Emotion Classification Project Implementation Guide

1. Environment Setup

- Import essential libraries for sequence modeling
- Tokenizer and pad_sequences for text preprocessing
- RNN variants from Keras layers
- load_dataset for Hugging Face datasets integration
- matplotlib for visualization

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt
from datasets import load_dataset
```

Data Loading

```
emotions = load_dataset("emotion")
emotions
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```

```
# sample data
emotions['train']['text'][0]
```

```
'i didnt feel humiliated'
```

Text Preprocessing

- Create word-level vocabulary with 10,000 words capacity
- Convert text to numerical indices
- Dynamic padding using 95th percentile of sequence lengths
- Post-padding to handle variable length sequences

```
## Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(emotions['train']['text'])

#get sequences
train_sequences = tokenizer.texts_to_sequences(emotions['train']['text'])
test_sequences = tokenizer.texts_to_sequences(emotions['test']['text'])

# get max length for padding
```

```
lengths = [len(seq) for seq in train_sequences]
max_length = max(lengths)

# pad sequences
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding='post', truncating='post')
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='post', truncating='post')


# Extract labels
train_labels = emotions['train']['label']
test_labels = emotions['test']['label']
```

Dataset Preparation

- Create TensorFlow Dataset objects
- Shuffle and batch training data
- Prefetch for optimized pipeline
- Maintain same batch size for comparison

```
BATCH_SIZE = 64

# Prepare training data - wrap features and labels in a TUPLE
train_dataset = tf.data.Dataset.from_tensor_slices((train_padded, train_labels))
train_dataset = train_dataset.shuffle(1000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

# Prepare test data
test_dataset = tf.data.Dataset.from_tensor_slices((test_padded, test_labels))
test_dataset = test_dataset.batch(BATCH_SIZE)
```

Model Building (RNN/LSTM/GRU)

```
# def build_model(model_type='lstm'):
#   model=Sequential([
#       Embedding(10000,32,input_length=max_length),
#       Bidirectional(LSTM(64 , return_sequences=True)) if model_type=='lstm' else
#       Bidirectional(GRU(64 , return_sequences=True)) if model_type=='gru' else
#       Bidirectional(SimpleRNN(64 , return_sequences=True)),
#       Bidirectional(LSTM(64)),
#       tf.keras.layers.GlobalAveragePooling1D(),

#       Dense(64, activation='relu'),
#       Dense(6, activation='softmax')

#   ])
#   model.compile(
#         optimizer=tf.keras.optimizers.Adam(1e-3),
#         loss='sparse_categorical_crossentropy',
#         metrics=['accuracy']
#     )
#   return model


# 1. Define Models Properly
def build_model(model_type):
    model = Sequential()
    model.add(Embedding(10000, 128))
    if model_type == 'lstm':
        model.add(LSTM(64))
    elif model_type == 'gru':
        model.add(GRU(64))
    elif model_type == 'rnn':
        model.add(SimpleRNN(64))
    else:
        raise ValueError("Invalid model type!")
    model.add(Dense(6, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model  # <-- Explicit return

# 2. Initialize Models
```

```
lstm_model = build_model('lstm')
gru_model = build_model('gru')
rnn_model = build_model('rnn')
```


```
lstm_model = build_model('lstm')
gru_model = build_model('gru')
rnn_model = build_model('rnn')
```


