```
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import keras
print(keras.__version__)
```

```
3.5.0
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
from tensorflow.keras import models,layers
import matplotlib.pyplot as plt
```

```
BATCH_SIZE=32
IMAGE_SIZE=640
CHANNELS=3
EPOCHS=50
```

```
data1=pd.read_csv('/content/drive/MyDrive/Disease dataset/test/_classes.csv')
data2=pd.read_csv('/content/drive/MyDrive/Disease dataset/train/_classes.csv')
data3=pd.read_csv('/content/drive/MyDrive/Disease dataset/valid/_classes.csv')
```

```
data1.head()
data2.head()
data3.head()
```

	filename	(BRD)	Bovine	Contagious	Dermatitis	Disease	Ecthym	Respiratory	Unlabeled	h
0	istockphoto-1319281522-612x612_jpg.rf.995226a1...	0	0	0	0	0	0	0	0	
1	IMG_20220829_124626_jpg.rf.a354942b646265839d9...	0	0	0	0	0	0	0	0	
2	images_jpeg.rf.a5131ece4904d8dcaf77e31b7258378...	1	1	0	0	1	0	1	0	
3	IMG-20220830-WA0065_jpg.rf.9b091d6fbdde75a3d8e...	0	0	0	0	0	0	0	0	
4	Ayrshirecattle69_jpg.rf.983d9add00cdf1d9e36b3d...	0	0	0	0	0	0	0	0	

Next steps: Generate code with data3 View recommended plots New interactive sheet

```
dataset=tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/Disease dataset",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 830 files belonging to 3 classes.
```

```
class_names=dataset.class_names
class_names
```

```
['test', 'train', 'valid']
```

```
for image_batch,labels_batch in dataset.take(1):
  print(image_batch.shape)
  print(labels_batch.numpy())
  for i in range(12):
    ax=plt.subplot(3,4,i+1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    plt.title(class_names[labels_batch[i]])
    plt.axis("off")
```

```
(32, 640, 640, 3)
[1 2 2 0 1 1 2 1 1 1 1 1 1 1 2 2 0 1 1 2 1 1 1 1 2 1 1 0 1 2 2 1]
```


```
train_size=0.8
len(dataset)*train_size
```

```
20.8
```

```
train_ds=dataset.take(20)
len(train_ds)
```

```
20
```

```
test_ds=dataset.skip(20)
len(test_ds)
```

```
6
```

```
val_size=0.1
len(dataset)*val_size
```

```
2.6
```

```
val_ds=test_ds.take(2)
len(val_ds)
```

```
2
```

```
def get_dataset_partitions_tf(ds,train_split=0.8,val_split=0.1,test_split=0.1,shuffle=True,shuffle_size=10000):
  assert(train_split+test_split+val_split)==1
  ds_size=len(ds)
  train_size=int(train_split*ds_size)
  val_size=int(val_split*ds_size)
  test_size=int(test_split*ds_size)
  if shuffle:
    ds=ds.shuffle(shuffle_size,seed=12)
  train_ds=ds.take(train_size)
  val_ds=ds.skip(train_size).take(val_size)
  test_ds=ds.skip(train_size+val_size)
  return train_ds,val_ds,test_ds


train_ds,val_ds,test_ds=get_dataset_partitions_tf(dataset)
len(train_ds)
len(val_ds)
len(test_ds)
```

```
4
```

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

for image_batch,labels_batch in train_ds.take(1):
  print(image_batch.shape)
  print(image_batch[0].numpy()/255)
```

```
(32, 640, 640, 3)
[[[0.7372549  0.7647059  0.7882353 ]
  [0.7372549  0.7647059  0.7882353 ]
  [0.7372549  0.7647059  0.7882353 ]
  ...
  [0.3137255  0.31764707 0.19607843]
  [0.3019608  0.30588236 0.18039216]
  [0.2901961  0.29411766 0.16862746]]

 [[0.7372549  0.7647059  0.7882353 ]
  [0.7372549  0.7647059  0.7882353 ]
  [0.7372549  0.7647059  0.7882353 ]
  ...
  [0.31764707 0.32156864 0.2       ]
  [0.29411766 0.29803923 0.1764706 ]
  [0.27058825 0.27450982 0.14901961]]

 [[0.7372549  0.7647059  0.7882353 ]
  [0.7372549  0.7647059  0.7882353 ]
  [0.7372549  0.7647059  0.7882353 ]
  ...
  [0.34117648 0.34117648 0.23137255]
  [0.3137255  0.3137255  0.20392157]
  [0.28235295 0.28627452 0.16470589]]

 ...

