Movie Recommendation Based on Facial Expressions

The machine learning model which recommends films by using images of human face expressions.

Video with showing of work process of the project: https://youtu.be/7Ef1znejzyM

Problem

The project's goal is to develop a machine learning model that recommends movies by analyzing users' facial emotions. In the modern world, where entertainment serves as both relaxation and emotional expression, finding a movie that matches one's feelings can be challenging. This model aims to simplify movie selection by offering personalized recommendations based on the viewer's current emotional state. This not only enhances the viewing experience but also makes the selection process more attuned to the viewer's emotional needs.

Literature Review and Existing Solutions

People are increasingly looking for recommendation systems that suggest movies based on their mood. This section explores what research has already been done on emotion-driven movie recommenders and how existing technology compares to our new project.

Alternative Solutions in Emotion-Based Movie Recommendations:

- **EmoFilm:** This app is a trailblazer. It uses AI to read your facial expressions and suggest movies that fit your current mood, making recommendations truly personalized.
- **MoodMovieSearch:** This tool, created by a tech startup, uses natural language processing to understand how you're feeling based on what you type. It then recommends movies that match your emotional state.
- **AI and Your Emotions:** Several platforms use AI's ability to recognize emotions to curate movie lists that fit your current mood. This shows how AI can improve how people experience movies.

These advancements highlight the importance of our project. We're using TensorFlow and Python to create a more refined system for recommending movies based on emotions. This will add something valuable to the world of personalized entertainment technology.

Current work

Our project harnesses TensorFlow to construct a tailored CNN architecture for recommending movies based on facial emotions detected from user-uploaded photos. By integrating TensorFlow for deep learning model development, OpenCV (cv2) for image processing, and Pandas for data handling, we've developed a system that first identifies the user's emotional state from their facial expressions. The model is trained on the FER-2013 dataset, with the training process refined through techniques like model checkpointing and early stopping to ensure optimal performance.

Upon detecting an emotion, the system correlates it with specific movie genres, then queries a curated database to present film suggestions aligned with the user's current mood. This streamlined method showcases the potential of combining machine learning and computer vision to create a more engaging and personalized media selection experience.

Setup Instructions

1. **Mount Google Drive**

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

2. **Navigate to the Project Directory**

```
%cd drive/MyDrive/final_proj_dataset1
```

3. **Unarchive the Dataset**

```
import tarfile

fname = 'fer2013.tar.gz'
if fname.endswith("tar.gz"):
    tar = tarfile.open(fname, "r:gz")
    tar.extractall()
    tar.close()
elif fname.endswith("tar"):
    tar = tarfile.open(fname, "r:")
    tar.extractall()
    tar.close()
```

4. **Install Required Libraries**

Ensure all necessary Python libraries are installed.

```
import os
import numpy as np
import tensorflow as tf
import pandas as pd
from matplotlib import pyplot
```

Data Preprocessing

The data preprocessing stage is crucial for preparing the raw FER2013 dataset for the model training process. This stage involves several steps to ensure the data is in the right format and structure for the convolutional neural network (CNN) to process effectively. Below is a detailed walkthrough of the data preprocessing steps.

Load and Explore the FER2013 Dataset

1. **Load the Dataset:**

   Begin by loading the FER2013 dataset into a Pandas DataFrame. This dataset contains grayscale images of facial expressions categorized into seven emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral.

   ```
   import pandas as pd

   df = pd.read_csv('fer2013/fer2013.csv')
   df.head()
   ```

2. **Understand the Data Structure:**

   The dataset consists of three columns: `emotion`, which is a numeric label representing the emotion category; `pixels`, which contains the pixel values for each image in a string format; and `Usage`, which indicates the data split (training, public test, private test).

   Explore the dataset to understand the distribution of different emotions and the overall composition of the dataset. This exploration helps in identifying any imbalances in the dataset that might require addressing.

