1. Import needed libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.layers import Dense, ReLU, Embedding, BatchNormalization, Concatenate, Conv1D, Gl
from tensorflow.keras.models import Model
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
import os
from google.colab import files, drive
```

2. Read and Preprocess data

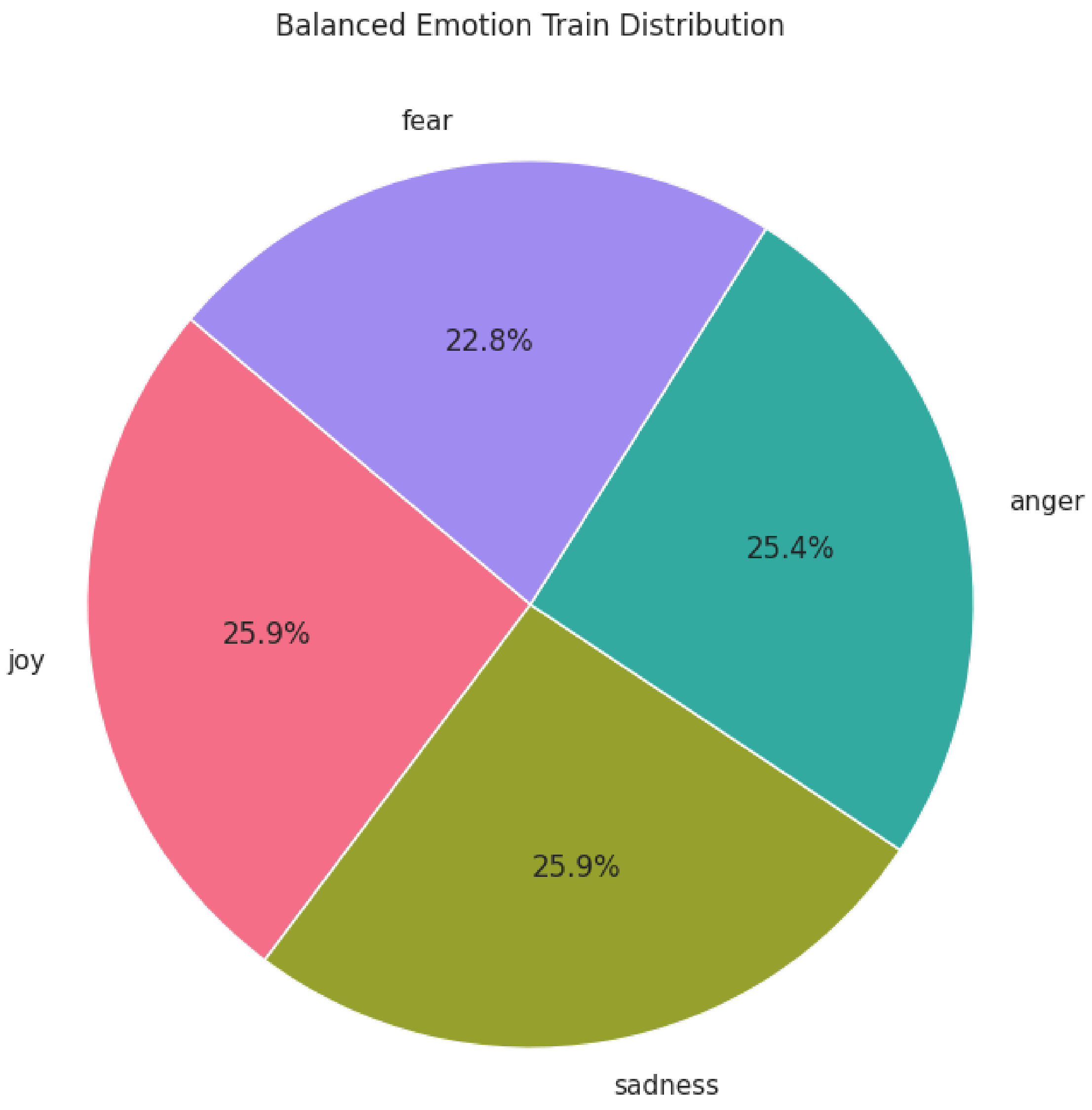
2.1 Read data

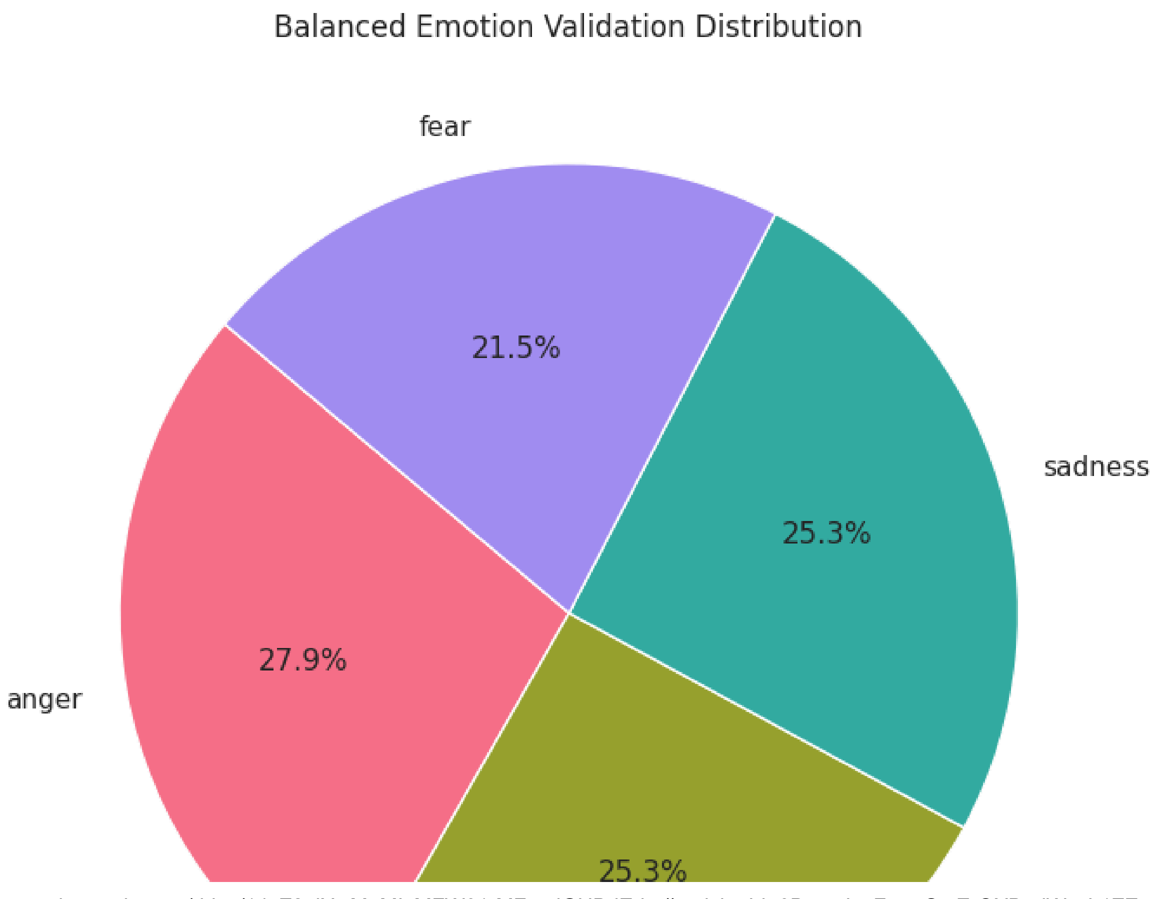
```
# Install Kaggle
!pip install kaggle

