```
# Cell 1: Import libraries
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Input, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import numpy as np
```

```
# Cell 2: Load the pretrained VGG16 base model
input_shape = (224, 224, 3)
num_classes = 5  # Change as per your dataset

base_model = VGG16(
    weights='imagenet',
    include_top=False,
    input_shape=input_shape
)

print("Base model loaded with output shape:", base_model.output_shape)
```

```
Base model loaded with output shape: (None, 7, 7, 512)
```

```
# Cell 3: Freeze all layers initially
for layer in base_model.layers:
    layer.trainable = False

print(f"Trainable layers after freezing: {len([l for l in base_model.layers if l.trainable])}")
```

```
Trainable layers after freezing: 0
```

```
# Cell 4: Add new fully-connected layers for classification
x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output_layer = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=output_layer)
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6,422,784
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 5)	1,285

Total params: 21,138,757 (80.64 MB)
Trainable params: 6,424,069 (24.51 MB)
Non-trainable params: 14,714,688 (56.13 MB)

```
# Cell 5: Compile for initial training
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# Cell 6: Prepare training and validation data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    'data/val',
    target_size=(224, 224),
    batch_size=32
```

```
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 50 images belonging to 5 classes.
Found 50 images belonging to 5 classes.
```

```
# Cell 7: Train only the newly added dense layers
callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ModelCheckpoint('vgg16_base_model.h5', save_best_only=True)
]

history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=callbacks
)
```

```
C:\Users\Parth\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py
  self._warn_if_super_not_called()
Epoch 1/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.1894 - loss: 3.8146  WARNING:absl:You are saving your model as an HDF5 file v
2/2 ━━━━━━━━━━━━━━━━━━━━ 80s 41s/step - accuracy: 0.1600 - loss: 5.5499 - val_accuracy: 0.2200 - val_loss: 3.7014
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.3519 - loss: 4.5341  WARNING:absl:You are saving your model as an HDF5 file v
2/2 ━━━━━━━━━━━━━━━━━━━━ 38s 25s/step - accuracy: 0.3600 - loss: 4.5018 - val_accuracy: 0.2000 - val_loss: 3.0742
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.1425 - loss: 5.2970  WARNING:absl:You are saving your model as an HDF5 file v
2/2 ━━━━━━━━━━━━━━━━━━━━ 36s 25s/step - accuracy: 0.1600 - loss: 5.0133 - val_accuracy: 0.2000 - val_loss: 2.1867
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.1838 - loss: 3.3972  WARNING:absl:You are saving your model as an HDF5 file v
2/2 ━━━━━━━━━━━━━━━━━━━━ 34s 23s/step - accuracy: 0.1800 - loss: 3.3460 - val_accuracy: 0.2000 - val_loss: 1.7937
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 31s 25s/step - accuracy: 0.2000 - loss: 2.9597 - val_accuracy: 0.2400 - val_loss: 1.9667
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 30s 25s/step - accuracy: 0.1200 - loss: 3.2664 - val_accuracy: 0.2000 - val_loss: 2.0243
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.1681 - loss: 2.9735  WARNING:absl:You are saving your model as an HDF5 file v
2/2 ━━━━━━━━━━━━━━━━━━━━ 40s 30s/step - accuracy: 0.1800 - loss: 2.7686 - val_accuracy: 0.2800 - val_loss: 1.6716
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.1425 - loss: 2.3911  WARNING:absl:You are saving your model as an HDF5 file v
2/2 ━━━━━━━━━━━━━━━━━━━━ 34s 23s/step - accuracy: 0.1600 - loss: 2.3660 - val_accuracy: 0.2200 - val_loss: 1.6450
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 31s 25s/step - accuracy: 0.2200 - loss: 1.9903 - val_accuracy: 0.2000 - val_loss: 1.6713
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 31s 25s/step - accuracy: 0.2200 - loss: 1.7283 - val_accuracy: 0.2400 - val_loss: 1.6652
```

```
# Cell 8: Unfreeze last block (block5) for fine-tuning
set_trainable = False
for layer in base_model.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    layer.trainable = set_trainable

print("Trainable layers after unfreezing block 5:")
for l in base_model.layers:
    if l.trainable:
        print(l.name)
```

```
Trainable layers after unfreezing block 5:
block5_conv1
block5_conv2
block5_conv3
block5_pool
```

```
# Cell 9: Recompile with a much lower learning rate
model.compile(
    optimizer=Adam(learning_rate=0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# Cell 10: Continue training (fine-tuning)
callbacks_finetune = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ModelCheckpoint('vgg16_finetuned_model.keras', save_best_only=True)
]
```

```
history_finetune = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=callbacks_finetune
)
```

```
Epoch 1/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 128s 79s/step - accuracy: 0.1200 - loss: 1.8019 - val_accuracy: 0.1800 - val_loss: 1.6228
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 57s 39s/step - accuracy: 0.2000 - loss: 1.8541 - val_accuracy: 0.2000 - val_loss: 1.6215
Epoch 3/10
```

```
# Cell 11: Evaluate fine-tuned model on validation data
val_loss, val_acc = model.evaluate(validation_generator)
print(f"Validation Accuracy: {val_acc * 100:.2f}%")
```