Training & Evaluation|

```
# 3. Create Models Dictionary
models = {
    'LSTM': lstm_model,
    'GRU': gru_model,
    'RNN': rnn_model
}

# 4. Train Models
for model_name, model in models.items():
    print(f"Training {model_name}...")
    history = model.fit(
        train_dataset,
        validation_data=test_dataset,
        epochs=10,
        callbacks=[tf.keras.callbacks.EarlyStopping(patience=3)]
    )
```

```
Training LSTM...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.3368 - loss: 1.6021 - val_accuracy: 0.3475 - val_loss: 1.5594
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3383 - loss: 1.5802 - val_accuracy: 0.3475 - val_loss: 1.5618
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3373 - loss: 1.5838 - val_accuracy: 0.2910 - val_loss: 1.5630
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3355 - loss: 1.5830 - val_accuracy: 0.3475 - val_loss: 1.5600
Training GRU...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.3348 - loss: 1.6029 - val_accuracy: 0.3475 - val_loss: 1.5649
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3353 - loss: 1.5866 - val_accuracy: 0.3475 - val_loss: 1.5625
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.3355 - loss: 1.5830 - val_accuracy: 0.3475 - val_loss: 1.5595
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3715 - loss: 1.5358 - val_accuracy: 0.6355 - val_loss: 0.9082
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.7088 - loss: 0.7727 - val_accuracy: 0.9080 - val_loss: 0.2581
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9230 - loss: 0.2083 - val_accuracy: 0.9190 - val_loss: 0.1978
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9489 - loss: 0.1206 - val_accuracy: 0.9220 - val_loss: 0.1955
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9593 - loss: 0.0919 - val_accuracy: 0.9190 - val_loss: 0.2078
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9685 - loss: 0.0740 - val_accuracy: 0.9190 - val_loss: 0.2322
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9739 - loss: 0.0631 - val_accuracy: 0.9195 - val_loss: 0.2182
Training RNN...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.3271 - loss: 1.6005 - val_accuracy: 0.3475 - val_loss: 1.5595
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.3420 - loss: 1.5795 - val_accuracy: 0.3150 - val_loss: 1.5739
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3760 - loss: 1.5319 - val_accuracy: 0.3470 - val_loss: 1.5910
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4608 - loss: 1.4298 - val_accuracy: 0.3865 - val_loss: 1.5825
```

```
# 1. Define Models Properly
def build_model(model_type):
    model = Sequential()
    model.add(Embedding(10000, 128))
    if model_type == 'lstm':
        model.add(LSTM(64))
    elif model_type == 'gru':
        model.add(GRU(64))
    elif model_type == 'rnn':
        model.add(SimpleRNN(64))
```

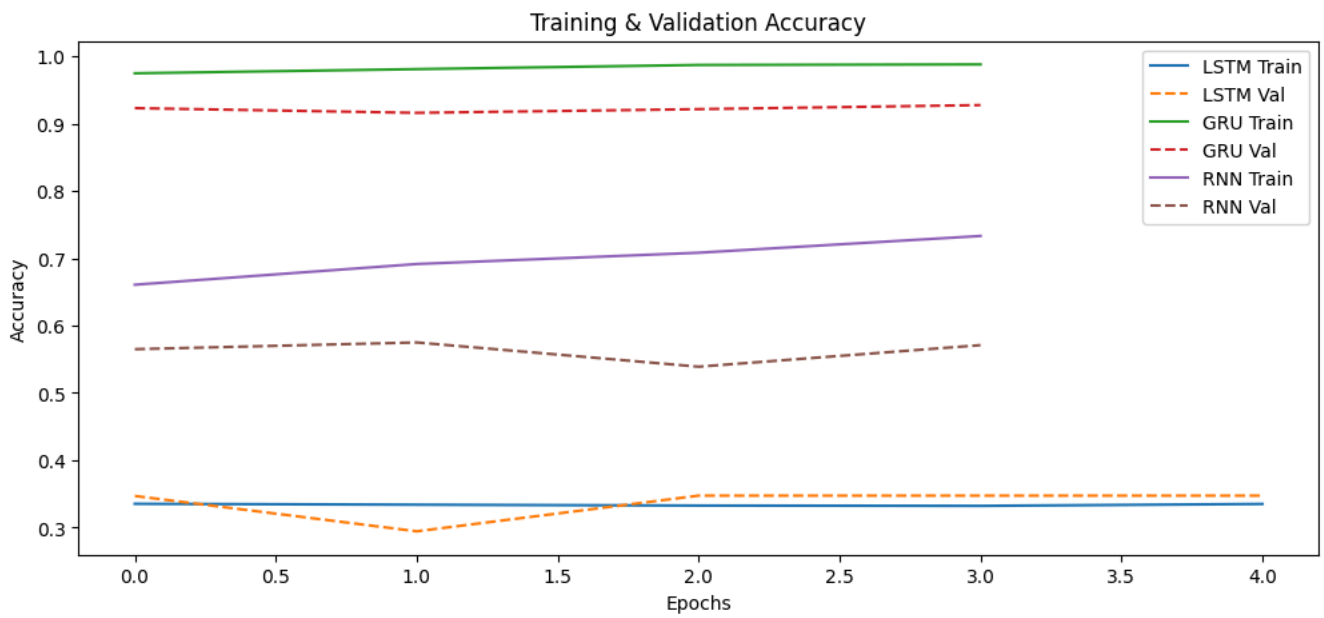
```
        model.add(SimpleRNN(64))
    else:
        raise ValueError("Invalid model type!")
    model.add(Dense(6, activation='softmax'))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model  # <-- Explicit return

# 2. Initialize Models
lstm_model = build_model('lstm')
gru_model = build_model('gru')
rnn_model = build_model('rnn')

# 3. Create Models Dictionary
models = {
    'LSTM': lstm_model,
    'GRU': gru_model,
    'RNN': rnn_model
}

# 4. Train Models
for model_name, model in models.items():
    print(f"Training {model_name}...")
    history = model.fit(
        train_dataset,
        validation_data=test_dataset,
        epochs=10,
        callbacks=[tf.keras.callbacks.EarlyStopping(patience=3)]
    )
```