# Mount Google Drive
drive.mount('/content/drive')

# Define paths
drive_kaggle_path = "/content/drive/MyDrive/Colab Notebooks/kaggle.json"
kaggle_dir = "/root/.kaggle/"

# Check for kaggle.json
if os.path.exists(drive_kaggle_path):
    print(f"Found kaggle.json in {drive_kaggle_path}")
    !mkdir -p {kaggle_dir}
    !cp "{drive_kaggle_path}" {kaggle_dir}
    !chmod 600 {kaggle_dir}/kaggle.json
else:
    print("kaggle.json not found in Google Drive. Please upload it.")
    uploaded = files.upload()
    if 'kaggle.json' not in uploaded:
        raise FileNotFoundError("You did not upload kaggle.json.")
    !mkdir -p {kaggle_dir}
    !mv kaggle.json {kaggle_dir}
    !chmod 600 {kaggle_dir}/kaggle.json

# Verify kaggle.json
if not os.path.exists(f"{kaggle_dir}/kaggle.json"):
    raise FileNotFoundError(f"Failed to set up kaggle.json in {kaggle_dir}")

# Download and unzip dataset
!kaggle datasets download -d praveengovi/emotions-dataset-for-nlp
!unzip -o emotions-dataset-for-nlp.zip -d emotions_dataset
```

```
# Load data
train_path = "/content/emotions_dataset/train.txt"
val_path = "/content/emotions_dataset/val.txt"
test_path = "/content/emotions_dataset/test.txt"

try:
    df = pd.read_csv(train_path, delimiter=';', header=None, names=['sentence', 'label'])
    val_df = pd.read_csv(val_path, delimiter=';', header=None, names=['sentence', 'label'])
    ts_df = pd.read_csv(test_path, delimiter=';', header=None, names=['sentence', 'label'])
    print("\nTrain DataFrame:\n", df.head())
    print("\nValidation DataFrame:\n", val_df.head())
    print("\nTest DataFrame:\n", ts_df.head())
except FileNotFoundError as e:
    print(f"Error: {e}")
    print("Check dataset files in /content/emotions_dataset/.")
    !ls /content/emotions_dataset/
    raise

# Visualize original label distributions
for dataset, title in [(df, 'Train'), (val_df, 'Validation'), (ts_df, 'Test')]:
    label_counts = dataset['label'].value_counts()
    light_colors = sns.husl_palette(n_colors=len(label_counts))
    sns.set(style="whitegrid")
    plt.figure(figsize=(8, 8))
    plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%', startangle=140, colors=light_co
    plt.title(f'Emotion {title} Distribution')
    plt.show()
```

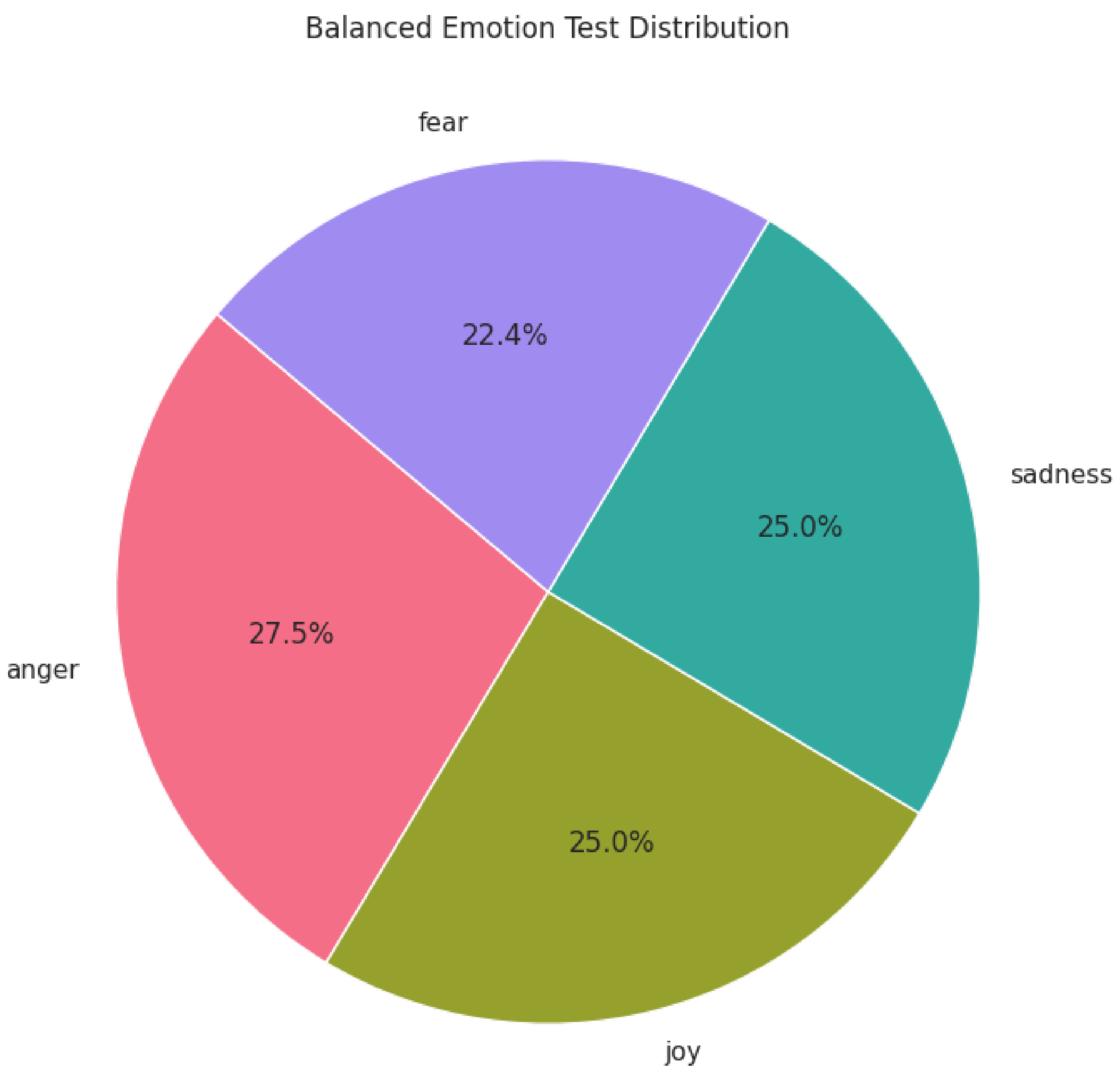
2.2 Balance dataset