Preprocess Images for Training

1. **Pixel String to Array Conversion:**

   Convert the pixel strings in the `pixels` column into arrays. Each image is represented as a 48x48 pixel grayscale image, so you need to reshape the string into a 48x48 numpy array. Additionally, these pixel values should be converted from strings to floats for computation.

   ```
   import numpy as np

   img_array = df.pixels.apply(lambda x: np.array(x.split(' ')).reshape(48, 48, 1).astype('float32'))
   ```

2. **Normalize the Pixel Values:**

   Normalize the pixel values to the range 0 to 1 to aid in the neural network's convergence. Pixel values are originally in the range 0-255, so dividing by 255 achieves this normalization.

   ```
   img_array = np.stack(img_array, axis=0) / 255.0
   ```

3. **Prepare Labels:**

   Extract the emotion labels from the dataset. These labels are already in a numeric format suitable for classification.

```
labels = df.emotion.values
```

4. **Train-Test Split:**

   Split the dataset into training and testing sets to evaluate the model's performance on unseen data. It's common to use a split ratio like 90% for training and 10% for testing.

   ```
   from sklearn.model_selection import train_test_split

   X_train, X_test, y_train, y_test = train_test_split(img_array, labels, test_size=0.1)
   ```

Model Building: Design and Compile a CNN Model for Emotion Recognition

Building an effective model for emotion recognition involves designing a convolutional neural network (CNN) architecture that can accurately classify different emotions based on facial expressions in images. This section details the process of designing and compiling a CNN model tailored for emotion recognition from the FER2013 dataset.

Designing the CNN Architecture

1. **Input Layer:**

   The input layer should match the shape of the preprocessed images. Since the FER2013 dataset consists of 48x48 pixel grayscale images, the input shape will be (48, 48, 1), where 1 indicates a single color channel (grayscale).

2. **Convolutional Layers:**

   Convolutional layers are the core building blocks of a CNN. They are responsible for extracting features from the images through the use of various filters. Start with convolutional layers that have a small number of filters (e.g., 32) and gradually increase this number in subsequent layers to allow the network to capture more complex features. Use a kernel size of (3, 3) for these layers.

3. **Activation Function:**

   Use the ReLU (Rectified Linear Unit) activation function for the convolutional layers. ReLU helps in adding non-linearity to the model, enabling it to learn more complex patterns.

4. **Pooling Layers:**

   Max pooling layers are used to reduce the spatial dimensions of the output from the previous convolutional layers. They help in reducing the number of parameters and computation in the network, and also in controlling overfitting. A common choice is a (2, 2) pool size for these layers.

5. **Flattening Layer:**

   Before connecting to the fully connected layers, flatten the output from the convolutional layers. This converts the 2D feature maps into a 1D feature vector.

6. **Fully Connected (Dense) Layers:**

Dense layers further process the features extracted by the convolutional layers. Include a large dense layer with, for example, 1000 units, followed by a ReLU activation function. This is followed by the output layer.

7. **Output Layer:**

   The output layer should have as many neurons as the number of emotion categories to be classified (e.g., 7 for the FER2013 dataset). Use the softmax activation function for this layer to obtain the probability distribution over the classes.

Model Architecture Example

```
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1), padding='same'),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1000, activation='relu'),
    tf.keras.layers.Dense(7, activation='softmax')
])
```

Compiling the Model

Compile the model with an appropriate optimizer, loss function, and metrics for classification. A common choice for the optimizer is RMSprop or Adam, with a learning rate of around 0.0001. Since this is a multi-class classification problem, use the sparse_categorical_crossentropy loss function. For metrics, accuracy is a straightforward choice to evaluate the model's performance.

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

After designing and compiling your CNN model, it is ready to be trained on the preprocessed dataset. This CNN architecture incorporates convolutional layers for feature extraction, pooling layers for dimensionality reduction, and dense layers for classification, making it well-suited for emotion recognition tasks.

