```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from sklearn.model_selection import KFold, StratifiedKFold, cross_val_score
```

```
import pathlib

dataset_url = "https://storage.googleapis.com/example_imagess/train.tar.gz"
data_dir = tf.keras.utils.get_file('train.tar', origin=dataset_url, extract=True)
data_dir = pathlib.Path(data_dir).with_suffix('')
```

```
Downloading data from https://storage.googleapis.com/example_imagess/train.tar.gz
611809929/611809929 [==============================] - 3s 0us/step
```

```
image_count = len(list(data_dir.glob('*/*.JPG')))
print(image_count)
```

```
36852
```

```
import pathlib
import PIL
import tensorflow as tf

dataset_url = "https://storage.googleapis.com/example_imagess/train.tar.gz"
data_dir = tf.keras.utils.get_file('train.tar', origin=dataset_url, extract=True)
data_dir = pathlib.Path(data_dir).with_suffix('')

# 1. Print data_dir to verify the extraction path:
print(f"Data directory: {data_dir}")

# 2. List all files and directories in data_dir:
print(f"Files and directories in data_dir:")
for item in data_dir.iterdir():
    print(item)

# 3. Use a more general glob pattern:
Blueberry___healthy = list(data_dir.glob('**/*.JPG'))  # Search recursively for JPG files

# 4. Check if any files were found:
if Blueberry___healthy:
    # If files were found, open the first one
    PIL.Image.open(str(Blueberry___healthy[0]))
else:
    # If no files were found, print an error message
    print("No image files found in the specified directory.")
```

```
Data directory: /root/.keras/datasets/train
Files and directories in data_dir:
/root/.keras/datasets/train/Potato___Early_blight
/root/.keras/datasets/train/Strawberry___healthy
/root/.keras/datasets/train/Cherry_(including_sour)___healthy
/root/.keras/datasets/train/Apple___healthy
/root/.keras/datasets/train/Pepper,_bell___healthy
/root/.keras/datasets/train/Corn_(maize)___healthy
/root/.keras/datasets/train/Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot
/root/.keras/datasets/train/Soybean___healthy
/root/.keras/datasets/train/Squash___Powdery_mildew
/root/.keras/datasets/train/Potato___healthy
/root/.keras/datasets/train/Apple___Black_rot
/root/.keras/datasets/train/Raspberry___healthy
/root/.keras/datasets/train/Apple___Apple_scab
/root/.keras/datasets/train/Blueberry___healthy
/root/.keras/datasets/train/Pepper,_bell___Bacterial_spot
/root/.keras/datasets/train/Cherry_(including_sour)___Powdery_mildew
/root/.keras/datasets/train/Corn_(maize)___Common_rust_
/root/.keras/datasets/train/Apple___Cedar_apple_rust
/root/.keras/datasets/train/Potato___Late_blight
/root/.keras/datasets/train/Strawberry___Leaf_scorch
/root/.keras/datasets/train/Corn_(maize)___Northern_Leaf_Blight
```

```
Apple___healthy = list(data_dir.glob('Apple___healthy/*'))  # Remove leading slash for relative path
PIL.Image.open(str(Apple___healthy[0]))
```

```
PIL.Image.open(str(Apple___healthy[1]))
```

```
Apple_scrab = list(data_dir.glob('Apple___Apple_scab/*'))
PIL.Image.open(str(Apple_scrab[0]))
```

```
PIL.Image.open(str(Apple_scrab[1]))
```

```
from PIL import Image

# Instead of PIL.Image.shape(str(tulips[1])), use the following:
img = Image.open(str(Apple_scrab[1]))
img_shape = img.size  # Get the image size (width, height)

print(img_shape)
```

```
(256, 256)
```

```
batch_size = 64
img_height = 180
img_width = 180
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 39152 files belonging to 21 classes.
Using 31322 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

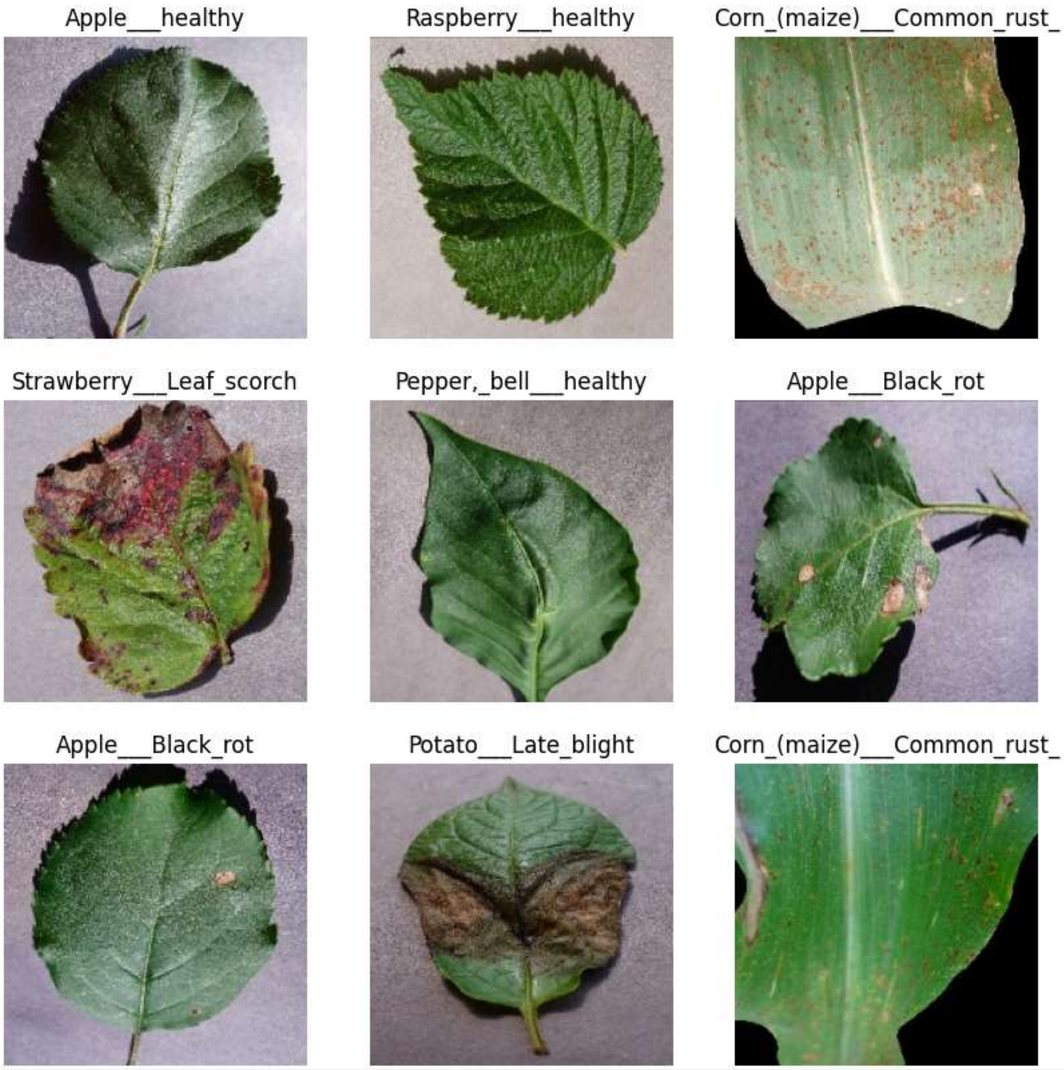
```
Found 39152 files belonging to 21 classes.
Using 7830 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy', 'Cherry_(including_sou
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```




```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(64, 180, 180, 3)
(64,)
```

