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

```
Mounted at /content/drive
```

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.python .keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix , classification_report , auc , accuracy_score
import tensorflow as tf
from tensorflow import keras
from PIL import Image
import cv2
import matplotlib.image as mpimg
import tensorflow as tf
from tensorflow import keras
from keras.applications.vgg19 import VGG19
from keras.layers import Dense,Dropout,Conv2D, Flatten,MaxPooling2D
from keras.models import Sequential
from tensorflow.keras.preprocessing import image_dataset_from_directory
import warnings
warnings.filterwarnings('ignore')
```

```
path="/content/drive/MyDrive/dataset (1)/tomato&grape/training"
train = image_dataset_from_directory(path, batch_size=32,
                                     image_size=(256,256),shuffle=True,labels='inferred', label_mode='int')
```

```
Found 2800 files belonging to 8 classes.
```

```
path="/content/drive/MyDrive/dataset (1)/tomato&grape/testing"
test = image_dataset_from_directory(path, batch_size=32,
                                     image_size=(256,256),shuffle=True)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-2b97a059585e> in <cell line: 2>()
      1 path="/content/drive/MyDrive/dataset (1)/tomato&grape/testing"
----> 2 test = image_dataset_from_directory(path, batch_size=32,
      3                                    image_size=(256,256),shuffle=True)

 5 frames
/usr/lib/python3.10/threading.py in wait(self, timeout)
    318         try:    # restore state no matter what (e.g., KeyboardInterrupt)
    319             if timeout is None:
--> 320                 waiter.acquire()
    321                 gotit = True
    322             else:

KeyboardInterrupt:
```

```
class_name=train.class_names
print(class_name)
```

```
['grape_black measles', 'grape_black rot', 'grape_healthy', 'grape_isoproisis spot', 'tomato_bacterial', 'tomato_healthy', 'tomato_late blight', 'tomato_leaf curl']
```

```
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(256, 256, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
```

```
for layer in base_model.layers:
    layer.trainable = False
```

```
model = Sequential()

model.add(base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(8, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 8, 8, 512)         20024384

 flatten (Flatten)           (None, 32768)             0
```

```
 dense (Dense)               (None, 512)               16777728

 dense_1 (Dense)             (None, 8)                 4104

=================================================================
Total params: 36806216 (140.40 MB)
Trainable params: 16781832 (64.02 MB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

Double-click (or enter) to edit

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

Double-click (or enter) to edit

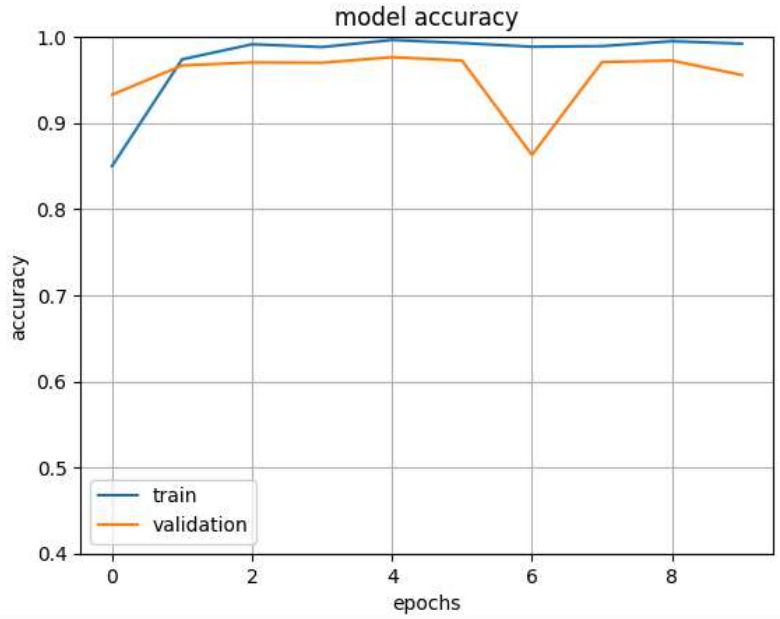
Double-click (or enter) to edit

```
from keras.optimizers import Adam

# Compile the model
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(train, epochs=10, validation_data=test)
```

```
Epoch 1/10
88/88 [==============================] - 1122s 13s/step - loss: 12.2391 - accuracy: 0.8500 - val_loss: 2.0591 - val_accuracy: 0.9329
Epoch 2/10
88/88 [==============================] - 34s 369ms/step - loss: 0.5451 - accuracy: 0.9739 - val_loss: 0.8695 - val_accuracy: 0.9668
Epoch 3/10
88/88 [==============================] - 34s 372ms/step - loss: 0.1947 - accuracy: 0.9914 - val_loss: 0.8038 - val_accuracy: 0.9704
Epoch 4/10
88/88 [==============================] - 34s 378ms/step - loss: 0.2332 - accuracy: 0.9882 - val_loss: 0.8114 - val_accuracy: 0.9700
Epoch 5/10
88/88 [==============================] - 38s 424ms/step - loss: 0.0266 - accuracy: 0.9964 - val_loss: 0.7662 - val_accuracy: 0.9764
Epoch 6/10
88/88 [==============================] - 39s 426ms/step - loss: 0.1502 - accuracy: 0.9929 - val_loss: 0.7582 - val_accuracy: 0.9725
Epoch 7/10
88/88 [==============================] - 38s 425ms/step - loss: 0.2039 - accuracy: 0.9886 - val_loss: 6.6685 - val_accuracy: 0.8629
Epoch 8/10
88/88 [==============================] - 39s 428ms/step - loss: 0.3325 - accuracy: 0.9893 - val_loss: 1.0396 - val_accuracy: 0.9707
Epoch 9/10
88/88 [==============================] - 38s 427ms/step - loss: 0.1042 - accuracy: 0.9950 - val_loss: 1.1343 - val_accuracy: 0.9725
Epoch 10/10
88/88 [==============================] - 39s 428ms/step - loss: 0.1476 - accuracy: 0.9921 - val_loss: 1.4197 - val_accuracy: 0.9557
```

