacing the manual session handling (from v1.x) with the more intuitive `model.fit()` and `model.evaluate()` methods.

Example Analysis of the Transition from TensorFlow 1.x to TensorFlow 2.x in the `predictor.py` Script

The transition from TensorFlow 1.x to TensorFlow 2.x involves several core changes, particularly around execution models, API structure, and layer management. Below is a detailed breakdown of how the original `predictor.py` script has been adapted to work with TensorFlow 2.

1. Removal of Session-Based Execution (`tf.Session`)

- **TensorFlow 1.x:** In TensorFlow 1.x, the code used to create a session (`tf.Session`) and execute the graph explicitly. The session had to be managed manually, and operations like predictions were done using `sess.run()`.

```
config = tf.ConfigProto()
config.gpu_options.allow_growth =
    ↪ True
with tf.Session(config=config) as
    ↪ sess:
        saver.restore(sess, os.path.join(
            ↪ CHECKPOINT_SAVE_PATH, '
            ↪ model.ckpt'))
        y_c = sess.run(y_conv, feed_dict
            ↪ ={X: img, keep_prob: 1.0})
```

- **TensorFlow 2.x:** TensorFlow 2.x uses eager execution by default, meaning that computations are executed immediately without needing a session. The `tf.Session` is no longer needed, and operations are evaluated as they are called.

```
model = tf.keras.models.load_model(os
    ↪ .path.join(CHECKPOINT_SAVE_PATH
    ↪ , 'model.h5'))
y_c = model.predict(img)
```

- **Changes:**
  - Session Removal: The code removes `tf.Session` and the config setup for GPU management (`tf.ConfigProto`).
  - Model Inference: Instead of using `sess.run()` to fetch predictions, `model.predict()` is used. This is more direct and aligns with the Keras API used in TensorFlow 2.x.

2. Placeholders (`tf.placeholder`) to Direct Model Input

- **TensorFlow 1.x:** In TensorFlow 1.x, placeholders (`tf.placeholder`) were used to define the input tensors. Data had to be fed into the placeholders using `feed_dict`.

```
X = tf.placeholder(tf.float32, shape
    ↪ =[None, 48, 48, 1])
y_conv = model(X, keep_prob)
y_c = sess.run(y_conv, feed_dict={X:
    ↪ img, keep_prob: 1.0})
```

- **TensorFlow 2.x:** In TensorFlow 2.x, placeholders are no longer required. Input data is passed directly into the model, and the data is processed immediately due to eager execution.

```
y_c = model.predict(img)   # Model
    ↪ inference is now direct without
    ↪  placeholders
```

- **Changes:**
  - Input Handling: The need for `tf.placeholder` is eliminated, and data is passed directly to the model using `model.predict()`.
  - Feed Dictionary Removal: No need for the `feed_dict` parameter used in TensorFlow 1.x. The data is passed to the model as an argument directly.

3. Model Definition and Construction

- **TensorFlow 1.x:** In the original TensorFlow 1.x code, the model was built manually, defining operations such as convolution layers and dropout using functions like `conv_layer`, `full_layer`, etc.

```
y_conv = model(X, keep_prob)
```

- **TensorFlow 2.x:** TensorFlow 2.x uses the Keras API to build and manage models. In the updated code, `tf.keras.Sequential` or the functional API is used to define models. Layers such as `Conv2D`, `Dense`, and `Dropout` are now managed by Keras.

```
model = tf.keras.models.load_model(os
    ↪ .path.join(CHECKPOINT_SAVE_PATH
    ↪ , 'model.h5'))
```

- **Changes:**
  - Model Loading: Instead of manually defining the layers and their relationships, the updated script loads a pre-trained model directly with `tf.keras.models.load_model()`.
  - Layer Management: The Keras model manages layers such as convolutions and dense layers internally.

4. Model Saving and Loading

- **TensorFlow 1.x:** In TensorFlow 1.x, the model was saved using `tf.train.Saver`, and the session's state was saved/loaded using `sess.run()` and `saver.restore()`.

```
saver = tf.train.Saver()
with tf.Session(config=config) as
    ↪ sess:
    saver.restore(sess, os.path.join(
        ↪ CHECKPOINT_SAVE_PATH, '
        ↪ model.ckpt'))
```

- **TensorFlow 2.x:** In TensorFlow 2.x, the Keras API provides a simplified approach to save and load models using `model.save()` and `tf.keras.models.load_model()`.

```
model = tf.keras.models.load_model(os
    ↪ .path.join(CHECKPOINT_SAVE_PATH
    ↪ , 'model.h5'))
```

- **Changes:**
  - Model Saving/Loading: `tf.train.Saver` has been replaced by `model.save()` and `tf.keras.models.load_model()`. This is much simpler and more integrated into the Keras workflow.

5. Dropout Layer Handling

- **TensorFlow 1.x:** In the original script, dropout was applied manually using `tf.nn.dropout`.

```
y_conv = tf.nn.dropout(y_conv,
    ↪ keep_prob)
```

- **TensorFlow 2.x:** The Dropout layer is now part of the Keras API and is used directly during the model definition.

```
from tensorflow.keras.layers import
    ↪ Dropout
model.add(Dropout(0.5))
```

- **Changes:** Dropout is now part of the Keras model architecture rather than being manually applied during training.

