Import Necessary Libraries

In Keras, the IMDb dataset is often used for sentiment analysis. The main modules involved in processing the IMDb dataset for deep learning include:

keras.datasets: This module provides a collection of datasets, including the IMDb dataset. You can use imdb.load_data() to load the dataset, which returns the training and testing data split.

keras.preprocessing.sequence: This module is used for sequence data preprocessing. You can use pad_sequences() to ensure that all sequences have the same length, which is necessary for feeding data into a neural network.

keras.models: This module is used to define and compile your neural network model. Typically, a sequential model is used for simplicity in text classification tasks.

keras.layers: This module contains various layer types that can be used to build the architecture of your neural network. For text processing, the Embedding layer is often used as the first layer to convert words into dense vectors.

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
```

Loading IMDb Dataset:

The imdb.load_data() function is used to load the IMDb dataset, which consists of movie reviews labeled as positive or negative.

```
# Load IMDb dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

Data Preprocessing:

The reviews are padded or truncated to a fixed length of 500 words using sequence.pad_sequences. This ensures that all reviews have the same length for modeling.

```
# Preprocess data
max_len = 500  # Limit the reviews to 500 words
train_data = sequence.pad_sequences(train_data, maxlen=max_len)
test_data = sequence.pad_sequences(test_data, maxlen=max_len)
```

Building the Model:

Embedding Layer (layers.Embedding): This layer learns dense representations of words. It turns positive integers into fixed-size dense vectors. Flatten Layer (layers.Flatten): Flattens the 2D output from the embedding layer to a 1D array. Dense Layers (layers.Dense): Two dense layers are added. The first one has 128 units with ReLU activation, and the second one has 1 unit with Sigmoid activation for binary classification.

```
# Build the model
model = models.Sequential()
model.add(layers.Embedding(10000, 16, input_length=max_len))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Model Compilation:

The model is compiled using the Adam optimizer, which is an efficient optimization algorithm, and binary cross-entropy loss, suitable for binary classification problems.

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Model Training:

The model is trained using the model.fit() function. The training data is used, and a validation split of 20% is specified. The training occurs over 5 epochs with a batch size of 32.

```
# Train the model
model.fit(train_data, train_labels, epochs=5, batch_size=32, validation_split=0.2)
```

```
    Epoch 1/5
    625/625 [==============================] - 16s 24ms/step - loss: 0.4090 - accuracy: 0.7975 - val_loss: 0.2869 - val_accuracy: 0.880
```

```
Epoch 2/5
625/625 [==============================] - 14s 22ms/step - loss: 0.1242 - accuracy: 0.9554 - val_loss: 0.3656 - val_accuracy: 0.864
Epoch 3/5
625/625 [==============================] - 15s 23ms/step - loss: 0.0234 - accuracy: 0.9937 - val_loss: 0.4899 - val_accuracy: 0.863
Epoch 4/5
625/625 [==============================] - 15s 23ms/step - loss: 0.0033 - accuracy: 0.9996 - val_loss: 0.5733 - val_accuracy: 0.863
Epoch 5/5
625/625 [==============================] - 14s 22ms/step - loss: 5.1956e-04 - accuracy: 1.0000 - val_loss: 0.6163 - val_accuracy: 0
<keras.src.callbacks.History at 0x7cc91190f7c0>
```

Model Evaluation:

The trained model is evaluated on the test data using the model.evaluate() function. Test accuracy is printed.

```
# Evaluate the model
test_loss, test_acc = model.evaluate(test_data, test_labels)
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

```
782/782 [==============================] - 3s 4ms/step - loss: 0.6594 - accuracy: 0.8564
Test accuracy: 85.64%
```