6. Implement Bidirectional LSTM for sentimental analysis on movie reviews.

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=20000)
x_train, x_test = pad_sequences(x_train, maxlen=200), pad_sequences(x_test, maxlen=200)
model = Sequential([
Embedding(20000, 256, input_length=200),
Bidirectional(LSTM(256, return_sequences=True)), Dropout(0.5),
Bidirectional(LSTM(128)), Dropout(0.5),
Dense(64, activation='relu'), Dropout(0.5),
Dense(1, activation='sigmoid')
])
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2) loss,
accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

OUTPUT:

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated.
  warnings.warn(
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 28s 78ms/step - accuracy: 0.6350 - loss: 0.6264 - val_accuracy: 0.7896 - val_loss: 0.4826
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.8249 - loss: 0.4289 - val_accuracy: 0.8170 - val_loss: 0.4415
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.8633 - loss: 0.3388 - val_accuracy: 0.8528 - val_loss: 0.3384

Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 78ms/step - accuracy: 0.9273 - loss: 0.2080 - val_accuracy: 0.8650 - val_loss: 0.3766
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 78ms/step - accuracy: 0.9493 - loss: 0.1516 - val_accuracy: 0.8706 - val_loss: 0.3916
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 42s 82ms/step - accuracy: 0.9690 - loss: 0.1007 - val_accuracy: 0.8490 - val_loss: 0.4522
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 79ms/step - accuracy: 0.9794 - loss: 0.0733 - val_accuracy: 0.8572 - val_loss: 0.6088
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 79ms/step - accuracy: 0.9847 - loss: 0.0521 - val_accuracy: 0.8668 - val_loss: 0.5456
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 79ms/step - accuracy: 0.9921 - loss: 0.0299 - val_accuracy: 0.8666 - val_loss: 0.5461
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 80ms/step - accuracy: 0.9943 - loss: 0.0244 - val_accuracy: 0.8642 - val_loss: 0.6577
782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 18ms/step - accuracy: 0.8519 - loss: 0.7082
Test Loss: 0.6862
Test Accuracy: 85.44%
```

7. Implement the standard VGG-16 & 19 CNN architecture model to classify multi category image dataset and check the accuracy

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16, VGG19
from tensorflow.keras import layers, models
import tensorflow_datasets as tfds
def preprocess_image(image, label):
image = tf.image.resize(image, (224, 224)) / 255.0 return image, tf.one_hot(label, depth=5)
(ds_train, ds_test), ds_info = tfds.load( 'tf_flowers',
split=['train[:80%]', 'train[80%:]'], as_supervised=True,
with_info=True
)
ds_train = ds_train.map(preprocess_image).batch(32).prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(preprocess_image).batch(32).prefetch(tf.data.AUTOTUNE)
def create_vgg_model(vgg_model_class):
base_model = vgg_model_class(weights='imagenet', include_top=False, input_shape=(224, 224,
3))
```

```
 base_model.trainable = False
return models.Sequential([ base_model,
layers.GlobalAveragePooling2D(),
layers.Dense(256, activation='relu'), layers.Dropout(0.5),
layers.Dense(5, activation='softmax')
])
def train_and_evaluate(model_class, model_name): print(f"Using {model_name}:")
model = create_vgg_model(model_class)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(ds_train, validation_data=ds_test, epochs=10) loss, acc = model.evaluate(ds_test)
print(f"{model_name} Accuracy: {acc:.2f}%") return model
vgg16_model = train_and_evaluate(VGG16, "VGG16")

vgg19_model = train_and_evaluate(VGG19, "VGG19")
```

OUTPUT:

```
Using VGG16: Epoch
     1/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 20s 195ms/step - accuracy: 0.2091 - loss: 1.7919 - val_accuracy: 0.4564 - val_loss: 1.4640
     Epoch 2/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step - accuracy: 0.3537 - loss: 1.5023 - val_accuracy: 0.5817 - val_loss: 1.3103
     Epoch 3/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 20s 179ms/step - accuracy: 0.4715 - loss: 1.3378 - val_accuracy: 0.6226 - val_loss: 1.1893
     Epoch 4/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 17s 180ms/step - accuracy: 0.5498 - loss: 1.2159 - val_accuracy: 0.6485 - val_loss: 1.1004
     Epoch 5/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 20s 180ms/step - accuracy: 0.6079 - loss: 1.1094 - val_accuracy: 0.6717 - val_loss: 1.0264
     Epoch 6/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 20s 180ms/step - accuracy: 0.6315 - loss: 1.0551 - val_accuracy: 0.6907 - val_loss: 0.9656
     Epoch 7/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 17s 181ms/step - accuracy: 0.6620 - loss: 0.9877 - val_accuracy: 0.7003 - val_loss: 0.9186
     Epoch 8/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 20s 181ms/step - accuracy: 0.6845 - loss: 0.9476 - val_accuracy: 0.7193 - val_loss: 0.8751
     Epoch 9/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 18s 156ms/step - accuracy: 0.6928 - loss: 0.9032 - val_accuracy: 0.7221 - val_loss: 0.8385
     Epoch 10/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 20s 156ms/step - accuracy: 0.6976 - loss: 0.8818 - val_accuracy: 0.7302 - val_loss: 0.8096
     23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 123ms/step - accuracy: 0.7241 - loss: 0.8143 VGG16
     Accuracy: 0.73%
     Using VGG19:
     Epoch 1/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 23s 224ms/step - accuracy: 0.2323 - loss: 1.7020 - val_accuracy: 0.4823 - val_loss: 1.4538
     Epoch 2/10
     92/92 ━━━━━━━━━━━━━━━━━━━━ 38s 205ms/step - accuracy: 0.3480 - loss: 1.5013 - val_accuracy: 0.5872 - val_loss: 1.3130
     Epoch 3/10
```