Summary of Key Changes:

- **Session Management:** No more `tf.Session`; TensorFlow 2.x uses eager execution and manages the computation graph automatically.
- **Placeholders:** Replaced with direct model input.
- **Model Definition:** The Keras API (`tf.keras.models`) replaces the custom model-building process.
- **Model Saving/Loading:** Use `model.save()` and `tf.keras.models.load_model()` instead of `tf.train.Saver()`.
- **Dropout:** Managed via Keras Dropout layer instead of manual application.

By making these changes, the updated script aligns with the TensorFlow 2.x and Keras paradigm, simplifying the code and enhancing compatibility with modern TensorFlow workflows.

V. Results & Evaluation

We evaluated our program by testing it on 100 total instances of facial expressions, with ten instances of each emoji (e.g., ten angry expressions, ten judging expressions, etc.). However, we only tested this with one face and one lighting, so our results do not take into account potential variations in facial features and light.

Confidence scores for all of the detected facial expressions were low and ranged from 0.2-0.4.

Accuracy was assessed by making a facial expression and recording the emoji assigned after three seconds. If the prediction is correct and is able to be held for three more seconds, then it is noted as correct.

Seven out of ten emojis were assigned with an accuracy of 80% or higher. The remaining three emojis were normal, judging, and yummy. The normal expression was rarely detected, and had an accuracy of only 20%. On the other hand, while the judging expression was always detected, it also had the issue of a large amount of false positives, with normal or neutral expressions often being detected as judging. The yummy expression was able to be detected within three seconds often, however the detected expression would not hold for very long and would also detect other expressions, namely tease, whose expression also includes sticking out the tongue.

We suspect that the dataset of judging and normal are too similar, as the only difference between the two expressions is the positioning of the eyebrows. The yummy expression has a similar issue with the tease expression. In fact, previous iterations of our program had included more emojis, however due to overlapping expressions, we removed them. For example, we originally included a winky face emoji, but due to its similarities with the kiss emoji, its detection did not perform well enough.

We believe that the overall low confidence scores in facial expression classification can be attributed to our small and uniform dataset. In a previous iteration, when trained on 2,000 images of a single face, testing it on that same face yielded higher confidence scores, as opposed to our final model, which was trained on 2,000 images of two different faces.

Expression	Number of correct predictions
Angry	10
Sunglasses	10
Judging	10
Kiss	8
LOL	9
Normal	2
Shocked	10
Smile	7
Tease	8
Yummy	4

Overall Model Evaluation	
Accuracy	0.79
Precision	0.796
Recall	0.78

VI. Conclusion

This project examines the role of AI in detecting emotion, and subsequently fostering empathy and understanding in digital interactions.

While the Emojifier system successfully detects and maps emotions to emojis with moderate accuracy, challenges such as dataset variation and volume and overlapping expressions leave room for future work. Future work should focus on diversifying datasets and expanding adaptability to more nuanced expressions across diverse demographics.

Applications of this research can extend to video conferencing platforms, expression analysis in fields like customer service, and improving AI emotional intelligence.

Here is a test citation [**?**]. Look at the references bib [**?**] to add new citations as well! Inset bibtex entries there, and automatically cite them here using the `cite` command.

References

[1] ”OpenCV Documentation,” Available: `https://docs.opencv.org/4.x/index.html`.

[2] ”OpenCV Cascade Classifier Tutorial,” Available: `https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html`.

[3] ”OpenCV Python Tutorial,” GeeksforGeeks. Available: `https://www.geeksforgeeks.org/opencv-python-tutorial/#-22-image-processing`. [Includes: `https://www.geeksforgeeks.org/image-resizing-using-opencv-python/`].

[4] ”Load and preprocess images,” TensorFlow. Available: `https://www.tensorflow.org/tutorials/load_data/images`.

[5] ”Convolutional Neural Networks (CNN),” TensorFlow. Available: `https://www.tensorflow.org/tutorials/images/cnn`.

[6] ”Image classification,” TensorFlow. Available: `https://www.tensorflow.org/tutorials/images/classification`.

[7] ”Keras model predict for a single image,” Stack Overflow. Available: `https://stackoverflow.com/questions/43017017/keras-model-predict-for-a-single-image`.

[8] ”tf.keras.layers.Conv2D,” TensorFlow. Available: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D`.

[9] ”tf.keras.layers.MaxPool2D,” TensorFlow. Available: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D`.

[10] ”tf.keras.layers.Dropout,” TensorFlow. Available: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout`.

[11] ”Custom training walkthrough,” TensorFlow. Available: `https://www.tensorflow.org/tutorials/customization/custom_training_walkthrough`.

[12] ”Face Detection in Python using a Webcam,” Real Python. Available: `https://realpython.com/face-detection-in-python-using-a-webcam/`.

[13] ”Softmax Activation,” Pinecone. Available: `https://www.pinecone.io/learn/softmax-activation/`.