Twitter Emotions Sentiment Analysis

1. Packages Import

In this section, essential Python packages for data manipulation, machine learning, and model evaluation were imported. Key libraries include TensorFlow for neural network construction, pandas for data handling, and Matplotlib for visualizations. The use of these packages allowed for efficient data processing, model training, and evaluation with minimal custom code, leveraging well-tested frameworks for robust results.

2. Data Loading and Exploration

The dataset was loaded, and initial exploration was conducted to understand its structure. This included inspecting the labels, data size, and overall quality. Statistics provided an overview of the dataset and guided subsequent data processing steps.

```
Dataset length: 416809

{0: 'sadness', 1: 'joy', 2: 'love', 3: 'anger', 4: 'fear', 5: 'surprise'}
                                                text  label
0  i feel awful about it too because it s my job ...      0
1                             im alone i feel awful      0
2  ive probably mentioned this before but i reall...      1
3          i was feeling a little low few days back      0
4  i beleive that i am much more sensitive to oth...      2
```

3. Data Preprocessing

Data preprocessing was performed to ensure data quality and compatibility with neural network models. Each preprocessing step aimed to clean and standardize the data, which is essential for effective training.

3.1. Lowercasing

All text was converted to lowercase to ensure uniformity, which helps reduce the dimensionality of the data.

3.2. Removing punctuation, emoticons, special characters, and URLs

Unnecessary characters were removed to prevent noise, focusing the model on meaningful words.

3.3. Stop words removal

Common but uninformative words were removed to allow the model to focus on relevant terms.

3.4. Duplicated and empty texts removal

Duplicate entries and empty rows were removed to avoid biases and irrelevant data during training.

3.5. Tokenization (to words)

The text was split into individual words to prepare for stemming and further processing.

	text_tokenized_to_words	text
0	[feel, awful, job, get, position, succeed, hap...	feel awful job get position succeed happen
1	[im, alone, feel, awful]	im alone feel awful
2	[ive, probably, mentioned, really, feel, proud...	ive probably mentioned really feel proud actua...
3	[feeling, little, low, days, back]	feeling little low days back
4	[beleive, much, sensitive, peoples, feelings, ...	beleive much sensitive peoples feelings tend c...

3.6. Stemming

Words were reduced to their base forms to further reduce data dimensionality.

	text_tokenized_to_words	text_tokenized_to_stemmed_words
0	[feel, awful, job, get, position, succeed, hap...	[feel, aw, job, get, posit, succeed, happen]
1	[im, alone, feel, awful]	[im, alon, feel, aw]
2	[ive, probably, mentioned, really, feel, proud...	[ive, probabl, mention, realli, feel, proud, a...
3	[feeling, little, low, days, back]	[feel, littl, low, day, back]
4	[beleive, much, sensitive, peoples, feelings, ...	[beleiv, much, sensit, peopl, feel, tend, comp...

3.7. Tokenization (to numbers)

Each word was converted to a numeric token using a tokenizer, facilitating input into neural networks.

	text_tokenized_to_numeric	text_tokenized_to_stemmed_words
0	[1, 349, 223, 4, 181, 1860, 122]	[feel, aw, job, get, posit, succeed, happen]
1	[3, 134, 1, 349]	[im, alon, feel, aw]
2	[21, 230, 662, 6, 1, 311, 79, 100, 88, 44, 260...	[ive, probabl, mention, realli, feel, proud, a...
3	[1, 13, 325, 17, 36]	[feel, littl, low, day, back]
4	[10013, 22, 1467, 14, 1, 759, 908]	[beleiv, much, sensit, peopl, feel, tend, comp...

3.8. Trim & pad text

Text sequences were trimmed or padded to **40 tokens** to ensure uniform input length.