```
# Remove 'love' and 'surprise' and sample to balance classes
def balance_df(dataframe, random_state=20):
    df_filtered = dataframe[~dataframe['label'].str.contains('love|surprise', case=False)]
    joy = df_filtered[df_filtered['label'] == 'joy'].sample(n=2200 if dataframe is df else 250, random_
    sad = df_filtered[df_filtered['label'] == 'sadness'].sample(n=2200 if dataframe is df else 250, ran
    fear = df_filtered[df_filtered['label'] == 'fear'].sample(n=1937 if dataframe is df else 212 if dat
    anger = df_filtered[df_filtered['label'] == 'anger'].sample(n=2159 if dataframe is df else 275, ran
    df_sampled = pd.concat([joy, sad, fear, anger]).sample(frac=1, random_state=random_state).reset_ind
    return df_sampled

df = balance_df(df)
val_df = balance_df(val_df)
ts_df = balance_df(ts_df)

# Verify balanced distributions
for dataset, title in [(df, 'Train'), (val_df, 'Validation'), (ts_df, 'Test')]:
    label_counts = dataset['label'].value_counts()
    light_colors = sns.husl_palette(n_colors=len(label_counts))
    sns.set(style="whitegrid")
    plt.figure(figsize=(8, 8))
    plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%', startangle=140, colors=light_co
    plt.title(f'Balanced Emotion {title} Distribution')
    plt.show()
```


Balanced Emotion Train Distribution
fear
22.8%
anger
25.4%
joy
25.9%
25.9%
sadness


Balanced Emotion Validation Distribution
fear
21.5%
sadness
25.3%
anger
27.9%
25.3%

joy


Balanced Emotion Test Distribution
fear
22.4%
sadness
25.0%
anger
27.5%
25.0%
joy

2.3 Split data into X, y

```
tr_text = df['sentence']
tr_label = df['label']
val_text = val_df['sentence']
val_label = val_df['label']
ts_text = ts_df['sentence']
ts_label = ts_df['label']
```

2.4 Encoding

```
encoder = LabelEncoder()
tr_label = encoder.fit_transform(tr_label)
val_label = encoder.transform(val_label)
ts_label = encoder.transform(ts_label)

# Convert to one-hot encoded labels
tr_y = to_categorical(tr_label)
val_y = to_categorical(val_label)
ts_y = to_categorical(ts_label)
```

2.5 Text preprocessing

