In [2]:

```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50, VGG16
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

In [3]:

```python
# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize images to the range [0, 1]
x_train = x_train / 255.0
x_test = x_test / 255.0

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 14s 0us/step
```

In [4]:

```python
# Check the data shapes
print(f"x_train shape: {x_train.shape}, y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape}, y_test shape: {y_test.shape}")
```

```
x_train shape: (50000, 32, 32, 3), y_train shape: (50000, 10)
x_test shape: (10000, 32, 32, 3), y_test shape: (10000, 10)
```

In [5]:

```python
# Define the model creation function
def create_model(base_model, num_classes):
    base_model.trainable = False  # Freeze the base model layers
    inputs = tf.keras.Input(shape=(32, 32, 3))
    x = base_model(inputs, training=False)  # Pass the input through the base model
    x = GlobalAveragePooling2D()(x)  # Reduce the spatial dimensions
    x = Dropout(0.5)(x)  # Dropout layer to avoid overfitting
    outputs = Dense(num_classes, activation='softmax')(x)  # Final classification layer
    model = Model(inputs, outputs)
    return model
```

In [6]:

```python
# Load pre-trained ResNet50 and VGG16 without the top layers
resnet_base = ResNet50(weights='imagenet', include_top=False)
vgg16_base = VGG16(weights='imagenet', include_top=False)

# Create models using the pre-trained base models
resnet_model = create_model(resnet_base, num_classes=10)
vgg16_model = create_model(vgg16_base, num_classes=10)

# Compile the models
optimizer = Adam(learning_rate=0.001)

resnet_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
vgg16_model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_
tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_k
ernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

[7]:

```python
# Load pre-trained ResNet50 and VGG16 without the top layers
resnet_base = ResNet50(weights='imagenet', include_top=False)
vgg16_base = VGG16(weights='imagenet', include_top=False)

# Create models using the pre-trained base models
resnet_model = create_model(resnet_base, num_classes=10)
vgg16_model = create_model(vgg16_base, num_classes=10)

# Compile the models
# Moved optimizer initialization AFTER model creation
optimizer_resnet = Adam(learning_rate=0.001)  # Optimizer for ResNet
optimizer_vgg16 = Adam(learning_rate=0.001)  # Optimizer for VGG16


resnet_model.compile(optimizer=optimizer_resnet, loss='categorical_crossentropy', metrics=['accuracy'])
vgg16_model.compile(optimizer=optimizer_vgg16, loss='categorical_crossentropy', metrics=['accuracy']) # Use separate opti
```

In [8]:

```python
# Train the ResNet-50 Model
print("Training ResNet-50 Model...")
resnet_history = resnet_model.fit(
    x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test), verbose=1
)

# Train the VGG16 Model
print("Training VGG16 Model...")
vgg16_history = vgg16_model.fit(
    x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test), verbose=1
)
```

```
Training ResNet-50 Model...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 32s 14ms/step - accuracy: 0.1541 - loss: 2.5710 - val_accuracy: 0.2833 - val_loss: 2.0171
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 27s 8ms/step - accuracy: 0.2129 - loss: 2.1540 - val_accuracy: 0.2890 - val_loss: 1.9838
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.2217 - loss: 2.1305 - val_accuracy: 0.2687 - val_loss: 2.0090
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 14s 9ms/step - accuracy: 0.2236 - loss: 2.1314 - val_accuracy: 0.2786 - val_loss: 2.0314
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.2268 - loss: 2.1303 - val_accuracy: 0.2708 - val_loss: 1.9817
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.2255 - loss: 2.1269 - val_accuracy: 0.2811 - val_loss: 1.9788
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.2308 - loss: 2.1173 - val_accuracy: 0.2974 - val_loss: 1.9554
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.2304 - loss: 2.1195 - val_accuracy: 0.3055 - val_loss: 1.9346
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 14s 9ms/step - accuracy: 0.2281 - loss: 2.1240 - val_accuracy: 0.2619 - val_loss: 2.0133
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.2277 - loss: 2.1218 - val_accuracy: 0.2820 - val_loss: 1.9647
Training VGG16 Model...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - accuracy: 0.3136 - loss: 1.9472 - val_accuracy: 0.5057 - val_loss: 1.4683
Epoch 2/10
```

```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.4577 - loss: 1.5586 - val_accuracy: 0.5180 - val_loss: 1.4155
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.4663 - loss: 1.5291 - val_accuracy: 0.5297 - val_loss: 1.3907
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.4746 - loss: 1.5181 - val_accuracy: 0.5264 - val_loss: 1.3804
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.4707 - loss: 1.5151 - val_accuracy: 0.5341 - val_loss: 1.3698
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.4759 - loss: 1.5106 - val_accuracy: 0.5321 - val_loss: 1.3686
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.4783 - loss: 1.4975 - val_accuracy: 0.5382 - val_loss: 1.3632
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.4812 - loss: 1.4981 - val_accuracy: 0.5416 - val_loss: 1.3632
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.4763 - loss: 1.4920 - val_accuracy: 0.5377 - val_loss: 1.3613
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.4723 - loss: 1.5118 - val_accuracy: 0.5402 - val_loss: 1.3540
```