 [[0.45882353 0.42745098 0.3764706 ]
  [0.4392157  0.40784314 0.35686275]
  [0.42745098 0.40392157 0.34901962]
  ...
  [0.8392157  0.5411765  0.40784314]
  [0.8352941  0.5411765  0.41568628]
  [0.8352941  0.5411765  0.41568628]]

 [[0.45882353 0.42745098 0.3764706 ]
  [0.44313726 0.4117647  0.36078432]
  [0.43529412 0.4117647  0.35686275]
  ...
  [0.8392157  0.5411765  0.40784314]
  [0.8392157  0.54509807 0.41960785]
  [0.8392157  0.54509807 0.41960785]]

 [[0.45490196 0.42352942 0.37254903]
  [0.44313726 0.4117647  0.36078432]
  [0.44313726 0.41960785 0.3647059 ]
  ...
  [0.84313726 0.54509807 0.4117647 ]
  [0.8392157  0.54509807 0.41960785]
  [0.8392157  0.54509807 0.41960785]]]
```

```
resize_and_rescale=tf.keras.Sequential([
    layers.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    layers.Rescaling(1.0/255)
])


data_augmentation=tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2)
])


input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
n_classes=3

model=models.Sequential([
    layers.Input(shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS)),
    data_augmentation,
    resize_and_rescale,
    layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
```

```
    layers.Dense(n_classes,activation='softmax'),

])
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
sequential_3 (Sequential)	(None, 640, 640, 3)	0
sequential_2 (Sequential)	(None, 640, 640, 3)	0
conv2d_1 (Conv2D)	(None, 638, 638, 32)	896
max_pooling2d (MaxPooling2D)	(None, 319, 319, 32)	0
conv2d_2 (Conv2D)	(None, 317, 317, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 158, 158, 32)	0
conv2d_3 (Conv2D)	(None, 156, 156, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 78, 78, 64)	0
conv2d_4 (Conv2D)	(None, 76, 76, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(None, 38, 38, 64)	0
conv2d_5 (Conv2D)	(None, 36, 36, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_6 (Conv2D)	(None, 16, 16, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_7 (Conv2D)	(None, 6, 6, 64)	36,928
max_pooling2d_6 (MaxPooling2D)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 3)	195

Total params: 213,475 (833.89 KB)