3.9. Split data

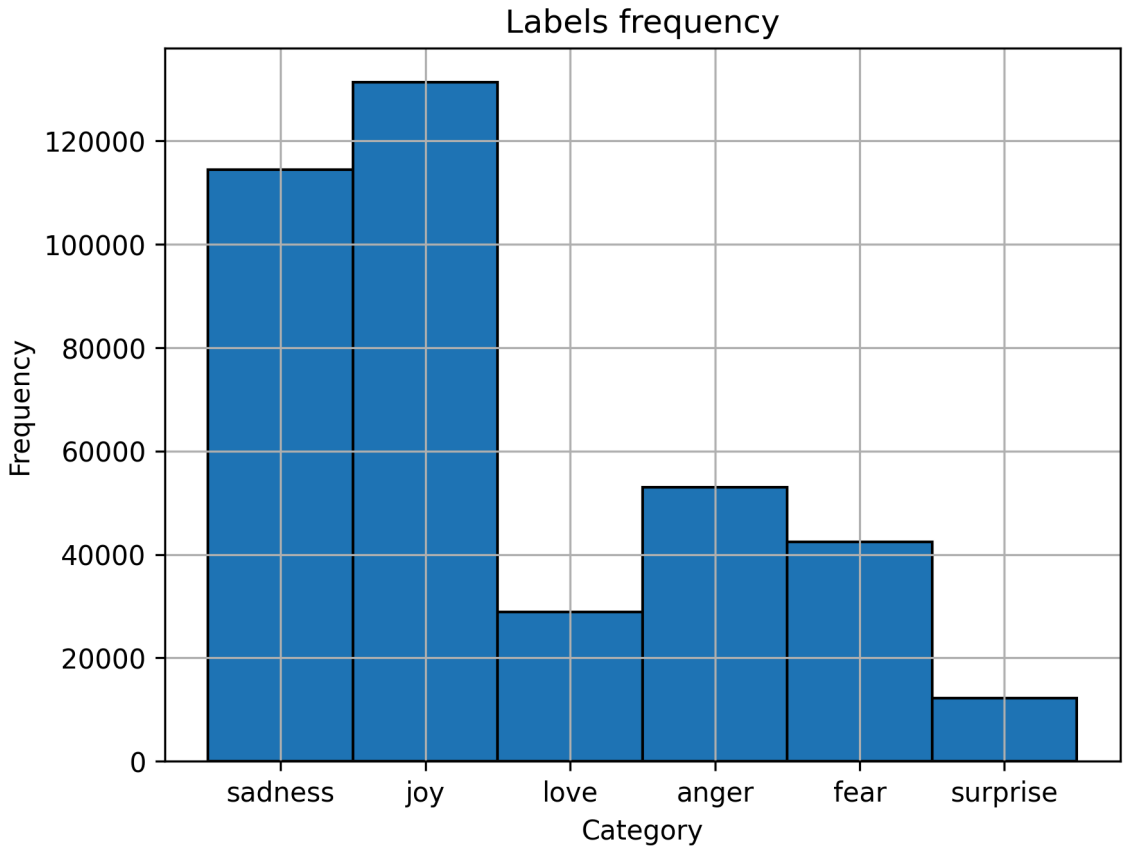
Data was split into training, validation, and test sets to evaluate model performance effectively.

4. Exploratory Data Analysis (EDA)

EDA was conducted to gain deeper insights into the data and identify patterns or issues that may impact model performance. Visualizations were used to examine the distribution of emotions, text lengths, and common words.

