```
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from mlxtend.plotting import plot_confusion_matrix
from sklearn import metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from tqdm.notebook import tqdm
import warnings
warnings.filterwarnings("ignore")
```

```
train_data, validation_data, test_data = tfds.load(
    name="imdb_reviews",
    split=('train[:60%]', 'train[60%:]', 'test'),
    as_supervised=True)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0 h
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Un
Dl Completed...: 100% 1/1 [00:12<00:00, 12.60s/ url]
Dl Size...: 100% 80/80 [00:12<00:00, 7.45 MiB/s]
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_review
```

```
train_examples_batch, train_labels_batch = next(iter(train_data.batch(10)))
```

```
train_labels_batch
```

```
<tf.Tensor: shape=(10,), dtype=int64, numpy=array([0, 0, 0, 1, 1, 1, 0, 0, 0, 0])>
```

```
embedding = "https://tfhub.dev/google/nnlm-en-dim128-with-normalization/2"
hub_layer = hub.KerasLayer(embedding, input_shape=[],
                           dtype=tf.string, trainable=True)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda inputs: hub_layer(inputs)), # Wrap hub_layer in Lambda
    tf.keras.layers.Dense(32, activation='relu', name='hidden-layer-2'),
    tf.keras.layers.Dense(16, activation='relu', name='hidden-layer-3'),
    tf.keras.layers.Dense(1, activation='sigmoid', name='output-layer')
])
```

```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
lambda (Lambda)	?	0 (unbuilt)
hidden-layer-2 (Dense)	?	0 (unbuilt)
hidden-layer-3 (Dense)	?	0 (unbuilt)
output-layer (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

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

```
history = model.fit(train_data.shuffle(10000).batch(512),
                    epochs=5,
                    validation_data=validation_data.batch(512),
                    verbose=1)
```

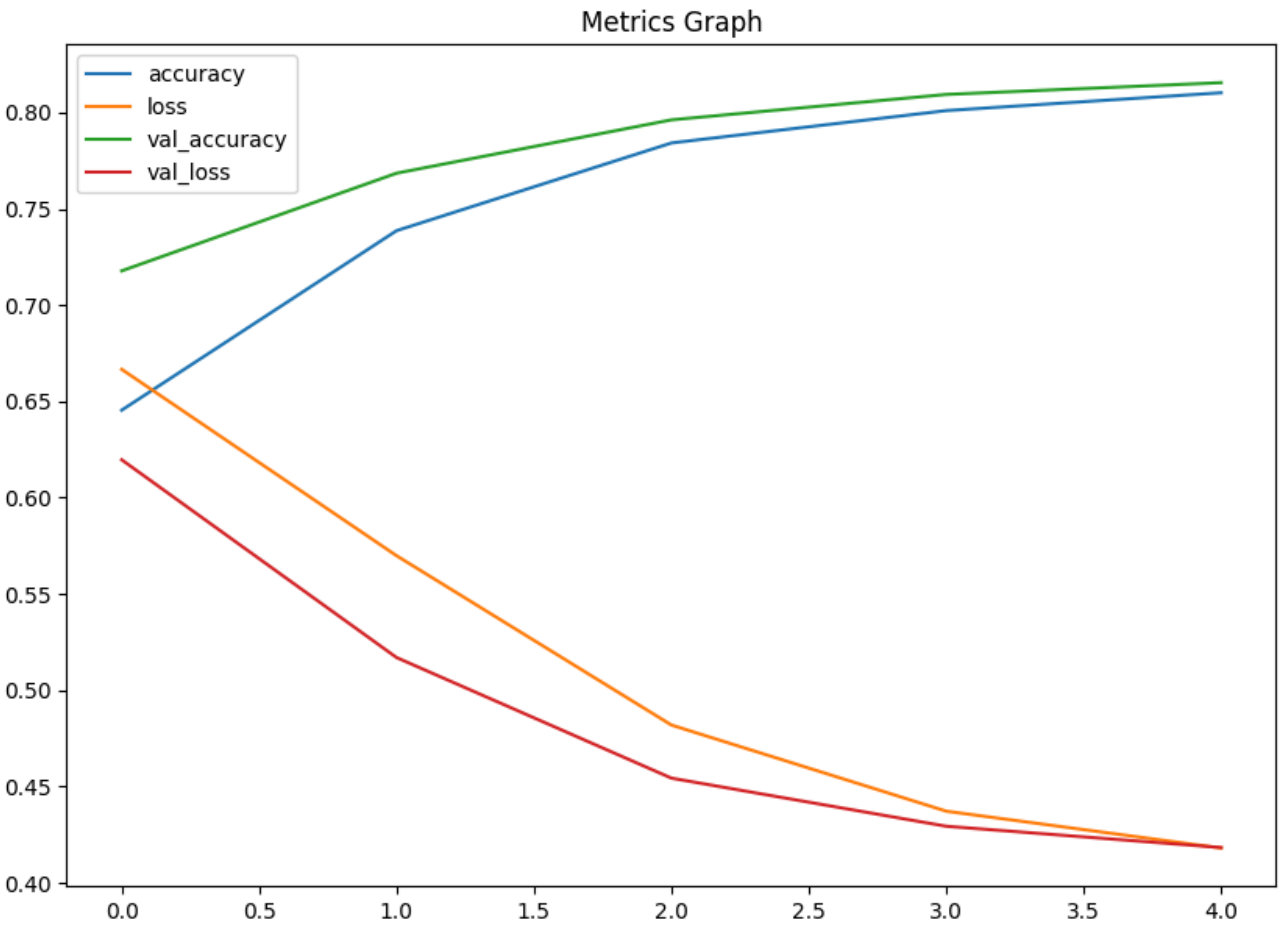
```
Epoch 1/5
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 92ms/step - accuracy: 0.5785 - loss: 0.6821 - val_accur
Epoch 2/5
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 68ms/step - accuracy: 0.7250 - loss: 0.5925 - val_accur
Epoch 3/5
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 102ms/step - accuracy: 0.7833 - loss: 0.4941 - val_accu
Epoch 4/5
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.7996 - loss: 0.4440 - val_accur
Epoch 5/5
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.8102 - loss: 0.4221 - val_accur
```

```
results = model.evaluate(test_data.batch(512), verbose=2)

for name, value in zip(model.metrics_names, results):
  print("%s: %.3f" % (name, value))
```

```
49/49 - 3s - 56ms/step - accuracy: 0.8075 - loss: 0.4264
loss: 0.426
compile_metrics: 0.807
```

```
pd.DataFrame(history.history).plot(figsize=(10,7))
plt.title("Metrics Graph")
plt.show()
```




```
texts = []
true_labels = []
for text, label in test_data:
    texts.append(text.numpy())
    true_labels.append(label.numpy())
texts = np.array(texts)
true_labels = np.array(true_labels)
```

```
predicted_probs = model.predict(tf.data.Dataset.from_tensor_slices(texts).batch(512))
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step
```

```
predicted_labels = (predicted_probs > 0.5).astype(int)
```

```
report = metrics.classification_report(true_labels, predicted_labels, target_names=['Nega
print(report)
```

```
              precision    recall  f1-score   support
```

```
              Negative       0.78      0.85      0.82     12500
              Positive       0.84      0.77      0.80     12500

              accuracy                           0.81     25000
             macro avg       0.81      0.81      0.81     25000
          weighted avg       0.81      0.81      0.81     25000
```

```
cm = metrics.confusion_matrix(true_labels, predicted_labels)
plot_confusion_matrix(cm, class_names=['Negative', 'Positive'])
plt.title("Confusion Matrix")
plt.show()
```