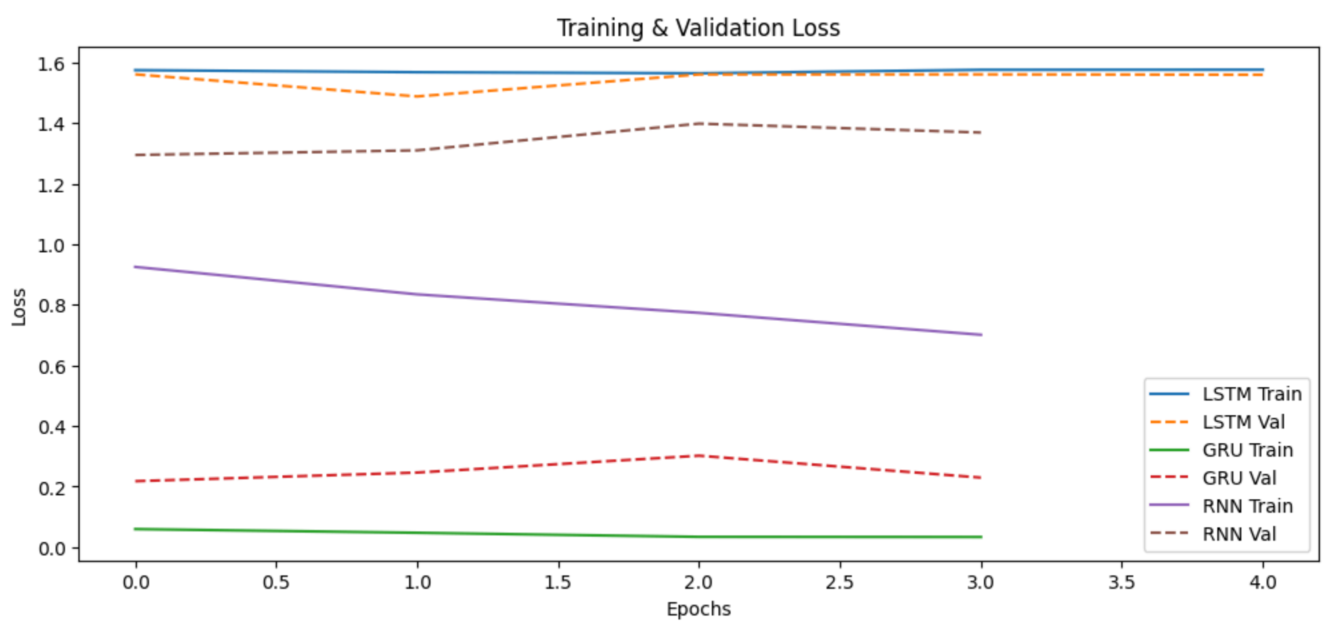
```
Training LSTM...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.3344 - loss: 1.6023 - val_accuracy: 0.3475 - val_loss: 1.5592
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.3367 - loss: 1.5807 - val_accuracy: 0.3475 - val_loss: 1.5613
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3378 - loss: 1.5817 - val_accuracy: 0.3475 - val_loss: 1.5599
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.3361 - loss: 1.5804 - val_accuracy: 0.3470 - val_loss: 1.5617
Training GRU...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.3246 - loss: 1.6037 - val_accuracy: 0.3475 - val_loss: 1.5661
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3362 - loss: 1.5823 - val_accuracy: 0.3475 - val_loss: 1.5614
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.3400 - loss: 1.5837 - val_accuracy: 0.3475 - val_loss: 1.5615
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3447 - loss: 1.5713 - val_accuracy: 0.6110 - val_loss: 1.0256
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.6665 - loss: 0.8739 - val_accuracy: 0.8790 - val_loss: 0.3603
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8866 - loss: 0.3212 - val_accuracy: 0.9185 - val_loss: 0.2217
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9362 - loss: 0.1709 - val_accuracy: 0.9230 - val_loss: 0.2095
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9490 - loss: 0.1222 - val_accuracy: 0.9200 - val_loss: 0.1999
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9639 - loss: 0.0856 - val_accuracy: 0.9180 - val_loss: 0.2004
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9735 - loss: 0.0691 - val_accuracy: 0.9150 - val_loss: 0.2507
Training RNN...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.3215 - loss: 1.6093 - val_accuracy: 0.2885 - val_loss: 1.5642
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.3248 - loss: 1.5811 - val_accuracy: 0.3505 - val_loss: 1.5602
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3951 - loss: 1.5425 - val_accuracy: 0.3930 - val_loss: 1.5447
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4545 - loss: 1.4642 - val_accuracy: 0.4680 - val_loss: 1.4388
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.4809 - loss: 1.4089 - val_accuracy: 0.4750 - val_loss: 1.4159
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.5097 - loss: 1.3333 - val_accuracy: 0.4975 - val_loss: 1.3586
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.5234 - loss: 1.2899 - val_accuracy: 0.4870 - val_loss: 1.3847
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.5575 - loss: 1.2145 - val_accuracy: 0.5160 - val_loss: 1.3996
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.5898 - loss: 1.1134 - val_accuracy: 0.5465 - val_loss: 1.3131
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.6150 - loss: 1.0691 - val_accuracy: 0.5660 - val_loss: 1.2506
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt

# Define a dictionary to store history objects
histories = {}

# Train Models & Store Histories
for model_name, model in models.items():
    print(f"Training {model_name}...")
    histories[model_name] = model.fit(
        train_dataset,
        validation_data=test_dataset,
        epochs=10,
        callbacks=[tf.keras.callbacks.EarlyStopping(patience=3)]
    )

# Plot Training Accuracy & Validation Accuracy
plt.figure(figsize=(12, 5))
for model_name, history in histories.items():
    plt.plot(history.history['accuracy'], label=f'{model_name} Train')
    plt.plot(history.history['val_accuracy'], linestyle='dashed', label=f'{model_name} Val')

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
