```
mypath = '/content/drive/My Drive/STARFISH'
```

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

```
import os
```

```
mypath = '/content/drive/My Drive/STARFISH'
```

```
file_name = []
tag = []
full_path = []
for path, subdirs, files in os.walk(mypath):
    for name in files:
        full_path.append(os.path.join(path, name))
        tag.append(path.split('/')[-1])
        file_name.append(name)
```

```
import pandas as pd
```

```
# input variables already collected in the looping above into a dataframe
df = pd.DataFrame({"path":full_path,'file_name':file_name,"tag":tag})
df.groupby(['tag']).size()
```

```
#load library train and test split
from sklearn.model_selection import train_test_split
```

```
#variables used in this data split
X= df['path']
y= df['tag']
```

```
# split the initial dataset into train and test data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=300)
```

```
# divide the test data into 2 more so that it becomes a test data and validation data.
X_test, X_val, y_test, y_val = train_test_split(
    X_test, y_test, test_size=0.5, random_state=100)
```

```
# unifying into each dataframe

df_tr = pd.DataFrame({'path':X_train
              ,'tag':y_train
             ,'set':'train'})

df_te = pd.DataFrame({'path':X_test
              ,'tag':y_test
```

```
            ,'set':'test'})

df_val = pd.DataFrame({'path':X_val
             ,'tag':y_val
            ,'set':'validation'})

print('train size', len(df_tr))
print('val size', len(df_te))
print('test size', len(df_val))


# see the proportions on each set whether it is ok or there is still something to change
df_all = df_tr.append([df_te,df_val]).reset_index(drop=1)\

print('========================================================= \n')
print(df_all.groupby(['set','tag']).size(),'\n')

print('========================================================= \n')

#check data sample
df_all.sample(4)


df.head()


import shutil
from tqdm.notebook import tqdm as tq


datasource_path = "/content/drive/My Drive/STARFISH/"
dataset_path = "dataset/"


for index, row in tq(df_all.iterrows()):

    #detect filepath
    file_path = row['path']
    if os.path.exists(file_path) == False:
            file_path = os.path.join(datasource_path,row['tag'],row['image'].split('.')[0]

    #make folder destination dirs
    if os.path.exists(os.path.join(dataset_path,row['set'],row['tag'])) == False:
        os.makedirs(os.path.join(dataset_path,row['set'],row['tag']))

    #define file dest
    destination_file_name = file_path.split('/')[-1]
    file_dest = os.path.join(dataset_path,row['set'],row['tag'],destination_file_name)

    #copy file from source to dest
    if os.path.exists(file_dest) == False:
        shutil.copy2(file_path,file_dest)


import tensorflow as tf
```

```
# Define Input Parameters
dim = (32, 32)
channel = (3, )
input_shape = dim + channel

#batch size
batch_size = 16

#Epoch
epoch = 10


from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_datagen = ImageDataGenerator(rescale=1./255,
                                   zoom_range=0.2,
                                   horizontal_flip=True)

val_datagen = ImageDataGenerator(rescale=1./255)

test_datagen = ImageDataGenerator(rescale=1./255)


train_data = train_datagen.flow_from_directory('dataset/train/',
                                               target_size=dim,
                                               batch_size=batch_size,
                                               class_mode='categorical',
                                               shuffle=True)

val_data = val_datagen.flow_from_directory('dataset/validation/',
                                           target_size=dim,
                                           batch_size=batch_size,
                                           class_mode='categorical',
                                           shuffle=True)

test_data = test_datagen.flow_from_directory('dataset/test/',
                                             target_size=dim,
                                             batch_size=batch_size,
                                             class_mode='categorical',
                                             shuffle=True)

num_class = test_data.num_classes
labels = train_data.class_indices.keys()


print(labels)


import numpy as np # linear algebra
from sklearn.metrics import accuracy_score, f1_score, precision_score, confusion_matrix
from sklearn.model_selection import StratifiedKFold
from PIL import Image
import random
#Dependencies
import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.preprocessing.image import ImageDataGenerator
#CNN
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
import warnings
import os
import shutil
from PIL import ImageFile
warnings.simplefilter('error', Image.DecompressionBombWarning)
ImageFile.LOAD_TRUNCATED_IMAGES = True
from PIL import Image
Image.MAX_IMAGE_PIXELS = 1000000000

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)



datasetFolderName='dataset/'
MODEL_FILENAME="FINALMODELCNN/starfish_project/model.h5"
sourceFiles=[]
classLabels=['Culcita_Novaeguineae', 'Linckia_Laevigata', 'Luidia_Foliolata', 'Protoreaste
def transferBetweenFolders(source, dest, splitRate):
    global sourceFiles
    sourceFiles=os.listdir(source)
    if(len(sourceFiles)!=0):
        transferFileNumbers=int(len(sourceFiles)*splitRate)
        transferIndex=random.sample(range(0, len(sourceFiles)), transferFileNumbers)
        for eachIndex in transferIndex:
            shutil.move(source+str(sourceFiles[eachIndex]), dest+str(sourceFiles[eachIndex
    else:
        print("No file moved. Source empty!")