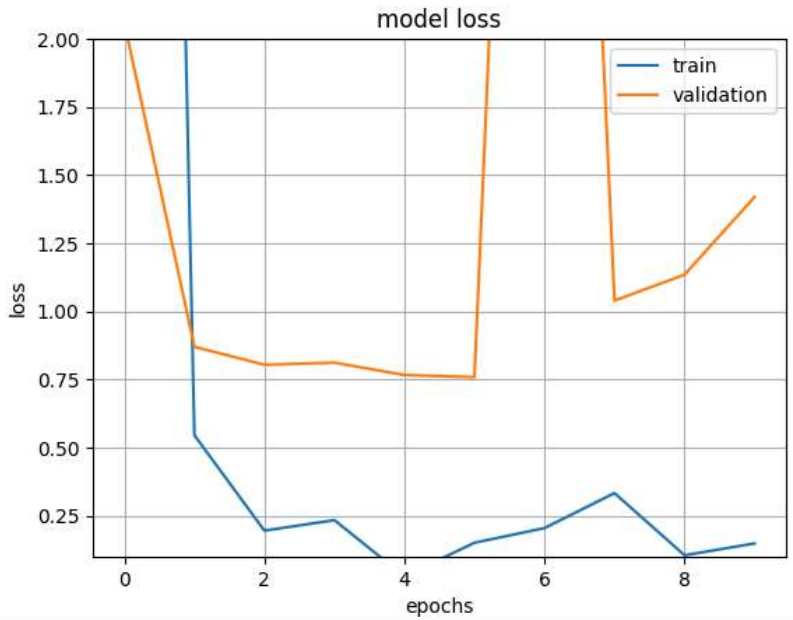
```
#from keras.utils import to_categorical

# Assuming y_train is your target data
```

```
# num_classes should be the total number of classes in your dataset
#num_classes = 8  # Adjust this according to your dataset

# Convert y_train to one-hot encoded format
#y_train_encoded = to_categorical(y_train, num_classes=num_classes)

# Now you can use y_train_encoded for training your model

#epochs=10
##history= model.fit(
  #train,
  #validation_data=test,
  #epochs=epochs
#)


fig1=plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train','validation'])
plt.show()
```




```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

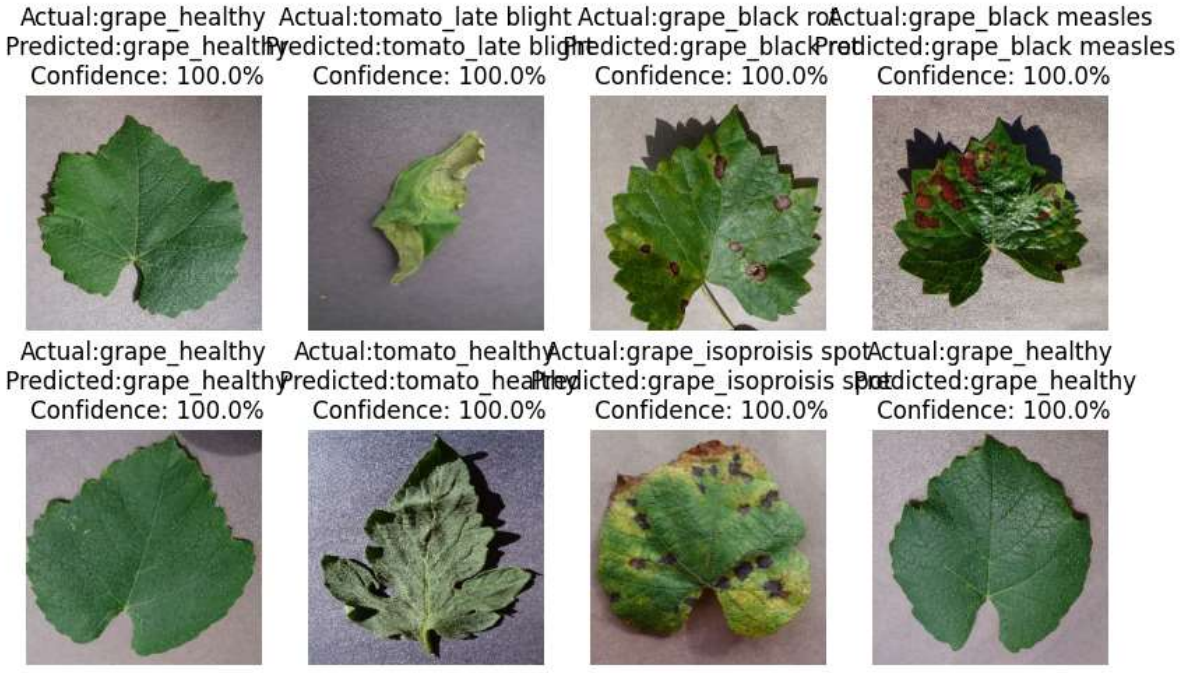
```
plt.plot(history.history['val_loss'])
plt.axis(ymin=0.1,ymax=2)
plt.grid()
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(['train','validation'])
plt.show()
```