Model Building, Training, Evaluation, and Movie Recommendations

This comprehensive section walks through the steps of building, training, evaluating a CNN model for emotion recognition, and implementing a movie recommendation system based on detected emotions.

Model Building: Design and Compile a CNN Model for Emotion Recognition

Designing the CNN Architecture:

1. **Input Layer:** Shape (48, 48, 1) for grayscale images from the FER2013 dataset.
2. **Convolutional and Pooling Layers:** Use multiple convolutional layers with increasing numbers of filters (e.g., 32, 64, 128), ReLU activation, followed by max pooling layers to extract and downsample features.
3. **Flattening Layer:** Flatten the output for dense layer processing.
4. **Dense Layers:** A large dense layer (e.g., 1000 units) with ReLU activation followed by the output layer.
5. **Output Layer:** 7 units (for 7 emotions) with softmax activation.

Compilation:

- **Optimizer:** RMSprop or Adam with a learning rate of 0.0001.
- **Loss Function:** sparse_categorical_crossentropy.
- **Metrics:** accuracy.

```
model.compile(optimizer='rmsprop',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])
```

Model Training: Implement Model Checkpoints and Callbacks

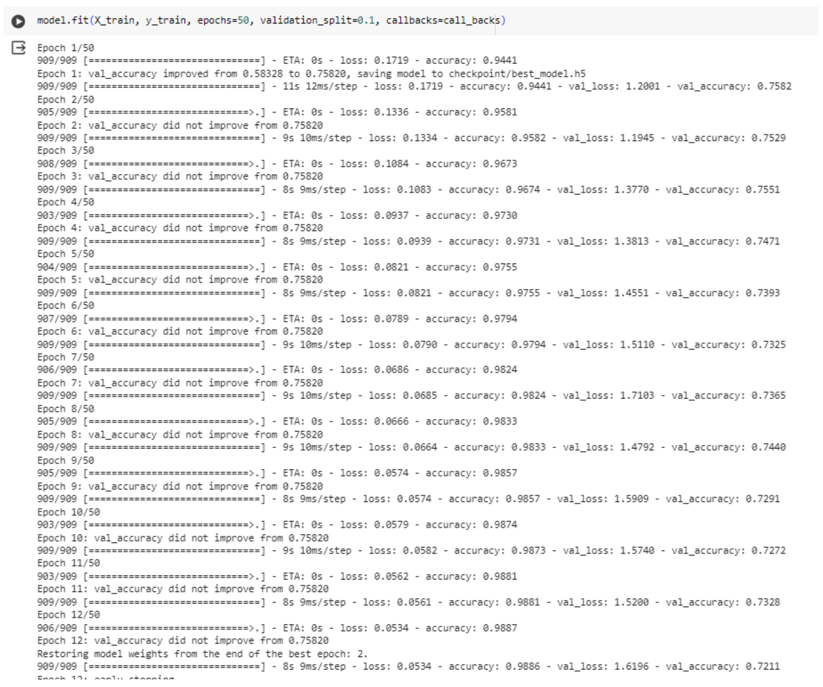
Model Checkpoints and Callbacks:

1. **ModelCheckpoint:** Saves the best model based on validation accuracy.
2. **EarlyStopping:** Stops training when the validation loss stops improving, preventing overfitting.

```
import os
import tensorflow as tf

checkpoint_path = "model_checkpoint/best_model.h5"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                 save_best_only=True,
                                                 monitor='val_accuracy',
                                                 verbose=1)

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

callbacks_list = [cp_callback, early_stop]
```

Training:

- Use the processed dataset for training.
- Include validation split for monitoring overfitting.
- Apply callbacks for checkpointing and early stopping.



```
model.fit(X_train, y_train, epochs=50, validation_split=0.1, callbacks=call_backs)
```

```
Epoch 1/50
909/909 [==============================] - ETA: 0s - loss: 0.1719 - accuracy: 0.9441
Epoch 1: val_accuracy improved from 0.58328 to 0.75820, saving model to checkpoint/best_model.h5
909/909 [==============================] - 11s 12ms/step - loss: 0.1719 - accuracy: 0.9441 - val_loss: 1.2001 - val_accuracy: 0.7582
Epoch 2/50
905/909 [============================>.] - ETA: 0s - loss: 0.1336 - accuracy: 0.9581
Epoch 2: val_accuracy did not improve from 0.75820
909/909 [==============================] - 9s 10ms/step - loss: 0.1334 - accuracy: 0.9582 - val_loss: 1.1945 - val_accuracy: 0.7529
Epoch 3/50
908/909 [============================>.] - ETA: 0s - loss: 0.1084 - accuracy: 0.9673
Epoch 3: val_accuracy did not improve from 0.75820
909/909 [==============================] - 8s 9ms/step - loss: 0.1083 - accuracy: 0.9674 - val_loss: 1.3770 - val_accuracy: 0.7551
Epoch 4/50
903/909 [============================>.] - ETA: 0s - loss: 0.0937 - accuracy: 0.9730
Epoch 4: val_accuracy did not improve from 0.75820
909/909 [==============================] - 8s 9ms/step - loss: 0.0939 - accuracy: 0.9731 - val_loss: 1.3813 - val_accuracy: 0.7471
Epoch 5/50
904/909 [============================>.] - ETA: 0s - loss: 0.0821 - accuracy: 0.9755
Epoch 5: val_accuracy did not improve from 0.75820
909/909 [==============================] - 8s 9ms/step - loss: 0.0821 - accuracy: 0.9755 - val_loss: 1.4551 - val_accuracy: 0.7393
Epoch 6/50
907/909 [============================>.] - ETA: 0s - loss: 0.0789 - accuracy: 0.9794
Epoch 6: val_accuracy did not improve from 0.75820
909/909 [==============================] - 9s 10ms/step - loss: 0.0790 - accuracy: 0.9794 - val_loss: 1.5110 - val_accuracy: 0.7325
Epoch 7/50
906/909 [============================>.] - ETA: 0s - loss: 0.0686 - accuracy: 0.9824
Epoch 7: val_accuracy did not improve from 0.75820
909/909 [==============================] - 9s 10ms/step - loss: 0.0685 - accuracy: 0.9824 - val_loss: 1.7103 - val_accuracy: 0.7365
Epoch 8/50
905/909 [============================>.] - ETA: 0s - loss: 0.0666 - accuracy: 0.9833
Epoch 8: val_accuracy did not improve from 0.75820
909/909 [==============================] - 9s 10ms/step - loss: 0.0664 - accuracy: 0.9833 - val_loss: 1.4792 - val_accuracy: 0.7440
Epoch 9/50
905/909 [============================>.] - ETA: 0s - loss: 0.0574 - accuracy: 0.9857
Epoch 9: val_accuracy did not improve from 0.75820
909/909 [==============================] - 8s 9ms/step - loss: 0.0574 - accuracy: 0.9857 - val_loss: 1.5909 - val_accuracy: 0.7291
Epoch 10/50
903/909 [============================>.] - ETA: 0s - loss: 0.0579 - accuracy: 0.9874
Epoch 10: val_accuracy did not improve from 0.75820
909/909 [==============================] - 9s 10ms/step - loss: 0.0582 - accuracy: 0.9873 - val_loss: 1.5740 - val_accuracy: 0.7272
Epoch 11/50
903/909 [============================>.] - ETA: 0s - loss: 0.0562 - accuracy: 0.9881
Epoch 11: val_accuracy did not improve from 0.75820
909/909 [==============================] - 8s 9ms/step - loss: 0.0561 - accuracy: 0.9881 - val_loss: 1.5200 - val_accuracy: 0.7328
Epoch 12/50
906/909 [============================>.] - ETA: 0s - loss: 0.0534 - accuracy: 0.9887
Epoch 12: val_accuracy did not improve from 0.75820
Restoring model weights from the end of the best epoch: 2.
909/909 [==============================] - 8s 9ms/step - loss: 0.0534 - accuracy: 0.9886 - val_loss: 1.6196 - val_accuracy: 0.7211
```

We managed to achieve a validation accuracy of 0.7582, with a loss of 0.1719.