```
max_words = 5000
max_len = 50
embedding_dim = 100

tokenizer = Tokenizer(num_words=max_words, oov_token='<OOV>')
tokenizer.fit_on_texts(tr_text)

# Convert texts to sequences and pad
tr_sequences = tokenizer.texts_to_sequences(tr_text)
val_sequences = tokenizer.texts_to_sequences(val_text)
ts_sequences = tokenizer.texts_to_sequences(ts_text)

tr_x = pad_sequences(tr_sequences, maxlen=max_len, padding='post')
val_x = pad_sequences(val_sequences, maxlen=max_len, padding='post')
ts_x = pad_sequences(ts_sequences, maxlen=max_len, padding='post')

# Verify shapes
print("tr_x shape:", tr_x.shape)  # Should be (8496, 50)
print("tr_y shape:", tr_y.shape)  # Should be (8496, 4)
print("val_x shape:", val_x.shape)  # Should be (987, 50)
print("val_y shape:", val_y.shape)  # Should be (987, 4)
print("ts_x shape:", ts_x.shape)   # Should be (999, 50)
print("ts_y shape:", ts_y.shape)   # Should be (999, 4)
print("Max index in tr_x:", np.max(tr_x))  # Should be < 5000
print("Max index in val_x:", np.max(val_x))
print("Max index in ts_x:", np.max(ts_x))
```

```
tr_x shape: (8496, 50)
tr_y shape: (8496, 4)
val_x shape: (987, 50)
val_y shape: (987, 4)
ts_x shape: (999, 50)
ts_y shape: (999, 4)
```

```
Max index in tr_x: 4999
Max index in val_x: 4996
Max index in ts_x: 4975
```

3. Building deep learning model

3.1 Model architecture

```
input1 = Input(shape=(max_len,), name='input_1')
input2 = Input(shape=(max_len,), name='input_2')

# Branch 1
x1 = Embedding(max_words, embedding_dim, input_length=max_len)(input1)
x1 = Conv1D(64, 3, padding='same', activation='relu')(x1)
x1 = BatchNormalization()(x1)
x1 = ReLU()(x1)
x1 = Dropout(0.5)(x1)
x1 = GlobalMaxPooling1D()(x1)

# Branch 2
x2 = Embedding(max_words, embedding_dim, input_length=max_len)(input2)
x2 = Conv1D(64, 3, padding='same', activation='relu')(x2)
x2 = BatchNormalization()(x2)
x2 = ReLU()(x2)
x2 = Dropout(0.5)(x2)
x2 = GlobalMaxPooling1D()(x2)

# Concatenate
concatenated = Concatenate()([x1, x2])
hid_layer = Dense(128, activation='relu')(concatenated)
dropout = Dropout(0.3)(hid_layer)
output_layer = Dense(4, activation='softmax')(dropout)

# Define model
model = Model(inputs=[input1, input2], outputs=output_layer)
model.summary()
```

```
Model: "functional_4"

┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)         ┃ Output Shape       ┃    Param # ┃ Connected to       ┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━┩
│ input_1              │ (None, 50)         │          0 │ -                  │
│ (InputLayer)         │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ input_2              │ (None, 50)         │          0 │ -                  │
│ (InputLayer)         │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ embedding_8          │ (None, 50, 100)    │    500,000 │ input_1[0][0]      │
│ (Embedding)          │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ embedding_9          │ (None, 50, 100)    │    500,000 │ input_2[0][0]      │
│ (Embedding)          │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ conv1d_8 (Conv1D)    │ (None, 50, 64)     │     19,264 │ embedding_8[0][0]  │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ conv1d_9 (Conv1D)    │ (None, 50, 64)     │     19,264 │ embedding_9[0][0]  │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ batch_normalizatio…  │ (None, 50, 64)     │        256 │ conv1d_8[0][0]     │
│ (BatchNormalizatio…  │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ batch_normalizatio…  │ (None, 50, 64)     │        256 │ conv1d_9[0][0]     │
│ (BatchNormalizatio…  │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ re_lu_8 (ReLU)       │ (None, 50, 64)     │          0 │ batch_normalizat…  │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ re_lu_9 (ReLU)       │ (None, 50, 64)     │          0 │ batch_normalizat…  │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ dropout_12           │ (None, 50, 64)     │          0 │ re_lu_8[0][0]      │
│ (Dropout)            │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ dropout_13           │ (None, 50, 64)     │          0 │ re_lu_9[0][0]      │
│ (Dropout)            │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ global_max_pooling…  │ (None, 64)         │          0 │ dropout_12[0][0]   │
│ (GlobalMaxPooling1…  │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ global_max_pooling…  │ (None, 64)         │          0 │ dropout_13[0][0]   │
│ (GlobalMaxPooling1…  │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ concatenate_4        │ (None, 128)        │          0 │ global_max_pooli…  │
│ (Concatenate)        │                    │            │ global_max_pooli…  │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ dense_8 (Dense)      │ (None, 128)        │     16,512 │ concatenate_4[0]…  │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ dropout_14           │ (None, 128)        │          0 │ dense_8[0][0]      │
│ (Dropout)            │                    │            │                    │
├──────────────────────┼────────────────────┼────────────┼────────────────────┤
│ dense_9 (Dense)      │ (None, 4)          │        516 │ dropout_14[0][0]   │
```

3.2 Compile model

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

3.3 Train the model