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

```
32
```

```
history=model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.6882 - loss: 0.7888 - val_accuracy: 0.7031 - val_loss: 0.7991
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 372ms/step - accuracy: 0.6750 - loss: 0.8158 - val_accuracy: 0.7031 - val_loss: 0.8062
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 373ms/step - accuracy: 0.6735 - loss: 0.8021 - val_accuracy: 0.7031 - val_loss: 0.8214
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 373ms/step - accuracy: 0.6715 - loss: 0.8242 - val_accuracy: 0.7031 - val_loss: 0.8239
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 380ms/step - accuracy: 0.6292 - loss: 0.8634 - val_accuracy: 0.6875 - val_loss: 0.8239
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 375ms/step - accuracy: 0.6534 - loss: 0.8026 - val_accuracy: 0.6250 - val_loss: 0.8393
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 376ms/step - accuracy: 0.6683 - loss: 0.7757 - val_accuracy: 0.6094 - val_loss: 0.8358
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 371ms/step - accuracy: 0.6760 - loss: 0.7921 - val_accuracy: 0.6094 - val_loss: 0.8322
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.6692 - loss: 0.7788 - val_accuracy: 0.6562 - val_loss: 0.8149
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.6545 - loss: 0.8019 - val_accuracy: 0.7031 - val_loss: 0.8025
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 378ms/step - accuracy: 0.6377 - loss: 0.8360 - val_accuracy: 0.6875 - val_loss: 0.8051
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 371ms/step - accuracy: 0.6988 - loss: 0.7852 - val_accuracy: 0.6094 - val_loss: 0.8425
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 370ms/step - accuracy: 0.6524 - loss: 0.7825 - val_accuracy: 0.6719 - val_loss: 0.7967
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 371ms/step - accuracy: 0.6883 - loss: 0.7640 - val_accuracy: 0.6094 - val_loss: 0.8288
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 372ms/step - accuracy: 0.7071 - loss: 0.7417 - val_accuracy: 0.6562 - val_loss: 0.8371
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 370ms/step - accuracy: 0.7168 - loss: 0.7191 - val_accuracy: 0.6562 - val_loss: 0.8276
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - accuracy: 0.6788 - loss: 0.7559 - val_accuracy: 0.6406 - val_loss: 0.8199
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 374ms/step - accuracy: 0.6343 - loss: 0.8072 - val_accuracy: 0.6875 - val_loss: 0.8035
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.6364 - loss: 0.8026 - val_accuracy: 0.6875 - val_loss: 0.7859
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 373ms/step - accuracy: 0.7441 - loss: 0.7056 - val_accuracy: 0.5781 - val_loss: 0.8604
```

```
train_ds
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 640, 640, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

```
tf.__version__
```

```
'2.17.1'
```

```
scores=model.evaluate(test_ds)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 108ms/step - accuracy: 0.5406 - loss: 1.0267
```

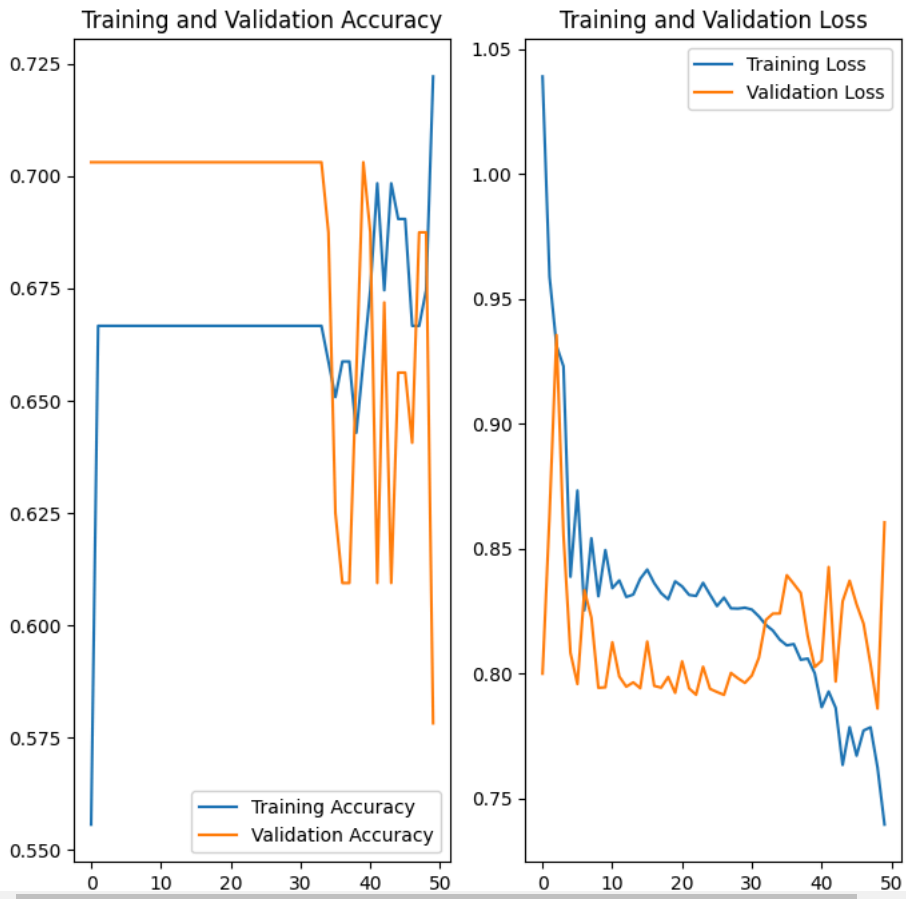
```
scores
```

