```
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

base_model = keras.applications.VGG16(
    weights='imagenet',
    input_shape=(224, 224, 3),
    include_top=False)
```

```
# Freeze base model
base_model.trainable = False
```

```
# Create inputs with correct shape
inputs = keras.Input(shape=(224, 224, 3))

x = base_model(inputs, training=False)

# Add pooling layer or flatten layer
x = keras.layers.GlobalAveragePooling2D()(x)

# Add final dense layer
outputs = keras.layers.Dense(6, activation = 'softmax')(x)

# Combine inputs and outputs to create model
model = keras.Model(inputs,outputs)
```

```
model.summary()
```

```
model.compile(loss = keras.losses.CategoricalCrossentropy(from_logits=True) , metrics =
keras.metrics.CategoricalAccuracy())
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(samplewise_center=True,rotation_range=10,
zoom_range=0.1,width_shift_range=0.1,height_shift_range=0.1,horizontal_flip=True,vertical_flip
=False)
```

```
# load and iterate training dataset
train_it = datagen.flow_from_directory('data/fruits/train/',
                                       target_size=(224,224),
                                       color_mode='rgb',
                                       class_mode="categorical",
                                      batch_size=8)
# load and iterate validation dataset
```

```
valid_it = datagen.flow_from_directory('data/fruits/valid/',
                                       target_size=(224,224),
                                       color_mode='rgb',
                                       class_mode="categorical",
                                       batch_size=8)
```

```
model.fit(train_it,
        validation_data=valid_it,
        steps_per_epoch=train_it.samples/train_it.batch_size,
        validation_steps=valid_it.samples/valid_it.batch_size,
        epochs=20)
```

```
# Unfreeze the base model
base_model.trainable = True

# Compile the model with a low learning rate
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate = 1e-5),
            loss = keras.losses.BinaryCrossentropy(from_logits=True) , metrics =
[keras.metrics.BinaryAccuracy()])
```

```
# Unfreeze the base model
base_model.trainable = True

# Compile the model with a low learning rate
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate = 1e-5),
            loss = keras.losses.BinaryCrossentropy(from_logits=True) , metrics =
[keras.metrics.BinaryAccuracy()])
```

```
model.fit(train_it,
        validation_data=valid_it,
        steps_per_epoch=train_it.samples/train_it.batch_size,
        validation_steps=valid_it.samples/valid_it.batch_size,
        epochs=20)
```

```
model.evaluate(valid_it, steps=valid_it.samples/valid_it.batch_size)
```

```
from run_assessment import run_assessment
```

```
run_assessment(model, valid_it)
```