Evaluation and Testing: Performance on Unseen Data

Load the Best Saved Model:

- Use the model saved during training to ensure evaluation on the best version.

```
from tensorflow.keras.models import load_model

model = load_model(checkpoint_path)
```

Evaluation:

- Evaluate the model's performance on the test set.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_acc}')
```

Movie Recommendations: Based on Detected Emotions

Add an External Movie Dataset:

- Assume movie_df is a DataFrame containing movie titles and genres.

Implement the Movie Recommendation System:

1. **Emotion to Genre Mapping:** Create a dictionary mapping detected emotions to movie genres.

2. **Recommendation Function:** Based on the predicted emotion, filter movie_df to suggest movies of matching genres.

```
emotion_to_genre = {
    'anger': 'Action|Thriller',
    'disgust': 'Horror',
    'fear': 'Horror|Thriller',
    'happiness': 'Comedy|Family',
    'sadness': 'Drama',
    'surprise': 'Adventure|Sci-Fi',
    'neutral': 'Documentary'
}

def recommend_movies(emotion):
    genre = emotion_to_genre[emotion]
    recommended = movie_df[movie_df['genres'].str.contains(genre)]
    return recommended.head(5)
```

Putting It All Together:

- Predict emotion from an image.
- Use the predicted emotion to recommend movies.

