```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

# Check the shape of the dataset
print(f"Training data shape: {x_train.shape}")
print(f"Testing data shape: {x_test.shape}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Training data shape: (25000,)
Testing data shape: (25000,)
```

```
# Pad sequences to ensure consistent length
max_len = 500
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Check the shape after padding
print(f"Shape of x_train after padding: {x_train.shape}")
print(f"Shape of x_test after padding: {x_test.shape}")
```

```
Shape of x_train after padding: (25000, 500)
Shape of x_test after padding: (25000, 500)
```

```
from tensorflow.keras import layers, models

model = models.Sequential([
    # Embedding layer to convert integer sequences to dense word vectors
    layers.Embedding(input_dim=10000, output_dim=128, input_length=max_len),

    # LSTM layer with 128 units
    layers.LSTM(128),

    # Dense layer with 1 unit and sigmoid activation for binary classification
    layers.Dense(1, activation='sigmoid')
])

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

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just
  warnings.warn(
```

```
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

```
Epoch 1/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 19s 38ms/step - accuracy: 0.6992 - loss: 0.5570 - val_accuracy: 0.6441 - val_loss: 0.6209
Epoch 2/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 14s 37ms/step - accuracy: 0.7993 - loss: 0.4264 - val_accuracy: 0.8411 - val_loss: 0.3591
Epoch 3/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 19s 33ms/step - accuracy: 0.9164 - loss: 0.2225 - val_accuracy: 0.8704 - val_loss: 0.3107
Epoch 4/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 22s 37ms/step - accuracy: 0.9350 - loss: 0.1764 - val_accuracy: 0.8470 - val_loss: 0.3658
Epoch 5/5
391/391 ━━━━━━━━━━━━━━━━━━━━ 20s 37ms/step - accuracy: 0.9516 - loss: 0.1351 - val_accuracy: 0.8643 - val_loss: 0.4210
```

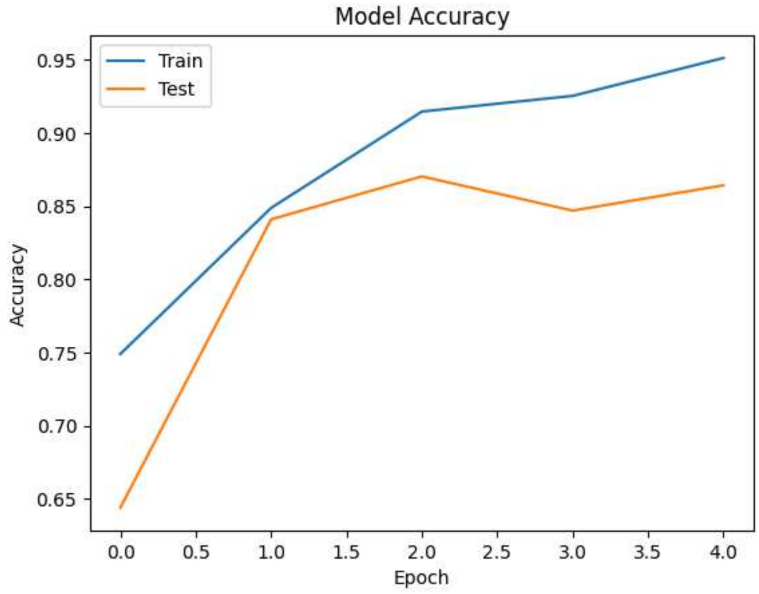
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.8626 - loss: 0.4272
Test accuracy: 0.8643199801445007
```

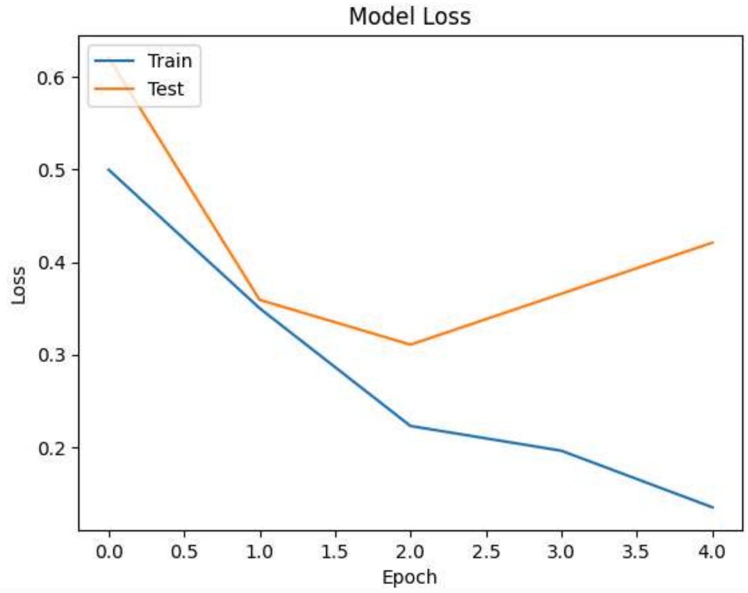
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Model Accuracy



Model Loss

```
# Example: Predict sentiment of a new review
sample_review = "This movie was fantastic, I loved it!"
# Preprocess the review (tokenize and pad the sequence)
sample_review_seq = imdb.get_word_index()
sample_review = [sample_review_seq.get(word, 0) for word in sample_review.lower().split()]
sample_review = pad_sequences([sample_review], maxlen=max_len)

# Make the prediction
prediction = model.predict(sample_review)
print(f"Predicted sentiment: {'Positive' if prediction >= 0.5 else 'Negative'}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 1s 1us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
Predicted sentiment: Negative
```