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

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

```
!pip install efficientnet
```

```
Collecting efficientnet
  Downloading efficientnet-1.1.1-py3-none-any.whl (18 kB)
Collecting keras-applications<=1.0.8,>=1.0.7 (from efficientnet)
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 50.7/50.7 kB 1.1 MB/s eta 0:00:00
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-packages (from efficientnet) (0.19.3)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->effi
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->efficientnet
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (1.11.4)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (3.3)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image-
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (202
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (1.6.0
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (24.0)
Installing collected packages: keras-applications, efficientnet
Successfully installed efficientnet-1.1.1 keras-applications-1.0.8
```

```
#!unzip /content/drive/MyDrive/handcrafted_image_classification/dataset_full.zip -d /content/drive/MyDrive/handcrafted_image_classificati
```

```
import os
import io
import cv2
import pickle
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm
from PIL import Image
from os import listdir
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from tensorflow.keras import Model
from efficientnet.keras import EfficientNetB2
from tensorflow.keras.utils import img_to_array
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization, AveragePooling2D, GlobalAveragePooling2D

import warnings
warnings.filterwarnings('ignore')
```

```
image_size=128
default_image_size = tuple((128, 128))
labels = os.listdir('/content/drive/MyDrive/handcrafted_image_classification/dataset_full')
directory_root = '/content/drive/MyDrive/handcrafted_image_classification/dataset_full'
classes = len(labels)
```

```
def image_array(dir):
    try:
        img = cv2.imread(dir)
        if img is not None :
            img = cv2.resize(img, default_image_size)
            return img_to_array(img)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

```
# image loading
image_list, label_list = [], []

try:
    print("[INFO] Loading images ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)

    for disease_folder in root_dir :
        plant_disease_folder_list = listdir(f"/content/drive/MyDrive/handcrafted_image_classification/dataset_full")

    for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)

    for  plant_disease_folder in plant_disease_folder_list:
            print(f"[INFO] Processing { plant_disease_folder} ...")
            plant_disease_image_list = listdir(f"{directory_root}/{ plant_disease_folder}/")

            for single_plant_disease_image in plant_disease_image_list :
                if single_plant_disease_image == ".DS_Store" :
                    plant_disease_image_list.remove(single_plant_disease_image)

            for image in plant_disease_image_list[:500]:   #loading 500 image from each class
                image_directory = f"{directory_root}/{ plant_disease_folder}/{image}"
                if image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True or image_directory.endswith(".pn
                    image_list.append(image_array(image_directory))
                    label_list.append( plant_disease_folder)
    print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")
```

```
[INFO] Loading images ...
[INFO] Processing Forest ...
[INFO] Processing Glacier ...
[INFO] Processing Mountains ...
[INFO] Processing Sea ...
[INFO] Processing Streets ...
[INFO] Processing Building ...
[INFO] Image loading completed
```

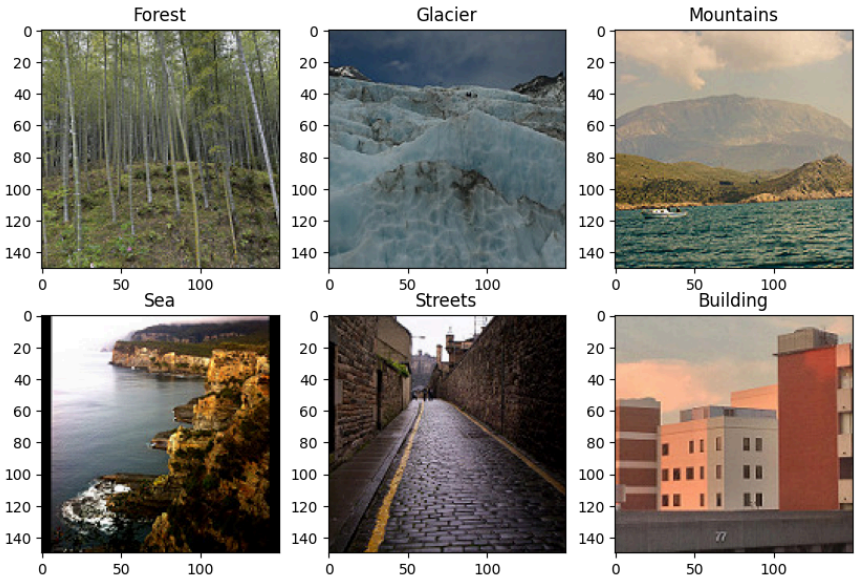
```
X = image_list
Y = label_list
```