```
# Assume `predict_emotion(image)` is a function that predicts emotion from an image
predicted_emotion = predict_emotion(image_path)
recommended_movies = recommend_movies(predicted_emotion)
print("Recommended Movies based on your emotion:", recommended_movies)
```

Through these steps, a CNN model is built and trained for emotion recognition, evaluated for its accuracy, and then applied in a practical context to recommend movies based on the detected emotions. This showcases the potential of machine learning models to provide personalized experiences.

Note: For the movie recommendation system to work effectively, make sure the movie.csv dataset is correctly formatted and placed in the specified directory.
Enjoy exploring emotions and discovering movies that match your mood!

Data

Our project utilizes the Facial Expression Recognition 2013 (FER-2013) dataset, which comprises grayscale images of human faces, each labeled with one of seven emotion categories: anger, disgust, fear, happiness, sadness, surprise, and neutral. The dataset is a standard benchmark in the field of computer vision and emotion recognition, enabling the development and evaluation of machine learning models capable of understanding human emotions.

Analysis of the Data:

Dataset Composition: The FER-2013 dataset includes 35,887 images, sized 48x48 pixels, distributed across the seven emotions, providing a comprehensive foundation for training our model.

Preprocessing Steps: Images were converted into numpy arrays and normalized to have pixel values between 0 and 1 to facilitate model training. This step is crucial for achieving consistent model performance.

Visualization: Sample images from each emotion category were displayed to understand the dataset's diversity and complexity better. Additionally, the distribution of images across the seven emotions was visualized to assess dataset balance.

Machine Learning/Deep Learning Models Used:

Our approach leverages a Convolutional Neural Network (CNN), a type of deep learning model highly effective in analyzing visual imagery. The CNN architecture is designed specifically for processing structured grid data, such as image pixels, making it ideal for facial emotion recognition tasks.