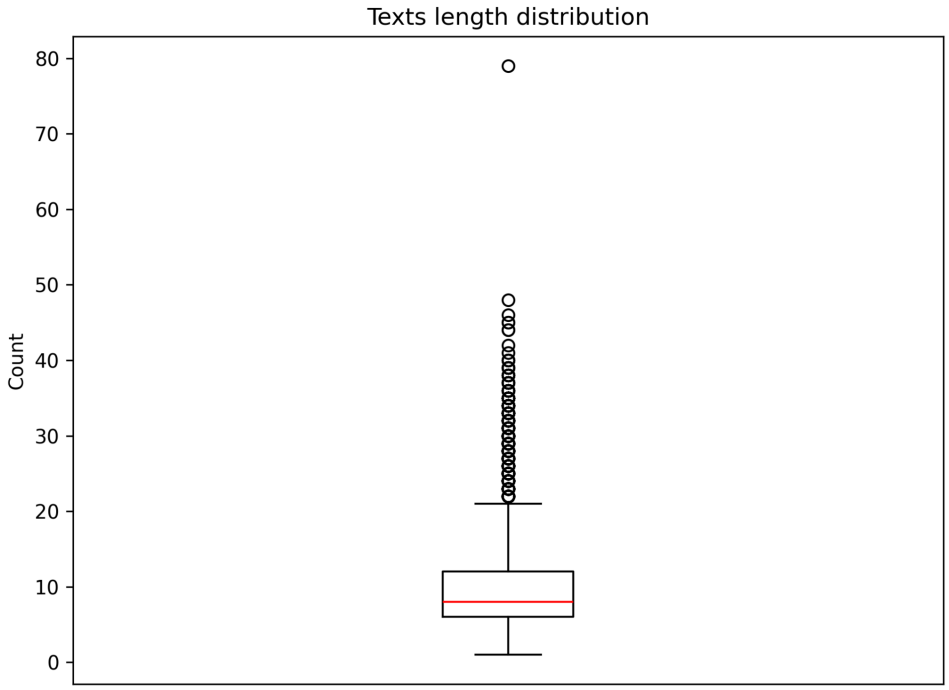
4.1. Emotions category distribution

Analyzed the distribution of each emotion category to identify any imbalances.



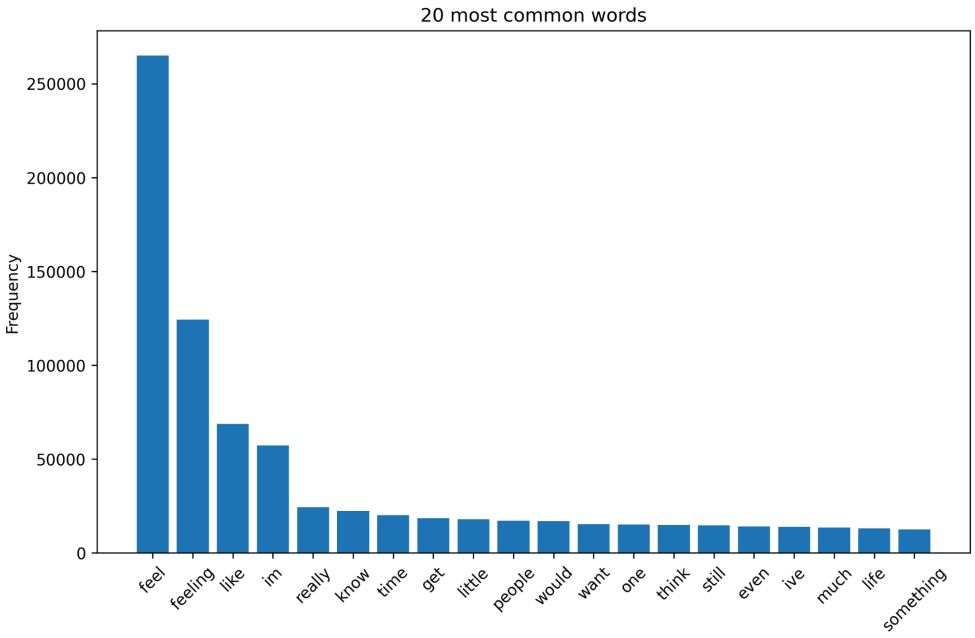
4.2. Texts length distribution

Examined the length of tweets to determine an optimal sequence length for the model.



4.3. Common words frequency

Identified frequently occurring words to understand the dataset's vocabulary.