```
[1.0516738891601562, 0.53125]
```

```
history.history.keys()
```

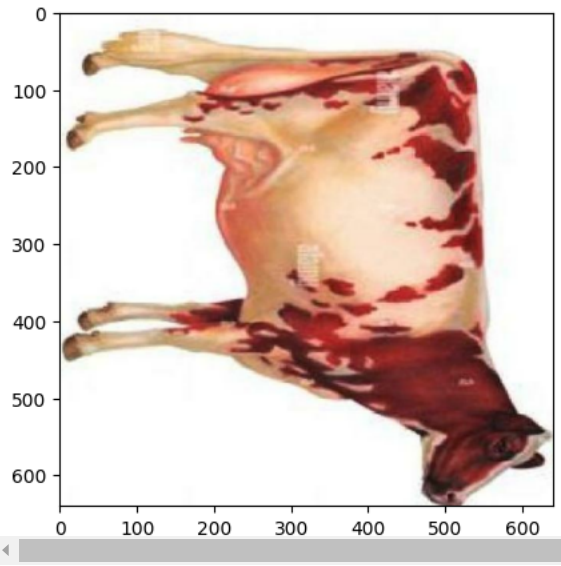
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

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

plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS),acc,label='Training Accuracy')
plt.plot(range(EPOCHS),val_acc,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1,2,2)
plt.plot(range(EPOCHS),loss,label='Training Loss')
plt.plot(range(EPOCHS),val_loss,label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```




```
import numpy as np

for images_batch,labels_batch in test_ds.take(1):
  first_image=images_batch[0].numpy().astype('uint8')
  first_label=labels_batch[0].numpy()
  print("first image to predict")
  plt.imshow(first_image)
  print("actual label:",class_names[first_label])
  batch_prediction=model.predict(images_batch)
  print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
actual label: train
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 335ms/step
predicted label: train
```



```
def predict(model,img):
  img_array=tf.keras.preprocessing.image.img_to_array(images[i].numpy())
  img_array=tf.expand_dims(img_array,0)

  predictions=model.predict(img_array)
  predicted_class=class_names[np.argmax(predictions[0])]
```

```
  confidence=round(100*(np.max(predictions[0])),2)
  return predicted_class,confidence