```
def Prediction(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)

  predict_class = class_name[np.argmax(predictions[0])]
  confidence= round(100*(np.max(predictions[0])),2)

  return predict_class , confidence


plt.figure(figsize=(10,15))
for images, labels in test.take(1):
  for i in range(8):
    ax= plt.subplot(5,4,i+1)
    plt.imshow(images[i].numpy().astype("uint8"))

    predicted_class, confidence = Prediction(model,images[i].numpy())
    actual_class =class_name[labels[i]]
```

```
    plt.title(f"Actual:{actual_class}\n Predicted:{predicted_class}\n Confidence: {confidence}%")
    plt.axis('off')
```

```
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 18ms/step
 1/1 [==============================] - 0s 17ms/step
 1/1 [==============================] - 0s 17ms/step
 1/1 [==============================] - 0s 23ms/step
 1/1 [==============================] - 0s 20ms/step
 1/1 [==============================] - 0s 27ms/step
 1/1 [==============================] - 0s 17ms/step
```



Double-click (or enter) to edit

```
class_labels = test.class_names

# Assuming the model.predict() returns the predictions
y_pred = model.predict(test)
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
 88/88 [==============================] - 17s 180ms/step
```

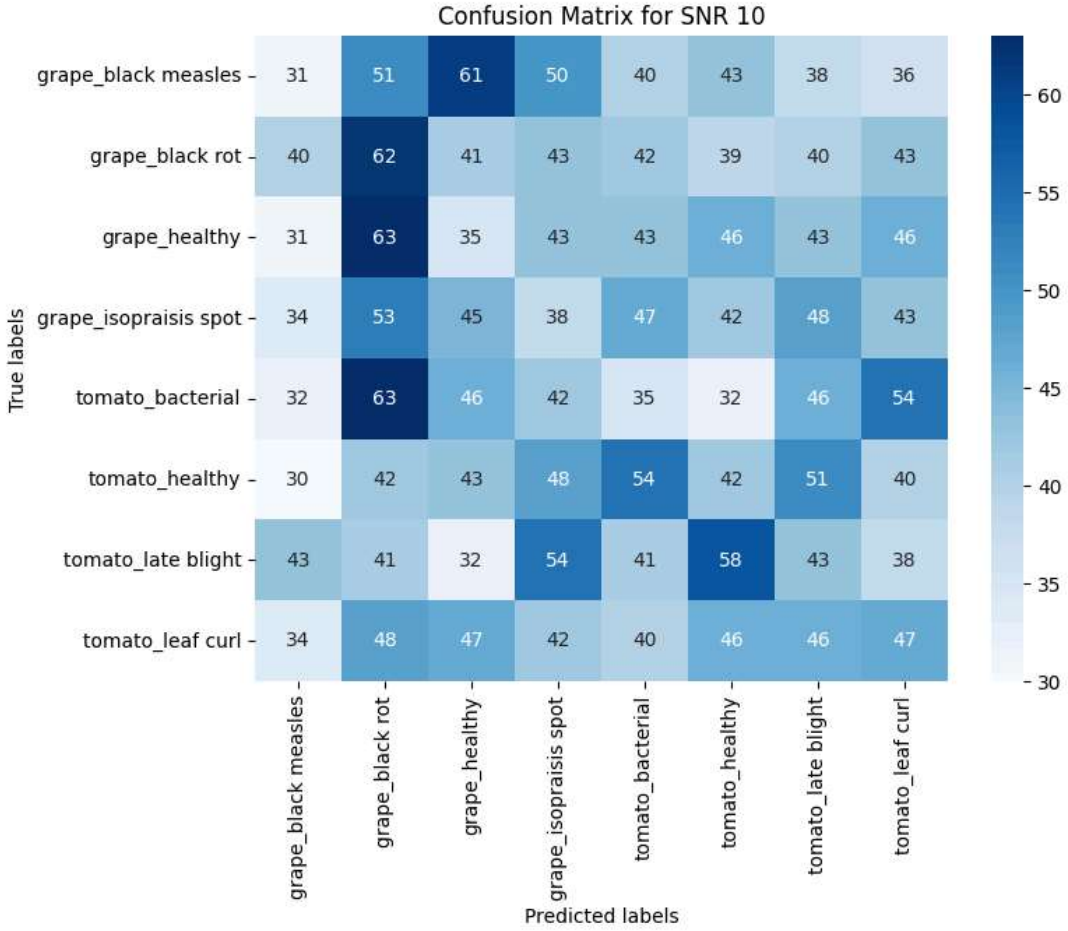
```
true_classes = []
predicted_classes = []

# Iterate through the dataset to extract true labels
for images, labels in test:
    true_classes.extend(labels.numpy())  # Convert TensorFlow tensor to numpy array
```

```
# Assuming the model.predict() returns the predictions
y_pred = model.predict(test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Extend predicted_classes list with predicted labels
predicted_classes.extend(y_pred_classes)

# Compute the confusion matrix
conf_matrix = confusion_matrix(true_classes, predicted_classes)

# Print the confusion matrix with class labels
print("Confusion matrix:")
print(conf_matrix)