```
batch_size = 256
```

```
epochs = 25
history = model.fit(
    [tr_x, tr_x], tr_y,
    epochs=epochs,
    batch_size=batch_size,
    validation_data=([val_x, val_x], val_y)
)

# 4. Evaluation and Visualize results

## 4.1 Evaluate the model
loss, accuracy, precision, recall = model.evaluate([tr_x, tr_x], tr_y)
print(f'Training - Loss: {round(loss, 2)}, Accuracy: {round(accuracy, 2)}, Precision: {round(precision, 2

loss, accuracy, precision, recall = model.evaluate([ts_x, ts_x], ts_y)
print(f'Test - Loss: {round(loss, 2)}, Accuracy: {round(accuracy, 2)}, Precision: {round(precision, 2)},

## 4.2 Visualize results
tr_acc = history.history['accuracy']
tr_loss = history.history['loss']
tr_per = history.history['precision']
tr_recall = history.history['recall']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
val_per = history.history['val_precision']
val_recall = history.history['val_recall']

index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
index_precision = np.argmax(val_per)
per_highest = val_per[index_precision]
index_recall = np.argmax(val_recall)
recall_highest = val_recall[index_recall]

Epochs = [i + 1 for i in range(len(tr_acc))]
loss_label = f'Best epoch = {str(index_loss + 1)}'
acc_label = f'Best epoch = {str(index_acc + 1)}'
per_label = f'Best epoch = {str(index_precision + 1)}'
recall_label = f'Best epoch = {str(index_recall + 1)}'
```