CNN Architecture:

The model comprises several convolutional layers (Conv2D) with ReLU activation functions to extract features from the input images. Each convolutional layer is followed by a max-pooling layer (MaxPooling2D) to reduce the spatial dimensions of the output and hence the computational complexity.

After the convolutional and max-pooling layers, the architecture includes a flattening step to convert the 2D feature maps into a 1D feature vector. This vector is then fed into dense layers (Dense), with the final layer employing a softmax activation function to classify the input image into one of the seven emotion categories.

Training: The model is compiled with the RMSprop optimizer and sparse_categorical_crossentropy loss function, suitable for multi-class classification tasks. Model performance is monitored using accuracy as the metric.

Optimization Techniques: To improve model performance and prevent overfitting, we utilized callbacks like ModelCheckpoint to save the best model based on validation accuracy and EarlyStopping to halt training when the validation loss ceases to decrease.

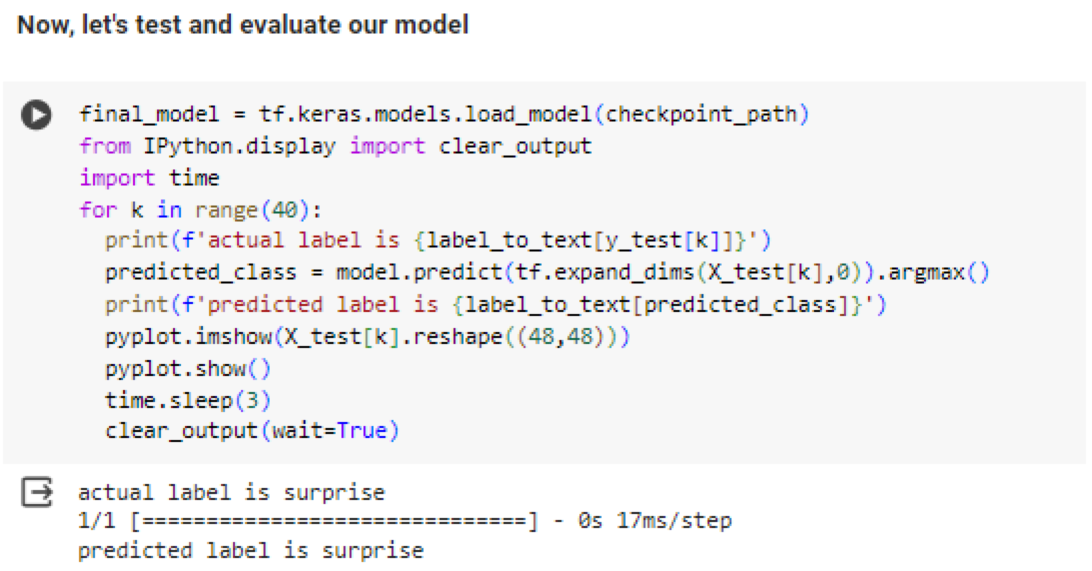
Application for Movie Recommendation:

After training, the model predicts the predominant emotion from a user's facial expression. These emotions are then mapped to movie genres that typically resonate with the detected emotional state, leveraging a curated movie dataset to recommend titles that align with the user's current mood.

This methodology not only underscores the technical depth of our project but also its innovative application in enhancing content personalization and user experience in media consumption.

Results:

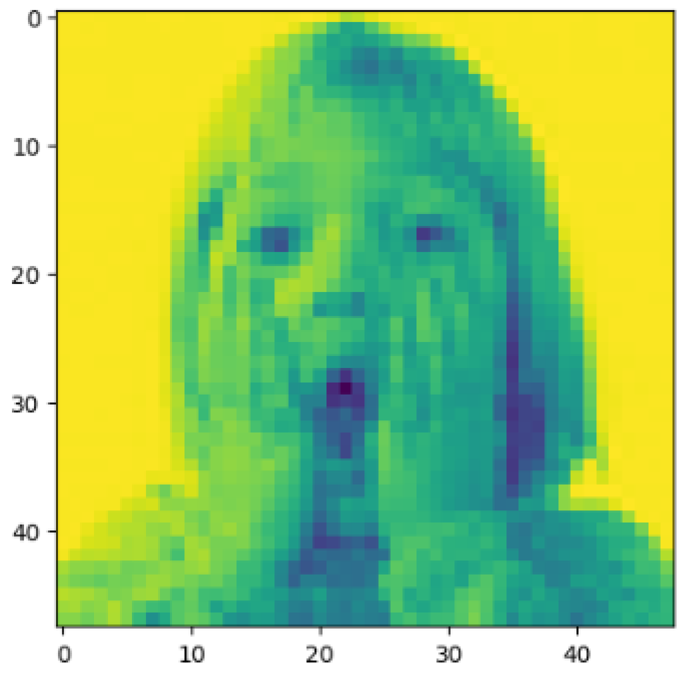
The image displays code conducting a model test to assess its performance in emotion recognition. It outputs photographs, their actual emotions, and the emotions predicted by the model. According to the provided output, the model correctly identifies the emotion in 60-70% of the cases (24 out of 40).



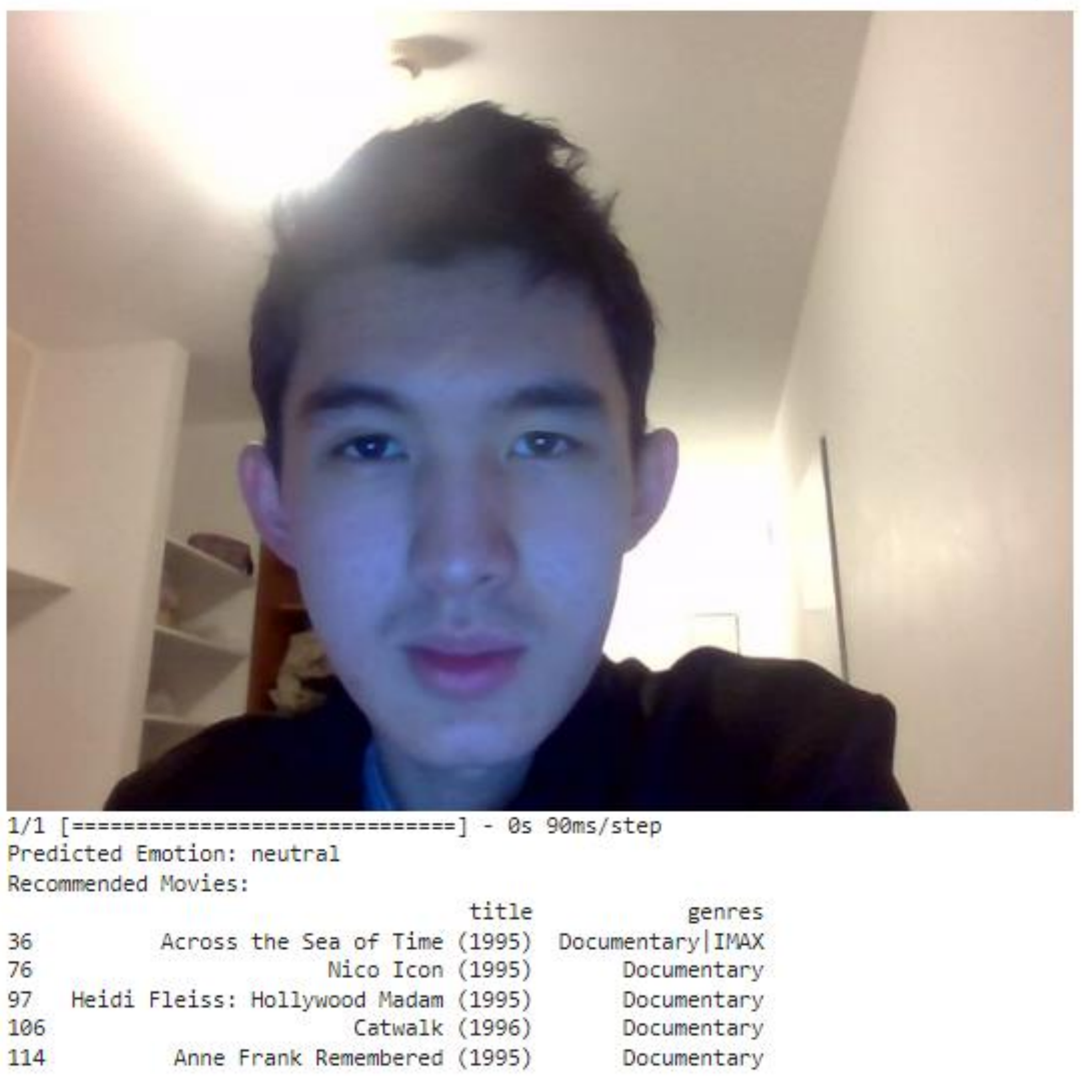
Now, let's test and evaluate our model