plt.legend()
plt.show()

# Plot Training Loss & Validation Loss
plt.figure(figsize=(12, 5))
for model_name, history in histories.items():
    plt.plot(history.history['loss'], label=f'{model_name} Train')
    plt.plot(history.history['val_loss'], linestyle='dashed', label=f'{model_name} Val')

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()
plt.show()
```

```
Training LSTM...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.3405 - loss: 1.5807 - val_accuracy: 0.3470 - val_loss: 1.5614
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.3412 - loss: 1.5752 - val_accuracy: 0.2945 - val_loss: 1.4884
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 6s 18ms/step - accuracy: 0.3316 - loss: 1.5432 - val_accuracy: 0.3475 - val_loss: 1.5611
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3381 - loss: 1.5858 - val_accuracy: 0.3475 - val_loss: 1.5612
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3372 - loss: 1.5830 - val_accuracy: 0.3475 - val_loss: 1.5600
Training GRU...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9751 - loss: 0.0586 - val_accuracy: 0.9230 - val_loss: 0.2179
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9830 - loss: 0.0447 - val_accuracy: 0.9160 - val_loss: 0.2463
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9892 - loss: 0.0314 - val_accuracy: 0.9215 - val_loss: 0.3020
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9898 - loss: 0.0306 - val_accuracy: 0.9275 - val_loss: 0.2297
Training RNN...
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6576 - loss: 0.9445 - val_accuracy: 0.5650 - val_loss: 1.2951
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6869 - loss: 0.8549 - val_accuracy: 0.5750 - val_loss: 1.3104
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7070 - loss: 0.7694 - val_accuracy: 0.5390 - val_loss: 1.3987
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.7280 - loss: 0.7109 - val_accuracy: 0.5710 - val_loss: 1.3692
```





```
! pip install datasets
```

```
Collecting datasets
  Downloading datasets-3.3.2-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets) (3.17.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: fsspec<=2024.12.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspec[http]<=2024.12.0,>=2
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets) (3.11.12)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.28.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (2.4.6)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.24.0->data
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets) (2025.1.3
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.17
Downloading datasets-3.3.2-py3-none-any.whl (485 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 485.4/485.4 kB 11.1 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 10.1 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 2.7 MB/s eta 0:00:00
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 7.3 MB/s eta 0:00:00
Installing collected packages: xxhash, dill, multiprocess, datasets
Successfully installed datasets-3.3.2 dill-0.3.8 multiprocess-0.70.16 xxhash-3.5.0
```

```
! pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.48.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.17.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.28.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.24.0->transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.24.0-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.1.31)
```

```
import tensorflow as tf
from transformers import TFAutoModel ,AutoTokenizer
from datasets import load_dataset
```