def transferAllClassBetweenFolders(source, dest, splitRate):
    for label in classLabels:
        transferBetweenFolders(datasetFolderName+'/'+source+'/'+label+'/',
                               datasetFolderName+'/'+dest+'/'+label+'/',
                               splitRate)



# First, check if test folder is empty or not, if not transfer all existing files to train
transferAllClassBetweenFolders('test', 'train', 1.0)
# Now, split some part of train data into the test folders.
transferAllClassBetweenFolders('train', 'test', 0.20)
```

```
X=[]
Y=[]

def prepareNameWithLabels(folderName):
    sourceFiles=os.listdir(datasetFolderName+'/train/'+folderName)
    for val in sourceFiles:
        X.append(val)
        if(folderName==classLabels[0]):
            Y.append(0)
        elif(folderName==classLabels[1]):
            Y.append(1)
        elif(folderName==classLabels[2]):
            Y.append(2)
        else:
            Y.append(3)


# Organize file names and class labels in X and Y variables
prepareNameWithLabels(classLabels[0])
prepareNameWithLabels(classLabels[1])
prepareNameWithLabels(classLabels[2])
prepareNameWithLabels(classLabels[3])


X=np.asarray(X)
Y=np.asarray(Y)

batch_size = 16
epoch=200
activationFunction='relu'
def getModel():
    model = Sequential()
    model.add(Conv2D(32, (5, 5), padding='same', strides=(1,1), activation=activationFunct
    model.add(Conv2D(32, (5, 5), activation=activationFunction))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(Conv2D(64, (5, 5), padding='same', strides=(1,1), activation=activationFunct
    model.add(Conv2D(64, (5, 5), activation=activationFunction))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(256, activation=activationFunction))
    model.add(Dropout(0.5))
    model.add(Dense(len(classLabels), activation='softmax'))

    opt = keras.optimizers.Adam(learning_rate = 0.0001)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model


def my_metrics(y_true, y_pred):
    accuracy=accuracy_score(y_true, y_pred)
```

```
    precision=precision_score(y_true, y_pred,average='weighted')
    f1Score=f1_score(y_true, y_pred, average='weighted')
    print("Accuracy  : {}".format(accuracy))
    print("Precision : {}".format(precision))
    print("f1Score : {}".format(f1Score))
    cm=confusion_matrix(y_true, y_pred)
    print(cm)
    return accuracy, precision, f1Score


train_path=datasetFolderName+'/train/'
validation_path=datasetFolderName+'/validation/'
test_path=datasetFolderName+'/test/'
model=getModel()


# ===============Stratified 5-Fold======================
skf = StratifiedKFold(n_splits=5, shuffle=True)
skf.get_n_splits(X, Y)
foldNum=0
for train_index, val_index in skf.split(X, Y):
    #First cut all images from validation to train (if any exists)
    transferAllClassBetweenFolders('validation', 'train', 1.0)
    foldNum+=1
    print("Results for fold",foldNum)
    X_train, X_val = X[train_index], X[val_index]
    Y_train, Y_val = Y[train_index], Y[val_index]
    # Move validation images of this fold from train folder to the validation folder
    for eachIndex in range(len(X_val)):
        classLabel=''
        if(Y_val[eachIndex]==0):
            classLabel=classLabels[0]
        elif(Y_val[eachIndex]==1):
            classLabel=classLabels[1]
        elif(Y_val[eachIndex]==2):
            classLabel=classLabels[2]
        else:
            classLabel=classLabels[3]
        #Then, copy the validation images to the validation folder
        shutil.move(datasetFolderName+'/train/'+classLabel+'/'+X_val[eachIndex],
                    datasetFolderName+'/validation/'+classLabel+'/'+X_val[eachIndex])

    #Augmentasi Citra
    train_datagen = ImageDataGenerator(
                rescale=1./255,
                rotation_range=40,
                shear_range=0.20,
                height_shift_range=0.25,
                width_shift_range=0.25,
                    zoom_range=0.20,
                horizontal_flip=True,
                fill_mode="nearest"
                )
    validation_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)
```

```
    #Start ImageClassification Model
    train_generator = train_datagen.flow_from_directory(
        train_path,
        target_size=(32,32),
        batch_size=batch_size,
        class_mode='categorical',
        subset='training')

    validation_generator = validation_datagen.flow_from_directory(
            validation_path,
            target_size=(32,32),
            batch_size=batch_size,
            class_mode=None,  # only data, no labels
            shuffle=False)

    # fit model
    history=model.fit_generator(train_generator,
                        epochs=epoch)

    predictions = model.predict_generator(validation_generator, verbose=1)
    yPredictions = np.argmax(predictions, axis=1)
    true_classes = validation_generator.classes
    # evaluate validation performance
    print("***Performance on Validation data***")
    valAcc, valPrec, valFScore = my_metrics(true_classes, yPredictions)


# =============TESTING=============
print("==============TEST RESULTS============")
test_generator = test_datagen.flow_from_directory(
        test_path,
        target_size=(32, 32),
        batch_size=batch_size,
        class_mode=None,
        shuffle=False)
predictions = model.predict(test_generator, verbose=1)
yPredictions = np.argmax(predictions, axis=1)
true_classes = test_generator.classes

testAcc,testPrec, testFScore = my_metrics(true_classes, yPredictions)
model.save(MODEL_FILENAME)


print('confusion matrix')
print(confusion_matrix(true_classes, yPredictions))


import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report

target_names = ['Culcita_Novaeguineae', 'Linckia_Laevigata', 'Luidia_Foliolata', 'Protorea
print('Classification Report')
print(classification_report(true_classes, yPredictions, target_names=target_names))
```

```
f, ax = plt.subplots(figsize=(8,5))
sns.heatmap(confusion_matrix(true_classes, yPredictions), annot=True, fmt=".0f", ax=ax)
plt.xlabel("Predict")
plt.ylabel("True")
plt.show()
```