```
X = np.array(X)
```

```
X, Y = shuffle(X, Y)
print(X.shape)
```

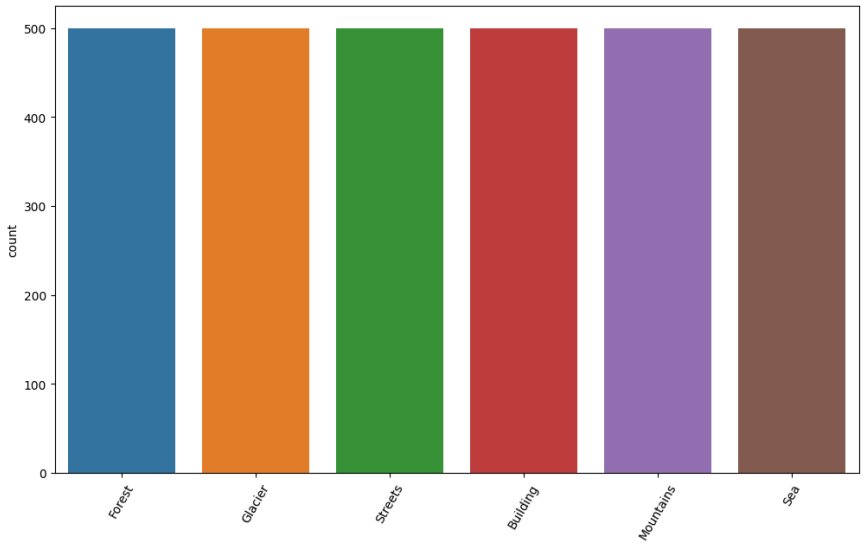
```
(3000, 128, 128, 3)
```

```
#Data Visualization
import matplotlib.image as mpimg
plt.figure(figsize = (10, 10))
image_count = 1
BASE_URL = '/content/drive/MyDrive/handcrafted_image_classification/dataset_full/'
for directory in os.listdir(BASE_URL):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(3, 3, image_count)
                image_count += 1
                image = mpimg.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(directory)
```




```
#count plot
plt.figure(figsize = (12, 7))
plt.xticks(rotation=60)
sns.countplot(x=Y, hue=Y)
```

```
<Axes: ylabel='count'>
```




```
class_labels = LabelBinarizer()
Y = class_labels.fit_transform(Y)
pickle.dump(class_labels,open('/content/drive/MyDrive/handcrafted_image_classification/label_transform.pkl', 'wb'))
n_classes = len(class_labels.classes_)
```

```
cls = len(class_labels.classes_)
print(class_labels.classes_)
```

```
['Building' 'Forest' 'Glacier' 'Mountains' 'Sea' 'Streets']
```

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, stratify=Y)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1, stratify=y_train)
```

EfficientNetB2

```
base_model = EfficientNetB2(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
```

```
Downloading data from https://github.com/Callidior/keras-applications/releases/download/efficientnet/efficientnet-b2_weights_tf_dim
31936256/31936256 [==============================] - 0s 0us/step
```

```
model = base_model.output
model = GlobalAveragePooling2D()(model)
model = Dense(cls, activation='softmax')(model)
model = Model(inputs = base_model.input, outputs=model)
```

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

```
model.summary()
```

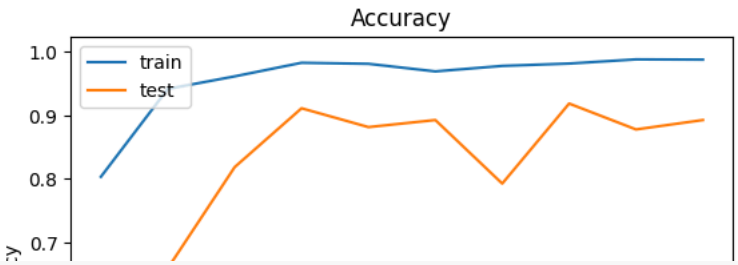
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0         []

 stem_conv (Conv2D)          (None, 64, 64, 32)        864       ['input_1[0][0]']

 stem_bn (BatchNormalizatio  (None, 64, 64, 32)        128       ['stem_conv[0][0]']
 n)

 stem_activation (Activatio  (None, 64, 64, 32)        0         ['stem_bn[0][0]']
 n)

 block1a_dwconv (DepthwiseC  (None, 64, 64, 32)        288       ['stem_activation[0][0]']
 onv2D)

 block1a_bn (BatchNormaliza  (None, 64, 64, 32)        128       ['block1a_dwconv[0][0]']
 tion)

 block1a_activation (Activa  (None, 64, 64, 32)        0         ['block1a_bn[0][0]']
 tion)

 block1a_se_squeeze (Global  (None, 32)                0         ['block1a_activation[0][0]']
 AveragePooling2D)

 block1a_se_reshape (Reshap  (None, 1, 1, 32)          0         ['block1a_se_squeeze[0][0]']
 e)

 block1a_se_reduce (Conv2D)  (None, 1, 1, 8)           264       ['block1a_se_reshape[0][0]']

 block1a_se_expand (Conv2D)  (None, 1, 1, 32)          288       ['block1a_se_reduce[0][0]']

 block1a_se_excite (Multipl  (None, 64, 64, 32)        0         ['block1a_activation[0][0]',
 y)                                                               'block1a_se_expand[0][0]']

 block1a_project_conv (Conv  (None, 64, 64, 16)        512       ['block1a_se_excite[0][0]']
 2D)

 block1a_project_bn (BatchN  (None, 64, 64, 16)        64        ['block1a_project_conv[0][0]']
 ormalization)

 block1b_dwconv (DepthwiseC  (None, 64, 64, 16)        144       ['block1a_project_bn[0][0]']
 onv2D)

 block1b_bn (BatchNormaliza  (None, 64, 64, 16)        64        ['block1b_dwconv[0][0]']
 tion)

 block1b_activation (Activa  (None, 64, 64, 16)        0         ['block1b_bn[0][0]']
 tion)

 block1b_se_squeeze (Global  (None, 16)                0         ['block1b_activation[0][0]']
 AveragePooling2D)

 block1b_se_reshape (Reshap  (None, 1, 1, 16)          0         ['block1b_se_squeeze[0][0]']
 e)

 block1b_se_reduce (Conv2D)  (None, 1, 1, 4)           68        ['block1b_se_reshape[0][0]']
```