5. Model Building and Configuration

Multiple neural network architectures were explored, including LSTM, Bidirectional LSTM, and GRU models. Each architecture was designed to capture temporal dependencies in the text data.

5.1. LSTM model architecture

Implemented a basic LSTM architecture for sequential data processing.

```python
def create_LSTM_model(params):
    model = Sequential([
        Embedding(input_dim=params['input_dim'], output_dim=params['output_dim'], input_length=params['input_length']),
        LSTM(params['lstm_units'], return_sequences=True),
        Dropout(params['dropout']),
        Flatten(),
        Dense(params['dense_units'], activation='relu', kernel_regularizer=l1_l2(params['l1'], params['l2'])),
        Dropout(params['dropout']),
        Dense(params['num_classes'], activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

    return model
```

✓ 0.0s

5.2. Bidirectional LSTM model architecture

Used a Bidirectional LSTM to capture dependencies in both directions.

```python
def create_BiLSTM_model(params):
    model = Sequential([
        Embedding(input_dim=params['input_dim'], output_dim=params['output_dim'], input_length=params['input_length']),
        Bidirectional(LSTM(params['lstm_units'], return_sequences=True)),
        Dropout(params['dropout']),
        Flatten(),
        Dense(params['dense_units'], activation='relu', kernel_regularizer=l1_l2(params['l1'], params['l2'])),
        Dropout(params['dropout']),
        Dense(params['num_classes'], activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

    return model
```

✓ 0.0s

5.3. GRU model architecture

Employed a GRU model as an alternative to LSTM for faster training.

```python
def create_GRU_model(params):
    model = Sequential([
        Embedding(input_dim=params['input_dim'], output_dim=params['output_dim'], input_length=params['input_length']),
        GRU(params['lstm_units'], return_sequences=True),
        Dropout(params['dropout']),
        Flatten(),
        Dense(params['dense_units'], activation='relu', kernel_regularizer=l1_l2(params['l1'], params['l2'])),
        Dropout(params['dropout']),
        Dense(params['num_classes'], activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
```

5.4. Model training configuration

Configured training parameters, including batch size and epochs, for optimal performance.

6. Hyperparameter Tuning and Training

Hyperparameter tuning was performed using grid search to optimize model performance. Key hyperparameters included the number of LSTM units, dropout rates, and learning rates.

6.1. MLflow Metrics and Artifacts Setup

Set up MLflow to track experiments, metrics, and model artifacts.

6.2. MLflow Experiment Setup

Defined experiments in MLflow to systematically evaluate different configurations.

6.3. Hyperparameter Grid Search

Conducted grid search to find optimal hyperparameters for best accuracy.