```
Epoch 1/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 11s 314ms/step - accuracy: 0.4659 - loss: 1.2044 - precision_5: 0.6888
Epoch 2/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 15s 160ms/step - accuracy: 0.6911 - loss: 0.7931 - precision_5: 0.8105
Epoch 3/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 11s 202ms/step - accuracy: 0.8951 - loss: 0.3170 - precision_5: 0.925
Epoch 4/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 5s 139ms/step - accuracy: 0.9510 - loss: 0.1572 - precision_5: 0.9565
Epoch 5/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 5s 138ms/step - accuracy: 0.9625 - loss: 0.1161 - precision_5: 0.9686
Epoch 6/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 7s 194ms/step - accuracy: 0.9775 - loss: 0.0737 - precision_5: 0.9794
Epoch 7/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 5s 139ms/step - accuracy: 0.9801 - loss: 0.0581 - precision_5: 0.9814
Epoch 8/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 6s 167ms/step - accuracy: 0.9844 - loss: 0.0504 - precision_5: 0.9858
Epoch 9/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 6s 170ms/step - accuracy: 0.9853 - loss: 0.0418 - precision_5: 0.9864
Epoch 10/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 11s 194ms/step - accuracy: 0.9877 - loss: 0.0379 - precision_5: 0.9888
Epoch 11/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 5s 142ms/step - accuracy: 0.9884 - loss: 0.0349 - precision_5: 0.9887
Epoch 12/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 5s 140ms/step - accuracy: 0.9924 - loss: 0.0250 - precision_5: 0.9933
Epoch 13/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 7s 197ms/step - accuracy: 0.9900 - loss: 0.0316 - precision_5: 0.9906
Epoch 14/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 8s 139ms/step - accuracy: 0.9936 - loss: 0.0200 - precision_5: 0.9945
Epoch 15/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 7s 203ms/step - accuracy: 0.9935 - loss: 0.0198 - precision_5: 0.9935
Epoch 16/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 8s 138ms/step - accuracy: 0.9917 - loss: 0.0242 - precision_5: 0.9921
Epoch 17/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 7s 201ms/step - accuracy: 0.9931 - loss: 0.0209 - precision_5: 0.9932
Epoch 18/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 5s 140ms/step - accuracy: 0.9938 - loss: 0.0217 - precision_5: 0.9939
Epoch 19/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 6s 178ms/step - accuracy: 0.9947 - loss: 0.0162 - precision_5: 0.9951
Epoch 20/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 9s 155ms/step - accuracy: 0.9956 - loss: 0.0142 - precision_5: 0.9957
Epoch 21/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 10s 142ms/step - accuracy: 0.9934 - loss: 0.0210 - precision_5: 0.9937
Epoch 22/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 6s 157ms/step - accuracy: 0.9944 - loss: 0.0174 - precision_5: 0.9944
Epoch 23/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 10s 144ms/step - accuracy: 0.9963 - loss: 0.0135 - precision_5: 0.996
Epoch 24/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 7s 200ms/step - accuracy: 0.9937 - loss: 0.0168 - precision_5: 0.9938
Epoch 25/25
34/34 ━━━━━━━━━━━━━━━━━━━━ 8s 140ms/step - accuracy: 0.9949 - loss: 0.0159 - precision_5: 0.9949
266/266 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9986 - loss: 0.0062 - precision_5: 0.9986
Training - Loss: 0.01, Accuracy: 1.0, Precision: 1.0, Recall: 1.0
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9409 - loss: 0.1500 - precision_5: 0.9451 -
Test - Loss: 0.17, Accuracy: 0.94, Precision: 0.94, Recall: 0.93
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-55-376c7ae394a1> in <cell line: 0>()
     20 tr_acc = history.history['accuracy']
     21 tr_loss = history.history['loss']
---> 22 tr_per = history.history['precision']
     23 tr_recall = history.history['recall']
     24 val_acc = history.history['val_accuracy']

KeyError: 'precision'
```

Next steps: Explain error

Double-click (or enter) to edit

4.2 Visualize results

```
plt.figure(figsize=(20, 12))
plt.style.use('fivethirtyeight')

plt.subplot(2, 2, 1)
plt.plot(Epochs, tr_loss, 'r', label='Training loss')
plt.plot(Epochs, val_loss, 'g', label='Validation loss')
plt.scatter(index_loss + 1, val_lowest, s=150, c='blue', label=loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(Epochs, tr_acc, 'r', label='Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label='Validation Accuracy')
plt.scatter(index_acc + 1, acc_highest, s=150, c='blue', label=acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 3)
plt.plot(Epochs, tr_per, 'r', label='Precision')
plt.plot(Epochs, val_per, 'g', label='Validation Precision')
plt.scatter(index_precision + 1, per_highest, s=150, c='blue', label=per_label)
plt.title('Precision and Validation Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(Epochs, tr_recall, 'r', label='Recall')
plt.plot(Epochs, val_recall, 'g', label='Validation Recall')
plt.scatter(index_recall + 1, recall_highest, s=150, c='blue', label=recall_label)
plt.title('Recall and Validation Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()
plt.grid(True)

plt.suptitle('Model Training Metrics Over Epochs', fontsize=16)
plt.tight_layout()
plt.show()

# Confusion matrix
y_true = np.argmax(ts_y, axis=1)
preds = model.predict([ts_x, ts_x])
y_pred = np.argmax(preds, axis=1)

plt.figure(figsize=(8, 6))
emotions = {0: 'anger', 1: 'fear', 2: 'joy', 3: 'sadness'}
emotions = list(emotions.values())
```

```
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=emotions, yticklabels=emotions)
plt.title('Confusion Matrix')
plt.show()

print("Classification Report:\n", classification_report(y_true, y_pred, target_names=emotions))
```