Load emotions Dataset

```
emotions = load_dataset("SetFit/emotion")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning: 
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

README.md: 100% 194/194 [00:00<00:00, 18.8kB/s]

```
Repo card metadata block was not found. Setting CardData to empty.
WARNING:huggingface_hub.repocard:Repo card metadata block was not found. Setting CardData to empty.
```

train.jsonl: 100% 2.23M/2.23M [00:00<00:00, 13.9MB/s]

validation.jsonl: 100% 276k/276k [00:00<00:00, 13.5MB/s]

test.jsonl: 100% 279k/279k [00:00<00:00, 19.5MB/s]

Generating train split: 100% 16000/16000 [00:00<00:00, 262262.84 examples/s]

Generating validation split: 100% 2000/2000 [00:00<00:00, 105412.33 examples/s]

Generating test split: 100% 2000/2000 [00:00<00:00, 69880.61 examples/s]

Initialize model and tokenizer

```
model = TFAutoModel.from_pretrained("bert-base-uncased")
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

config.json: 100% 570/570 [00:00<00:00, 50.4kB/s]

model.safetensors: 100% 440M/440M [00:02<00:00, 201MB/s]

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.seq_relationship.weight', 'cls.pre
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. i
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializi
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without
```

tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.44kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 1.68MB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 6.41MB/s]

```
emotions
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 'label_text'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label', 'label_text'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label', 'label_text'],
        num_rows: 2000
    })
})
```

create tokenize function

```
def tokenize (batch):
    return tokenizer(batch["text"] ,padding=True ,truncation=True)
```

tokenize and encode emotions dataset

```
encoded_emotions = emotions.map(tokenize, batched=True ,batch_size=None)
```

```
Map: 100%    16000/16000 [00:05<00:00, 2954.87 examples/s]
Map: 100%    2000/2000 [00:00<00:00, 4699.28 examples/s]
Map: 100%    2000/2000 [00:00<00:00, 4588.65 examples/s]
```

```
encoded_emotions
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 'label_text', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label', 'label_text', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label', 'label_text', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 2000
    })
})
```

Convert Data into TF format

```
# setting 'input_ids', 'attention_mask', 'token_type_ids', and 'label'
# to the tensorflow format. Now if you access this dataset you will get these
# columns in `tf.Tensor` format

encoded_emotions.set_format('tf',
                            columns=['input_ids', 'attention_mask', 'token_type_ids', 'label'])


ABTCH_SIZE =64

def order (inp):
    """
    This function will group all the inputs of BERT
    into a single dictionary and then output it with
    labels.