plt.figure(figsize=(15,15))
for images,labels in test_ds.take(1):
  for i in range(9):
    ax=plt.subplot(3,3,i+1)
    plt.imshow(images[i].numpy().astype("uint8"))
    predicted_class,confidence=predict(model,images[i].numpy())
    actual_class=class_names[labels[i]]
    plt.title(f"Actual:{actual_class},\n Predicted:{predicted_class}.\n Confidence:{confidence}%")
    plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 694ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

```
import os
save_dir="/content/drive/MyDrive/Disease dataset/models/"
model_version=1
os.makedirs(f"{save_dir}{model_version}",exist_ok=True)
model.save(f"{save_dir}{model_version}.keras")
```

```
model= models.Sequential([
    layers.Input(shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS)),
    data_augmentation,
    resize_and_rescale,
    layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(128,128,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(128,128,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,kernel_size=(3,3),activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(n_classes,activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
sequential_3 (Sequential)	(None, 640, 640, 3)	0
sequential_2 (Sequential)	(None, 640, 640, 3)	0
conv2d_14 (Conv2D)	(None, 638, 638, 32)	896
max_pooling2d_13 (MaxPooling2D)	(None, 319, 319, 32)	0
conv2d_15 (Conv2D)	(None, 317, 317, 32)	9,248
max_pooling2d_14 (MaxPooling2D)	(None, 158, 158, 32)	0
conv2d_16 (Conv2D)	(None, 156, 156, 64)	18,496
max_pooling2d_15 (MaxPooling2D)	(None, 78, 78, 64)	0
conv2d_17 (Conv2D)	(None, 76, 76, 64)	36,928
max_pooling2d_16 (MaxPooling2D)	(None, 38, 38, 64)	0
conv2d_18 (Conv2D)	(None, 36, 36, 64)	36,928
max_pooling2d_17 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_19 (Conv2D)	(None, 16, 16, 64)	36,928
max_pooling2d_18 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_2 (Flatten)	(None, 4096)	0
dense_4 (Dense)	(None, 64)	262,208
dense_5 (Dense)	(None, 3)	195

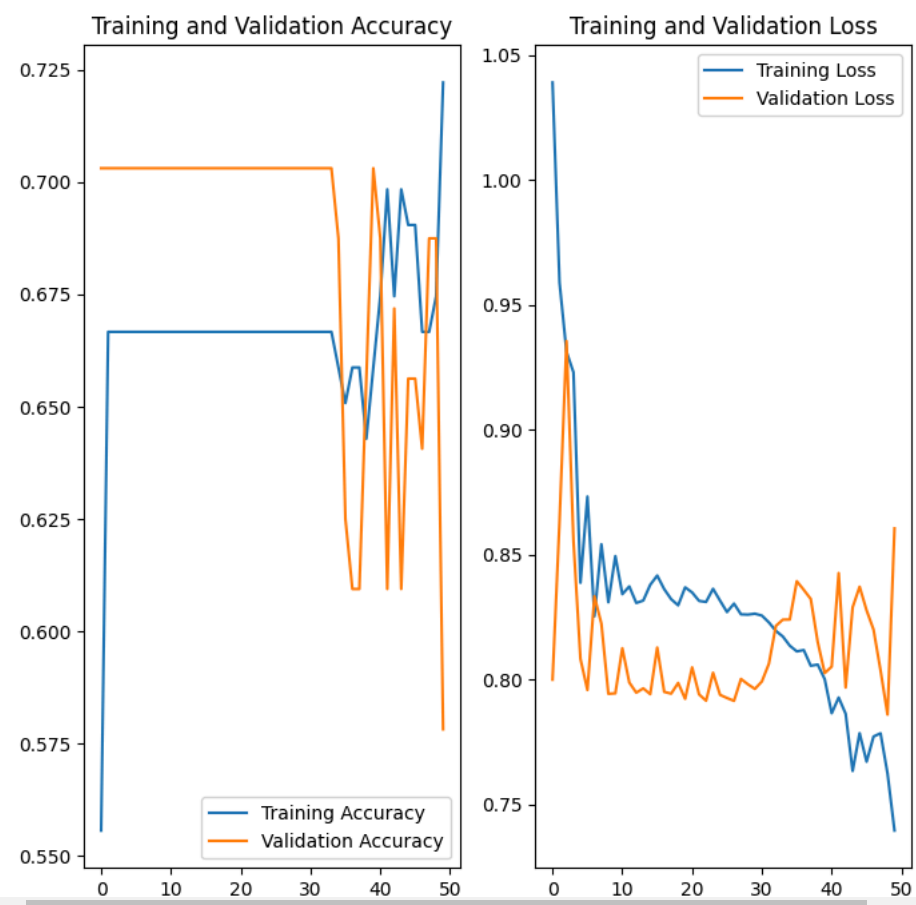
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
history=model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
```

```
    verbose=1,
    epochs=EPOCHS,
)
```

```
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 377ms/step - accuracy: 0.6882 - loss: 0.7519 - val_accuracy: 0.6719 - val_loss: 0.8136
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 376ms/step - accuracy: 0.6830 - loss: 0.7858 - val_accuracy: 0.6250 - val_loss: 0.8155
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.6822 - loss: 0.7477 - val_accuracy: 0.5938 - val_loss: 0.8227
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 375ms/step - accuracy: 0.6851 - loss: 0.7502 - val_accuracy: 0.5625 - val_loss: 0.8355
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 413ms/step - accuracy: 0.6548 - loss: 0.8003 - val_accuracy: 0.5938 - val_loss: 0.8060
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 378ms/step - accuracy: 0.6593 - loss: 0.7833 - val_accuracy: 0.6094 - val_loss: 0.7856
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 410ms/step - accuracy: 0.7022 - loss: 0.7473 - val_accuracy: 0.5625 - val_loss: 0.8413