```
history = model.fit(x=x_train, y=y_train, batch_size=64, epochs=10, validation_data=(x_val, y_val))
```

```
Epoch 1/10
38/38 [==============================] - 88s 394ms/step - loss: 0.5556 - accuracy: 0.8033 - val_loss: 3.1004 - val_accuracy: 0.2815
Epoch 2/10
38/38 [==============================] - 9s 234ms/step - loss: 0.1859 - accuracy: 0.9416 - val_loss: 1.2995 - val_accuracy: 0.6593
Epoch 3/10
38/38 [==============================] - 9s 236ms/step - loss: 0.1139 - accuracy: 0.9613 - val_loss: 0.9138 - val_accuracy: 0.8185
Epoch 4/10
38/38 [==============================] - 9s 238ms/step - loss: 0.0564 - accuracy: 0.9827 - val_loss: 0.3663 - val_accuracy: 0.9111
Epoch 5/10
38/38 [==============================] - 9s 239ms/step - loss: 0.0619 - accuracy: 0.9811 - val_loss: 0.6742 - val_accuracy: 0.8815
Epoch 6/10
38/38 [==============================] - 9s 241ms/step - loss: 0.1057 - accuracy: 0.9691 - val_loss: 0.5299 - val_accuracy: 0.8926
Epoch 7/10
38/38 [==============================] - 9s 241ms/step - loss: 0.0810 - accuracy: 0.9778 - val_loss: 0.9396 - val_accuracy: 0.7926
Epoch 8/10
38/38 [==============================] - 10s 251ms/step - loss: 0.0542 - accuracy: 0.9815 - val_loss: 0.4461 - val_accuracy: 0.9185
Epoch 9/10
38/38 [==============================] - 9s 242ms/step - loss: 0.0385 - accuracy: 0.9881 - val_loss: 0.6160 - val_accuracy: 0.8778
Epoch 10/10
38/38 [==============================] - 9s 244ms/step - loss: 0.0365 - accuracy: 0.9877 - val_loss: 0.8053 - val_accuracy: 0.8926
```

```
model.save("/content/drive/MyDrive/handcrafted_image_classification/model/EfficientNetB2.h5")
```

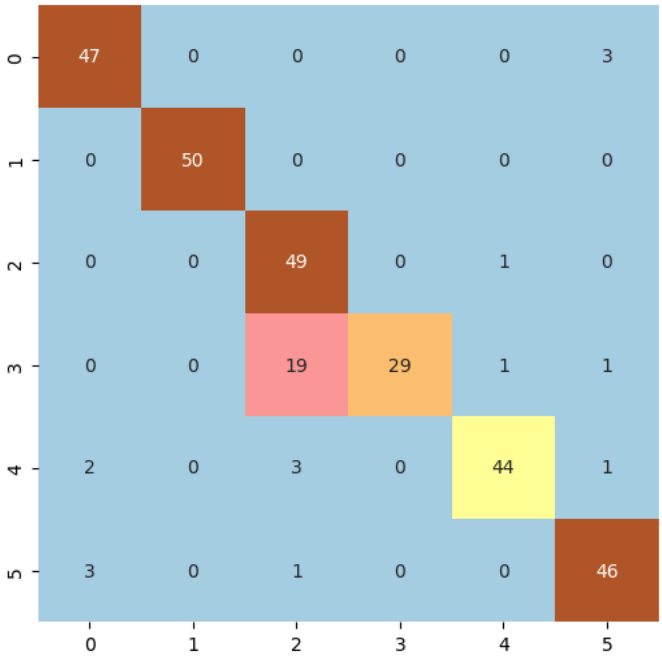
```
#accuracy and loss plot

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()


#loss plot
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```




```
pred = model.predict(x_test)
pred = np.argmax(pred,axis=1)
y_test_new = np.argmax(y_test,axis=1)
```

```
10/10 [==============================] - 4s 146ms/step
```

```
cmat = confusion_matrix(y_test_new,pred)
plt.figure(figsize=(6,6))
sns.heatmap(cmat, annot = True, cbar = False, cmap='Paired', fmt="d");
```




```
print(classification_report(y_test_new,pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        50
           1       1.00      1.00      1.00        50
           2       0.68      0.98      0.80        50
           3       1.00      0.58      0.73        50
           4       0.96      0.88      0.92        50
           5       0.90      0.92      0.91        50

    accuracy                           0.88       300
   macro avg       0.91      0.88      0.88       300
weighted avg       0.91      0.88      0.88       300
```