    """

    return {
        'input_ids': inp['input_ids'],
        'attention_mask': inp['attention_mask'],
        'token_type_ids': inp['token_type_ids']
    }, inp['label']

# converting train split of `emotions_encoded` to tensorflow format
train_dataset = tf.data.Dataset.from_tensor_slices(encoded_emotions['train'][:])

# set batch_size and shuffle
train_dataset = train_dataset.shuffle(1000).batch(ABTCH_SIZE)

# map the `order` function
train_dataset=train_dataset.map(order, num_parallel_calls=tf.data.AUTOTUNE)

#same thing to test_dataset
test_dataset = tf.data.Dataset.from_tensor_slices(encoded_emotions['test'][:])
test_dataset = test_dataset.batch(ABTCH_SIZE)
test_dataset=test_dataset.map(order, num_parallel_calls=tf.data.AUTOTUNE)


inp, out = next(iter(train_dataset)) # a batch from train_dataset
print(inp, '\n\n', out)
```

```
{'input_ids': <tf.Tensor: shape=(64, 87), dtype=int64, numpy=
array([[ 101, 1045, 2074, ...,    0,    0,    0],
       [ 101, 1045, 2411, ...,    0,    0,    0],
       [ 101, 1045, 4919, ...,    0,    0,    0],
       ...,
       [ 101, 1045, 2318, ...,    0,    0,    0],
       [ 101, 1045, 2079, ...,    0,    0,    0],
       [ 101, 1045, 2514, ...,    0,    0,    0]])>, 'attention_mask': <tf.Tensor: shape=(64, 87), dtype=int64, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
```

```
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0]])>, 'token_type_ids': <tf.Tensor: shape=(64, 87), dtype=int64, numpy=
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])>}

 tf.Tensor(
[0 2 4 1 1 1 1 1 4 1 0 0 1 3 1 1 1 1 3 5 0 1 0 1 1 1 3 1 1 4 0 1 3 0 2 1 1
 3 0 4 1 0 2 0 0 0 0 3 1 1 1 3 3 0 1 4 1 2 1 2 0 0 3 3], shape=(64,), dtype=int64)
```

Create Bert Classifier

```
class BertForClassification(tf.keras.Model):
  def __init__(self,model,num_classes):
    super().__init__()
    # load modeel
    self.bert = model
    # output layer
    self.fc= tf.keras.layers.Dense(num_classes , activation="softmax")

  def call(self,inputs):
    x= self.bert(inputs)[1]
    return self.fc(x)
```

compile

```
classifier = BertForClassification(model,6)

classifier.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                   metrics=['accuracy'])
```

fit model

```
history = classifier.fit(train_dataset,epochs=3)
```

```
Epoch 1/3
250/250 [==============================] - 315s 1s/step - loss: 0.9425 - accuracy: 0.6607
Epoch 2/3
250/250 [==============================] - 255s 1s/step - loss: 0.2454 - accuracy: 0.9091
Epoch 3/3
250/250 [==============================] - 254s 1s/step - loss: 0.1450 - accuracy: 0.9407
```

Evaluate model

```
classifier.evaluate(test_dataset)
```

```
32/32 [==============================] - 13s 286ms/step - loss: 0.1654 - accuracy: 0.9280
[0.16544492542743683, 0.9279999732971191]
```

```
import matplotlib.pyplot as plt

# Training data
epochs = [1, 2, 3]
train_loss = [0.9370, 0.2436, 0.1491]
train_accuracy = [0.6598, 0.9123, 0.9389]

# Test data
test_loss = 0.1678
test_accuracy = 0.9240
```

```
# Create subplots
plt.figure(figsize=(12, 5))

# Plot Loss
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'bo-', label='Training Loss')
plt.axhline(test_loss, color='r', linestyle='--', label='Test Loss')
plt.title('Training and Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.xticks(epochs)
plt.legend()

# Plot Accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_accuracy, 'go-', label='Training Accuracy')
plt.axhline(test_accuracy, color='r', linestyle='--', label='Test Accuracy'
plt.title('Training and Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.xticks(epochs)
plt.legend()

plt.tight_layout()
plt.show()
```