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 378ms/step - accuracy: 0.6681 - loss: 0.7632 - val_accuracy: 0.5625 - val_loss: 0.8552
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 373ms/step - accuracy: 0.6989 - loss: 0.7630 - val_accuracy: 0.6250 - val_loss: 0.7860
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 372ms/step - accuracy: 0.6999 - loss: 0.7529 - val_accuracy: 0.5781 - val_loss: 0.8355
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 373ms/step - accuracy: 0.6388 - loss: 0.8057 - val_accuracy: 0.6094 - val_loss: 0.8415
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 373ms/step - accuracy: 0.7074 - loss: 0.7550 - val_accuracy: 0.6250 - val_loss: 0.8062
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 370ms/step - accuracy: 0.7198 - loss: 0.6983 - val_accuracy: 0.5625 - val_loss: 0.8387
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 375ms/step - accuracy: 0.6904 - loss: 0.7720 - val_accuracy: 0.6250 - val_loss: 0.7848
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 388ms/step - accuracy: 0.6798 - loss: 0.7596 - val_accuracy: 0.6406 - val_loss: 0.8039
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.7167 - loss: 0.7465 - val_accuracy: 0.6562 - val_loss: 0.8268
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - accuracy: 0.6903 - loss: 0.7397 - val_accuracy: 0.6250 - val_loss: 0.8108
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.6956 - loss: 0.7549 - val_accuracy: 0.6406 - val_loss: 0.7765
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 371ms/step - accuracy: 0.6979 - loss: 0.7828 - val_accuracy: 0.6875 - val_loss: 0.7727
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 371ms/step - accuracy: 0.6851 - loss: 0.7784 - val_accuracy: 0.6562 - val_loss: 0.7977
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 373ms/step - accuracy: 0.7095 - loss: 0.7347 - val_accuracy: 0.6719 - val_loss: 0.8139
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 377ms/step - accuracy: 0.7180 - loss: 0.7223 - val_accuracy: 0.6719 - val_loss: 0.8228
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 373ms/step - accuracy: 0.6697 - loss: 0.7730 - val_accuracy: 0.6250 - val_loss: 0.8603
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 373ms/step - accuracy: 0.7031 - loss: 0.7712 - val_accuracy: 0.6562 - val_loss: 0.8032
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 373ms/step - accuracy: 0.7230 - loss: 0.7456 - val_accuracy: 0.6719 - val_loss: 0.8039
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.6915 - loss: 0.7531 - val_accuracy: 0.6562 - val_loss: 0.7870
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 371ms/step - accuracy: 0.7000 - loss: 0.6877 - val_accuracy: 0.6406 - val_loss: 0.8369
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 371ms/step - accuracy: 0.7295 - loss: 0.7351 - val_accuracy: 0.6562 - val_loss: 0.7860
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 436ms/step - accuracy: 0.7269 - loss: 0.7457 - val_accuracy: 0.6719 - val_loss: 0.7849
```