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
normalization_layer = layers.Rescaling(1./255)
```

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 0.98651874
```

```
num_classes = len(class_names)
num_classes
```

```
21
```

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2  (None, 90, 90, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 21)                2709

=================================================================
Total params: 3991349 (15.23 MB)
Trainable params: 3991349 (15.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs=15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/15
490/490 [==============================] - 107s 217ms/step - loss: 0.8614 - accuracy: 0.7318 - val_loss: 0.2893 - val_accuracy: 0.9072
Epoch 2/15
490/490 [==============================] - 102s 208ms/step - loss: 0.2561 - accuracy: 0.9150 - val_loss: 0.2586 - val_accuracy: 0.9124
Epoch 3/15
490/490 [==============================] - 101s 206ms/step - loss: 0.1465 - accuracy: 0.9516 - val_loss: 0.2206 - val_accuracy: 0.9250
Epoch 4/15
490/490 [==============================] - 102s 208ms/step - loss: 0.0920 - accuracy: 0.9682 - val_loss: 0.1625 - val_accuracy: 0.9458
Epoch 5/15
490/490 [==============================] - 102s 209ms/step - loss: 0.0608 - accuracy: 0.9797 - val_loss: 0.2006 - val_accuracy: 0.9384
Epoch 6/15
490/490 [==============================] - 103s 210ms/step - loss: 0.0563 - accuracy: 0.9812 - val_loss: 0.1333 - val_accuracy: 0.9596
Epoch 7/15
490/490 [==============================] - 103s 211ms/step - loss: 0.0493 - accuracy: 0.9841 - val_loss: 0.2044 - val_accuracy: 0.9391
Epoch 8/15
490/490 [==============================] - 102s 209ms/step - loss: 0.0315 - accuracy: 0.9901 - val_loss: 0.1825 - val_accuracy: 0.9479
Epoch 9/15
490/490 [==============================] - 104s 211ms/step - loss: 0.0370 - accuracy: 0.9875 - val_loss: 0.1806 - val_accuracy: 0.9484
Epoch 10/15
490/490 [==============================] - 103s 210ms/step - loss: 0.0351 - accuracy: 0.9880 - val_loss: 0.2650 - val_accuracy: 0.9384
```

```
Epoch 11/15
230/490 [=============>................] - ETA: 51s - loss: 0.0228 - accuracy: 0.9925
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                  img_width,
                                  3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

```
num_classes = len(class_names)
num_classes
```

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Updated code with Conv Layer Output Shapes !!!

```
epochs = 15
history = model.fit(
```

```
  train_ds,
  validation_data=val_ds
  epochs=epochs


acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()


Apple_url = "https://storage.googleapis.com/example_imagess/AppleCedarRust4.JPG"

# Download the image and load it
Apple_path = tf.keras.utils.get_file('test', origin=Apple_url)
img = tf.keras.utils.load_img(Apple_path, target_size=(img_height, img_width))

# Convert the image to an array and expand dimensions to create a batch
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, axis=0)  # Create a batch of size 1

# Make predictions
predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

# Print the result
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)


Apple_url = "https://storage.googleapis.com/example_imagess/00a6039c-e425-4f7d-81b1-d6b0e668517e___RS_HL%207669.JPG"

# Download the image and load it
Apple_path = tf.keras.utils.get_file('Apple_healthy', origin=Apple_url)
img = tf.keras.utils.load_img(Apple_path, target_size=(img_height, img_width))

# Convert the image to an array and expand dimensions to create a batch
img_array = tf.keras.utils.img_to_array(img)
```