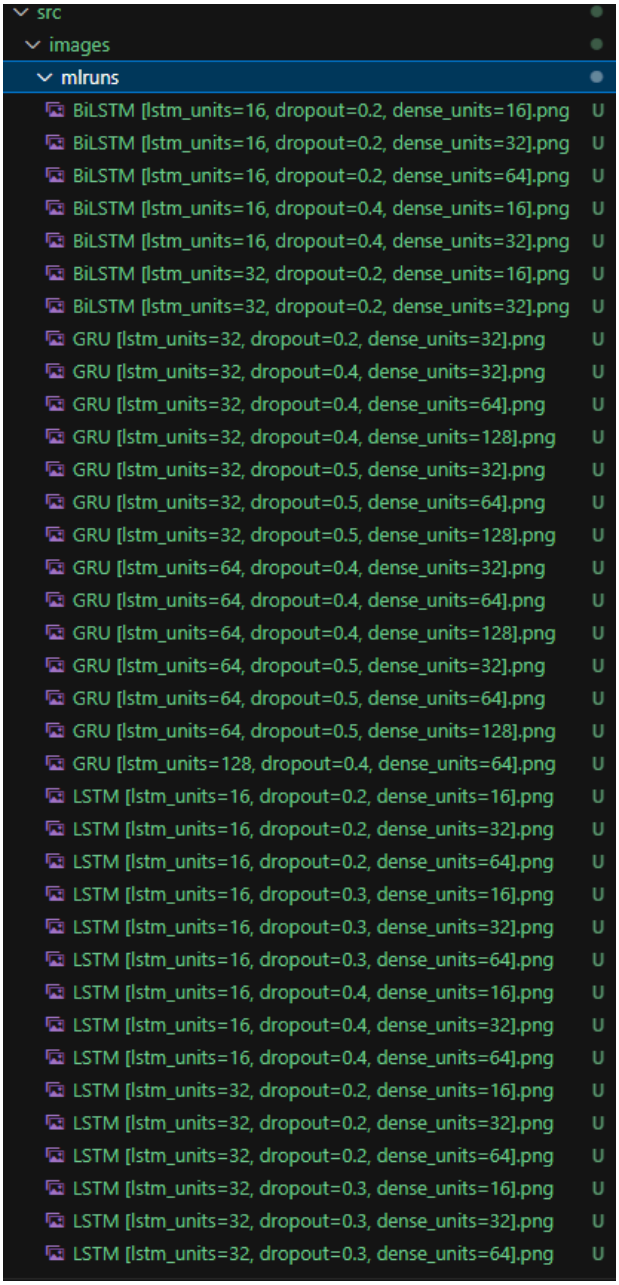
Each NN (LSTM, BiLSTM, GRU) was trained against several hyperparameters. Below initial values for model and training parameters and example of grid parameters. During grid search **3 parameters** were likely to change **lstm_units, dense_units, dropout**. All others remained fixed for all investigated models.

```python
init_model_params = {
    'input_dim': unique_words_count + 1,
    'output_dim': 64,
    'input_length': 40,
    'lstm_units': 16,
    'dense_units': 16,
    'l1': 0.00001,
    'l2': 0.0001,
    'dropout': 0.2,
    'num_classes': 6
}

init_training_params = {
    'epochs': 20,
    'batch_size': 25000
}
```

```
param_grid = {
    'lstm_units': [16, 32, 64],
    'dense_units': [16, 32, 64],
    'dropout': [0.2, 0.4],
}
```

As a result 40 models were calculated. Plots demonstrating loss and accuracy during learning are saved in *src/images/mlruns* directory.