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

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




```
plt.figure(figsize=(15,15))
for images,labels in test_ds.take(1):
  for i in range(9):
    ax=plt.subplot(3,3,i+1)
    plt.imshow(images[i].numpy().astype("uint8"))
    predicted_class,confidence=predict(model,images[i].numpy())
    actual_class=class_names[labels[i]]
    plt.title(f"Actual:{actual_class},\n Predicted:{predicted_class}.\n Confidence:{confidence}%")
    plt.axis("off")
```

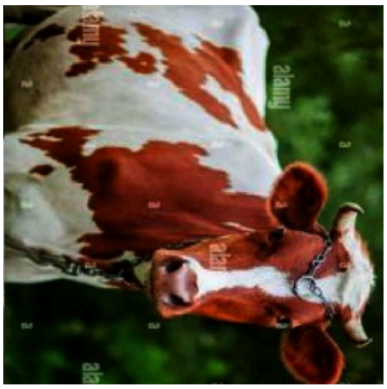
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 217ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
```

Actual:train,
Predicted:train.
Confidence:56.95%

Actual:test,
Predicted:train.
Confidence:65.87%

Actual:train,
Predicted:train.
Confidence:69.36%

Actual:test,
Predicted:train.
Confidence:69.46%



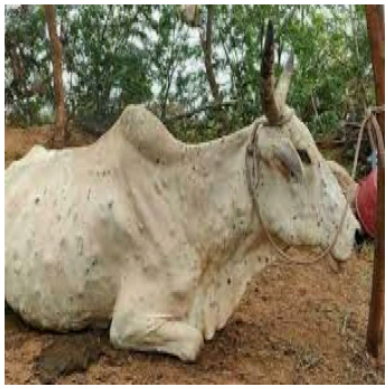
Actual:train,
Predicted:train.
Confidence:47.4%

Actual:valid,
Predicted:train.
Confidence:79.19%

Actual:valid,
Predicted:train.
Confidence:63.89%

Actual:train,
Predicted:train.
Confidence:81.83%

Actual:valid,
Predicted:train.
Confidence:68.52%



```
import os
save_dir="/content/drive/MyDrive/Disease dataset/models/"
model_version=2
os.makedirs(f"{save_dir}{model_version}",exist_ok=True)
model.save(f"{save_dir}{model_version}.keras")
```

```
model=models.Sequential([
    layers.Input(shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS)),
    data_augmentation,
    resize_and_rescale,
    layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(640,640,3))
```

```
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(n_classes,activation='softmax')
])
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_ onv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
sequential_3 (Sequential)	(None, 640, 640, 3)	0
sequential_2 (Sequential)	(None, 640, 640, 3)	0
conv2d_20 (Conv2D)	(None, 638, 638, 32)	896
max_pooling2d_19 (MaxPooling2D)	(None, 319, 319, 32)	0
flatten_3 (Flatten)	(None, 3256352)	0
dense_6 (Dense)	(None, 64)	208,406,592
dense_7 (Dense)	(None, 3)	195

Total params: 208,407,683 (795.01 MB)
Trainable params: 208,407,683 (795.01 MB)

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


history=model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)
```

```
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 341ms/step - accuracy: 0.6967 - loss: 0.7488 - val_accuracy: 0.6094 - val_loss: 0.8848
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 341ms/step - accuracy: 0.6978 - loss: 0.7338 - val_accuracy: 0.6719 - val_loss: 0.7929
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 339ms/step - accuracy: 0.6884 - loss: 0.7037 - val_accuracy: 0.6406 - val_loss: 0.8243
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 341ms/step - accuracy: 0.7043 - loss: 0.7046 - val_accuracy: 0.6406 - val_loss: 0.8670
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 337ms/step - accuracy: 0.6875 - loss: 0.6991 - val_accuracy: 0.6094 - val_loss: 0.9327
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 337ms/step - accuracy: 0.6677 - loss: 0.7140 - val_accuracy: 0.6250 - val_loss: 0.8643
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 342ms/step - accuracy: 0.7020 - loss: 0.6891 - val_accuracy: 0.6406 - val_loss: 0.8224
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 344ms/step - accuracy: 0.6710 - loss: 0.7760 - val_accuracy: 0.6250 - val_loss: 0.8907
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 341ms/step - accuracy: 0.7080 - loss: 0.7264 - val_accuracy: 0.6406 - val_loss: 0.8757
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 393ms/step - accuracy: 0.7127 - loss: 0.7101 - val_accuracy: 0.6094 - val_loss: 0.9244
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 339ms/step - accuracy: 0.7013 - loss: 0.7313 - val_accuracy: 0.6250 - val_loss: 0.9024
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 339ms/step - accuracy: 0.7033 - loss: 0.7308 - val_accuracy: 0.6406 - val_loss: 0.8483
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 344ms/step - accuracy: 0.6862 - loss: 0.7602 - val_accuracy: 0.6094 - val_loss: 0.9511
Epoch 35/50
```