print("")

# Generate and print the classification report
class_report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print("Classification report:")
print(class_report)
```

```
88/88 [==============================] - 16s 180ms/step
Confusion matrix:
[[31 51 61 50 40 43 38 36]
 [40 62 41 43 42 39 40 43]
 [31 63 35 43 43 46 43 46]
 [34 53 45 38 47 42 48 43]
 [32 63 46 42 35 32 46 54]
 [30 42 43 48 54 42 51 40]
 [43 41 32 54 41 58 43 38]
 [34 48 47 42 40 46 46 47]]

Classification report:
                       precision    recall  f1-score   support

  grape_black measles       0.11      0.09      0.10       350
      grape_black rot       0.15      0.18      0.16       350
        grape_healthy       0.10      0.10      0.10       350
grape_isopraisis spot       0.11      0.11      0.11       350
     tomato_bacterial       0.10      0.10      0.10       350
       tomato_healthy       0.12      0.12      0.12       350
  tomato_late blight       0.12      0.12      0.12       350
    tomato_leaf curl       0.14      0.13      0.13       350

            accuracy                           0.12      2800
           macro avg       0.12      0.12      0.12      2800
        weighted avg       0.12      0.12      0.12      2800
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)

# Add labels and title
```

```
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion Matrix for SNR 10")

# Display the plot
plt.show()
```




```
def calculate_precision_recall_f1(TP, FP, TN, FN):
    precision = (TP / (TP + FP)) * 100
    recall = (TP / (TP + FN)) * 100
    f1_score = (2 * precision * recall) / (precision + recall)
    return precision, recall, f1_score

# Example usage:
TP = 80
FP = 10
TN = 100
FN = 20
```

```
precision, recall, f1_score = calculate_precision_recall_f1(TP, FP, TN, FN)
print("Precision: {:.2f}%".format(precision))
print("Recall: {:.2f}%".format(recall))
print("F1 Score: {:.2f}%".format(f1_score))
```