Based on loss and accuracy **GRU [lstm_units=32, dropout=0.4, dense_units=32]** was selected as the best model for prediction purposes. Selected GRU model has high accuracy over 90% - it is not undertrained. Metrics for training, validation and test sets are similar - it is not overfitted as well. Learning could be finished at 12 epochs, since then accuracy stopped increasing significantly on the validation set.

```
Started run for 'GRU [lstm_units=32, dropout=0.4, dense_units=32]'
Epoch 1/20
10/10 [==============================] - 17s 2s/step - loss: 1.8116 - accuracy: 0.1748 - val_loss: 1.7972 - val_accuracy: 0.2605
Epoch 2/20
10/10 [==============================] - 16s 2s/step - loss: 1.7930 - accuracy: 0.2410 - val_loss: 1.7648 - val_accuracy: 0.4269
Epoch 3/20
10/10 [==============================] - 16s 2s/step - loss: 1.7239 - accuracy: 0.3673 - val_loss: 1.6373 - val_accuracy: 0.5047
Epoch 4/20
10/10 [==============================] - 16s 2s/step - loss: 1.5314 - accuracy: 0.4698 - val_loss: 1.3638 - val_accuracy: 0.5825
Epoch 5/20
10/10 [==============================] - 16s 2s/step - loss: 1.2024 - accuracy: 0.5761 - val_loss: 0.9583 - val_accuracy: 0.7607
Epoch 6/20
10/10 [==============================] - 16s 2s/step - loss: 0.8549 - accuracy: 0.7019 - val_loss: 0.6773 - val_accuracy: 0.8237
Epoch 7/20
10/10 [==============================] - 16s 2s/step - loss: 0.6220 - accuracy: 0.7933 - val_loss: 0.5400 - val_accuracy: 0.8506
Epoch 8/20
10/10 [==============================] - 16s 2s/step - loss: 0.4810 - accuracy: 0.8496 - val_loss: 0.4890 - val_accuracy: 0.8634
Epoch 9/20
10/10 [==============================] - 16s 2s/step - loss: 0.3999 - accuracy: 0.8787 - val_loss: 0.4313 - val_accuracy: 0.8798
Epoch 10/20
10/10 [==============================] - 16s 2s/step - loss: 0.3539 - accuracy: 0.8942 - val_loss: 0.4184 - val_accuracy: 0.8846
Epoch 11/20
10/10 [==============================] - 17s 2s/step - loss: 0.3255 - accuracy: 0.9033 - val_loss: 0.4004 - val_accuracy: 0.8886
Epoch 12/20
10/10 [==============================] - 16s 2s/step - loss: 0.3038 - accuracy: 0.9092 - val_loss: 0.3993 - val_accuracy: 0.8890
...
10/10 [==============================] - 17s 2s/step - loss: 0.2172 - accuracy: 0.9339 - val_loss: 0.3644 - val_accuracy: 0.9024
Epoch 20/20
10/10 [==============================] - 17s 2s/step - loss: 0.2073 - accuracy: 0.9366 - val_loss: 0.3588 - val_accuracy: 0.9032
```

7. Model Evaluation and Validation

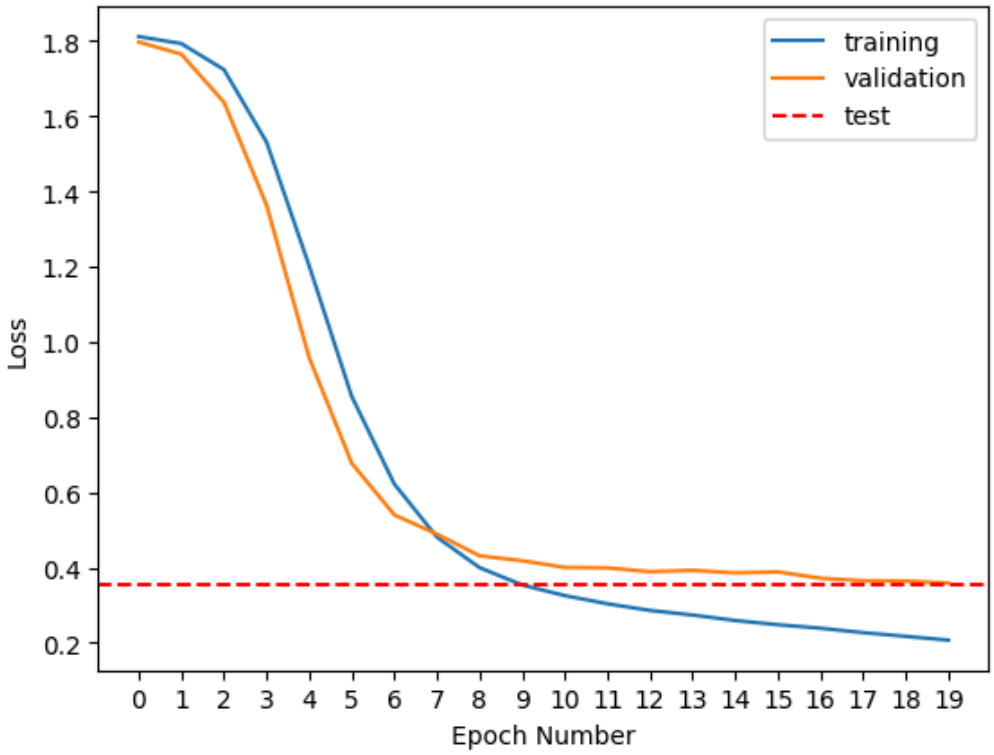
The trained models were evaluated on a test set, using various metrics to assess accuracy and robustness.