[11]:

```
def fine_tune_model(model, base_model):
    base_model.trainable = True  # Unfreeze the base model layers
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Fine-tune the models
resnet_model = fine_tune_model(resnet_model, resnet_base)
vgg16_model = fine_tune_model(vgg16_model, vgg16_base)

# Fine-tune for a few more epochs
print("Fine-tuning ResNet-50 Model...")
resnet_fine_history = resnet_model.fit(
    x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test), verbose=1
)

print("Fine-tuning VGG16 Model...")
vgg16_fine_history = vgg16_model.fit(
    x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test, y_test), verbose=1
)
```

```
Fine-tuning ResNet-50 Model...
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 116s 43ms/step - accuracy: 0.7406 - loss: 0.9052 - val_accuracy: 0.7229 - val_loss: 0.8016
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 48s 31ms/step - accuracy: 0.7453 - loss: 1.0101 - val_accuracy: 0.7870 - val_loss: 0.6477
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 30ms/step - accuracy: 0.8008 - loss: 0.6762 - val_accuracy: 0.7655 - val_loss: 1.3357
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.8158 - loss: 0.6289 - val_accuracy: 0.7982 - val_loss: 0.6501
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.8321 - loss: 0.5530 - val_accuracy: 0.6706 - val_loss: 3.7838
Fine-tuning VGG16 Model...
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 63s 37ms/step - accuracy: 0.9535 - loss: 0.1477 - val_accuracy: 0.8550 - val_loss: 0.5043
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 34ms/step - accuracy: 0.9667 - loss: 0.1021 - val_accuracy: 0.8487 - val_loss: 0.6144
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 35ms/step - accuracy: 0.9729 - loss: 0.0891 - val_accuracy: 0.8429 - val_loss: 0.6579
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 35ms/step - accuracy: 0.9762 - loss: 0.0747 - val_accuracy: 0.8520 - val_loss: 0.6302
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 35ms/step - accuracy: 0.9799 - loss: 0.0664 - val_accuracy: 0.8603 - val_loss: 0.6347
```

In [14]:

```python
# Evaluate ResNet-50 Model
print("Evaluating ResNet-50 Model...")
resnet_eval = resnet_model.evaluate(x_test, y_test, verbose=1)

# Evaluate VGG16 Model
print("Evaluating VGG16 Model...")
vgg16_eval = vgg16_model.evaluate(x_test, y_test, verbose=1)

# Classification Report for ResNet-50
y_pred_resnet = tf.argmax(resnet_model.predict(x_test), axis=-1).numpy()
# Convert y_test to the same format as y_pred_resnet (multiclass)
y_test_classes = tf.argmax(y_test, axis=-1).numpy()  # Assuming y_test is one-hot encoded
print("ResNet-50 Classification Report:")
print(classification_report(y_test_classes, y_pred_resnet)) # Use y_test_classes

# Classification Report for VGG16
y_pred_vgg16 = tf.argmax(vgg16_model.predict(x_test), axis=-1).numpy()
# Convert y_test to the same format as y_pred_vgg16 (multiclass)
y_test_classes = tf.argmax(y_test, axis=-1).numpy()  # Assuming y_test is one-hot encoded
print("VGG16 Classification Report:")
print(classification_report(y_test_classes, y_pred_vgg16)) # Use y_test_classes
```