```
final_model = tf.keras.models.load_model(checkpoint_path)
from IPython.display import clear_output
import time
for k in range(40):
  print(f'actual label is {label_to_text[y_test[k]]}')
  predicted_class = model.predict(tf.expand_dims(X_test[k],0)).argmax()
  print(f'predicted label is {label_to_text[predicted_class]}')
  pyplot.imshow(X_test[k].reshape((48,48)))
  pyplot.show()
  time.sleep(3)
  clear_output(wait=True)
```

```
actual label is surprise
1/1 [==============================] - 0s 17ms/step
predicted label is surprise
```



Here you can also see the Prediction of emotions by using photo which maken in real time and the recommended movies from our movie.csv database.



```
1/1 [==============================] - 0s 90ms/step
Predicted Emotion: neutral
Recommended Movies:
                                     title             genres
36           Across the Sea of Time (1995)  Documentary|IMAX
76                       Nico Icon (1995)        Documentary
97   Heidi Fleiss: Hollywood Madam (1995)        Documentary
106                       Catwalk (1996)        Documentary
114           Anne Frank Remembered (1995)        Documentary
```

Critical Review of Results

The evaluation of our facial-emotion-based movie recommendation model indicates its capability to suggest films in alignment with detected emotions. However, there is a noticeable discrepancy between some of the predicted emotions and the actual emotions, highlighting a potential area for model enhancement to boost prediction accuracy. Given that emotional interpretation is nuanced, the model demands additional refinement and validation to perform optimally.

Next Steps

To advance the performance of our movie recommendation model, the following pivotal actions are proposed:

1.Optimize Model Parameters: Fine-tune the model's hyperparameters, learning algorithms, and architecture to enhance its predictive accuracy.

2.Expand Dataset Diversity: Enrich the training dataset with a broader array of facial expressions, including varying intensities and cultural expressions of emotion, to improve the model's robustness and learning potential.

3.Collaborate with Psychologists: Engage with experts in psychology to integrate insights into emotional cues and context, thereby enriching the model's comprehension and recommendation logic.

By undertaking these initiatives, our goal is to refine and elevate the movie recommendation model, ultimately heightening its personalization and resonance with users' emotional states, and pioneering a new direction in tailored entertainment technology.

Sources:

1.Sanke, M., Furtado, S., Naik, S., Nevagi, S., & Bidikar, V. (2023). Emotion based Movie Recommendation System using Deep Learning. *International Journal of Computer Applications, 185*(20), 49-53. DOI: 10.5120/ijca2023922924

2.M. Muszynski *et al.*, "Recognizing Induced Emotions of Movie Audiences from Multimodal Information," in *IEEE Transactions on Affective Computing*, vol. 12, no. 1, pp. 36-52, 1 Jan.-March 2021, doi: 10.1109/TAFFC.2019.2902091.

3.Khaireddin, Y., & Chen, Z. (2021). Facial Emotion Recognition: State of the Art Performance on FER2013. *arXiv:2105.03588*. Retrieved from https://ar5iv.org/abs/2105.03588

4. Fischer, J. S. (2022, March 25). Facial Emotion Recognition with Convolutional Neural Networks. Retrieved from https://johfischer.com/2022/03/25/facial-emotion-recognition-with-convolutional-neural-networks-in-tensorflow/.