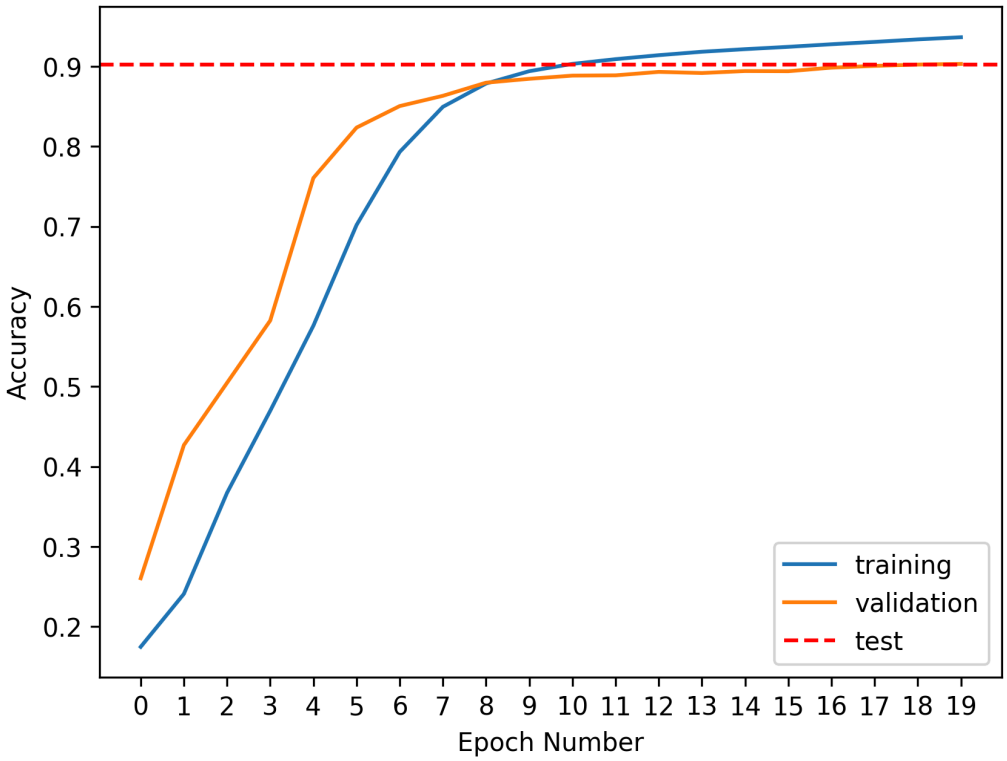
7.1. Selected Model Loading

Loaded the best-performing model based on loss and accuracy results:
GRU [lstm_units=32, dropout=0.4, dense_units=32]

7.2. Loss & Accuracy

Evaluated model performance in terms of accuracy and loss on the test data.


training
validation
test
Loss
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Epoch Number


Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Epoch Number
training
validation
test

7.3. Classification report

Generated a classification report detailing precision, recall, and F1-score for each emotion.

```
2389/2389 [==============================] - 4s 1ms/step
              precision    recall  f1-score   support

           0       0.95      0.92      0.93     22879
           1       0.95      0.88      0.92     26265
           2       0.73      0.89      0.80      5769
           3       0.88      0.92      0.90     10603
           4       0.86      0.90      0.88      8475
           5       0.75      0.89      0.81      2440

    accuracy                           0.90     76431
   macro avg       0.85      0.90      0.87     76431
weighted avg       0.91      0.90      0.90     76431
```

7.4. Confusion matrix

Visualized the confusion matrix to understand model errors and misclassifications.