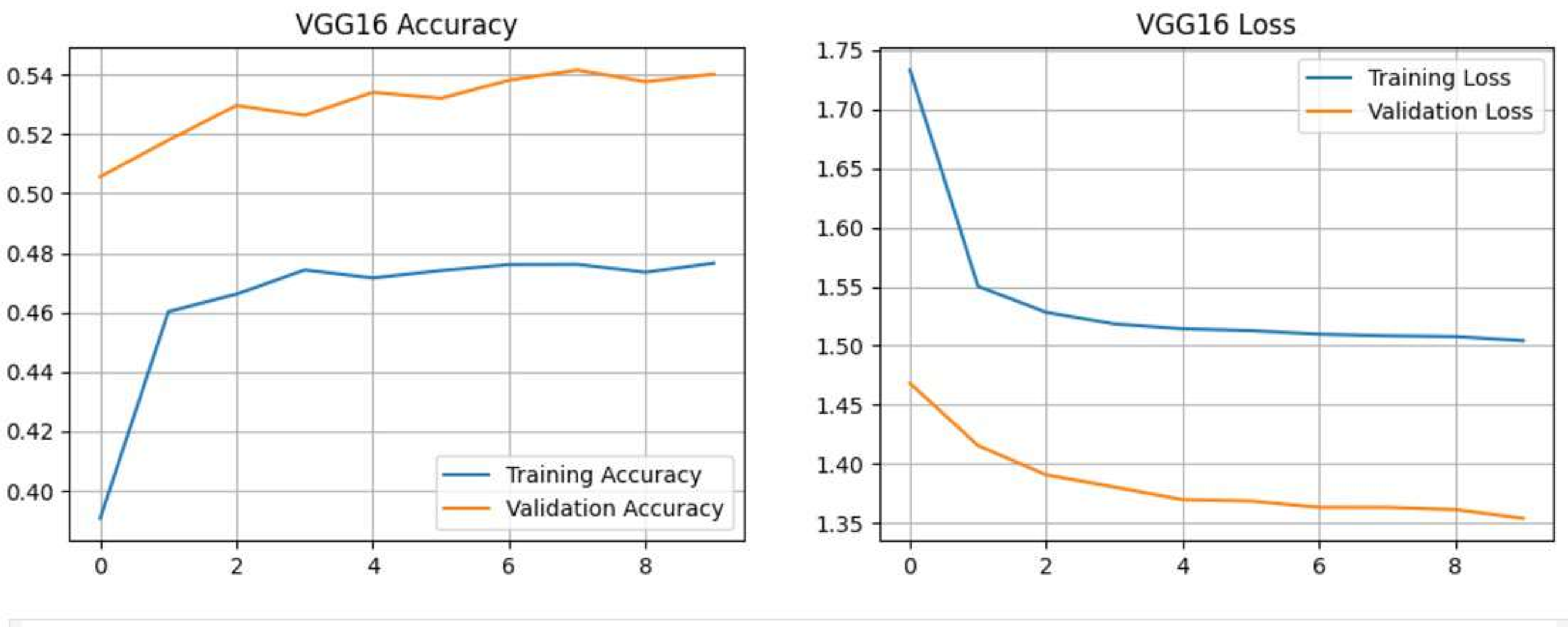
```
Evaluating ResNet-50 Model...
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.6719 - loss: 3.7679
Evaluating VGG16 Model...
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8631 - loss: 0.6249
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
ResNet-50 Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.81      0.75      1000
           1       0.56      0.43      0.49      1000
           2       0.69      0.67      0.68      1000
           3       0.58      0.54      0.56      1000
           4       0.68      0.69      0.68      1000
           5       0.70      0.63      0.66      1000
           6       0.75      0.84      0.79      1000
           7       0.71      0.56      0.63      1000
           8       0.88      0.76      0.81      1000
           9       0.53      0.78      0.63      1000

    accuracy                           0.67     10000
   macro avg       0.68      0.67      0.67     10000
weighted avg       0.68      0.67      0.67     10000

313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step
VGG16 Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.93      0.89      1000
           1       0.87      0.97      0.92      1000
           2       0.86      0.79      0.83      1000
           3       0.71      0.72      0.72      1000
           4       0.86      0.85      0.85      1000
           5       0.77      0.81      0.79      1000
           6       0.93      0.87      0.90      1000
           7       0.87      0.92      0.90      1000
           8       0.96      0.90      0.93      1000
           9       0.94      0.85      0.89      1000
```

```
    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

In [15]:

```
def plot_curves(history, title):
    plt.figure(figsize=(12, 4))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{title} Accuracy')
    plt.legend()
    plt.grid()

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{title} Loss')
    plt.legend()
    plt.grid()

    plt.show()

# Plot curves for ResNet-50 and VGG16
plot_curves(resnet_history, "ResNet-50")
plot_curves(vgg16_history, "VGG16")
```


ResNet-50 Accuracy
Training Accuracy
Validation Accuracy
0.30
0.28
0.26
0.24
0.22
0.20
0.18
0
2
4
6
8
ResNet-50 Loss
Training Loss
Validation Loss
2.30
2.25
2.20
2.15
2.10
2.05
2.00
1.95
0
2
4
6
8



VGG16 Accuracy
0.54
0.52
0.50
0.48
0.46
0.44
0.42
0.40
0
2
4
6
8
Training Accuracy
Validation Accuracy
VGG16 Loss
1.75
1.70
1.65
1.60
1.55
1.50
1.45
1.40
1.35
0
2
4
6
8
Training Loss
Validation Loss