```
!pip install keras_tuner
!pip install bayesian-optimization
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting keras_tuner
  Downloading keras_tuner-1.3.5-py3-none-any.whl (176 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 176.1/176.1 kB 8.2 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (23.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras_tuner) (2.27.1)
Collecting kt-legacy (from keras_tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (1.26.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (2022.12.
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (2
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras_tuner) (3.4)
Installing collected packages: kt-legacy, keras_tuner
Successfully installed keras_tuner-1.3.5 kt-legacy-1.0.5
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting bayesian-optimization
  Downloading bayesian_optimization-1.4.3-py3-none-any.whl (18 kB)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.22.4)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.10.1)
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.2.2)
Collecting colorama>=0.4.6 (from bayesian-optimization)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian-optimi
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian
Installing collected packages: colorama, bayesian-optimization
Successfully installed bayesian-optimization-1.4.3 colorama-0.4.6
```

```
#!git clone https://github.com/CorbenYkt/imageclassification.git
```

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

```
!ls
```

```
drive  sample_data
```

```
import pandas as pd
import numpy as np
import os
import seaborn as sn; sn.set(font_scale=1.4)
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import os
import random

import kerastuner as kt

from keras.layers import Input
from bayes_opt import BayesianOptimization
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

```
<ipython-input-5-5af3f7e24a7a>:11: DeprecationWarning: `import kerastuner` is deprecated, please use `import keras_tuner`.
  import kerastuner as kt
```

```
class_names=['buildings', 'forest', 'glacier','mountain','sea','street']
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}
nb_classes=len(class_names)
print(class_names_label)
image_size=(150,150)
work_path='/content/drive/Othercomputers/Thinkpad/Project'
```

```
{'buildings': 0, 'forest': 1, 'glacier': 2, 'mountain': 3, 'sea': 4, 'street': 5}
```

```
#load dataset
def load_data():
    #directory=os.path.abspath("")
    directory=work_path + '/dataset'
    #print(directory)
    category=['seg_train','seg_test']
    output=[]
    for eachcategory in category:
        #print(eachcategory)
        path=os.path.join(directory,eachcategory)
        #print(path)
        images=[]
        labels=[]
        #print('Loading {}'. format(eachcategory) + '...')

        for folder in os.listdir(path):
            label=class_names_label[folder]
            for file in os.listdir(os.path.join(path, folder)):
                img_path=os.path.join(os.path.join(path,folder),file)
                #print(img_path)
                image=cv2.imread(img_path)
                image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image=cv2.resize(image,image_size)

                images.append(image)
                labels.append(label)
        images=np.array(images, dtype='float32')
        labels=np.array(labels, dtype='int32')

        output.append((images,labels))
    return output


(train_images, train_labels), (test_images, test_labels) = load_data()
train_x, val_x, train_y, val_y = train_test_split(train_images, train_labels, stratify=train_labels, random_state=48, test_size=0.05)
(test_x, test_y)=(test_images, test_labels)


train_images, train_labels = shuffle(train_images, train_labels, random_state=25)
print(len(train_x))
print(len(val_x))
print(len(test_x))
```

```
13281
700
2964
```

```
def display_example(class_name, images, labels):
    figsize=(15,15)
    fig=plt.figure(figsize=figsize)
    #fig.subtitle("Some examples of images from the datset", fontsize=16) - deprecated?
    for i in range(10):
        plt.subplot(5,5, i+1)
        plt.yticks([])
        plt.xticks([])
        plt.grid(False)
        #image=cv2.resize(images[i], figsize)
        #plt.imshow(image.astype(np.uint8))
        plt.imshow(images[i].astype(np.uint8))
        plt.xlabel(class_names[labels[i]],fontsize = 8)
    plt.show()

display_example(class_names,train_images,train_labels)
```

```
train_x = train_x / 255.0
val_x = val_x / 255.0
test_x = test_x / 255.0

train_y = to_categorical(train_y)
val_y = to_categorical(val_y)
test_y = to_categorical(test_y)
```

```
print(train_x.shape)
print(train_y.shape)
print(val_x.shape)
print(val_y.shape)
print(test_x.shape)
print(test_y.shape)
```

```
(13281, 150, 150, 3)
(13281, 6)
(700, 150, 150, 3)
(700, 6)
(2964, 150, 150, 3)
(2964, 6)
```

```
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(150, 150, 3)))

    for i in range(hp.Int('num_blocks', 1, 2)):
        hp_padding=hp.Choice('padding_'+ str(i), values=['valid', 'same'])
        hp_filters=hp.Choice('filters_'+ str(i), values=[32, 64])

        model.add(Conv2D(hp_filters, (3, 3), padding=hp_padding, activation='relu', kernel_initializer='he_uniform', input_shape=(150, 15
        model.add(MaxPooling2D((2, 2)))
        model.add(Dropout(hp.Choice('dropout_'+ str(i), values=[0.0, 0.1, 0.2])))

    model.add(Flatten())

    hp_units = hp.Int('units', min_value=16, max_value=256, step=16)
    #hp_units = hp.Int('units', min_value=25, max_value=150, step=25)
    model.add(Dense(hp_units, activation='relu', kernel_initializer='he_uniform'))

    model.add(Dense(6,activation="softmax"))

    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
    hp_optimizer=hp.Choice('Optimizer', values=['Adam', 'SGD'])

    if hp_optimizer == 'Adam':
        hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
    elif hp_optimizer == 'SGD':
        hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3])
        nesterov=True
        momentum=0.9

    model.compile(optimizer=hp_optimizer,loss='categorical_crossentropy', metrics=['accuracy'])
    return model


#This part of code is not needed to run everytime.
#all values of necessary hyperparameters are stored in bestHP variable
#----------------
# tuner_cnn = kt.tuners.BayesianOptimization(
#     build_model,
#     objective='val_loss',
#     max_trials=100,
#     directory='.',
#     project_name='tuning-cnn')

# tuner_cnn.search(train_x, train_y,
#              validation_data= (val_x,val_y),
#              epochs=30,
#              callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])

# print("________")
# bestHP = tuner_cnn.get_best_hyperparameters(num_trials=1)[0]
# print(bestHP)
```

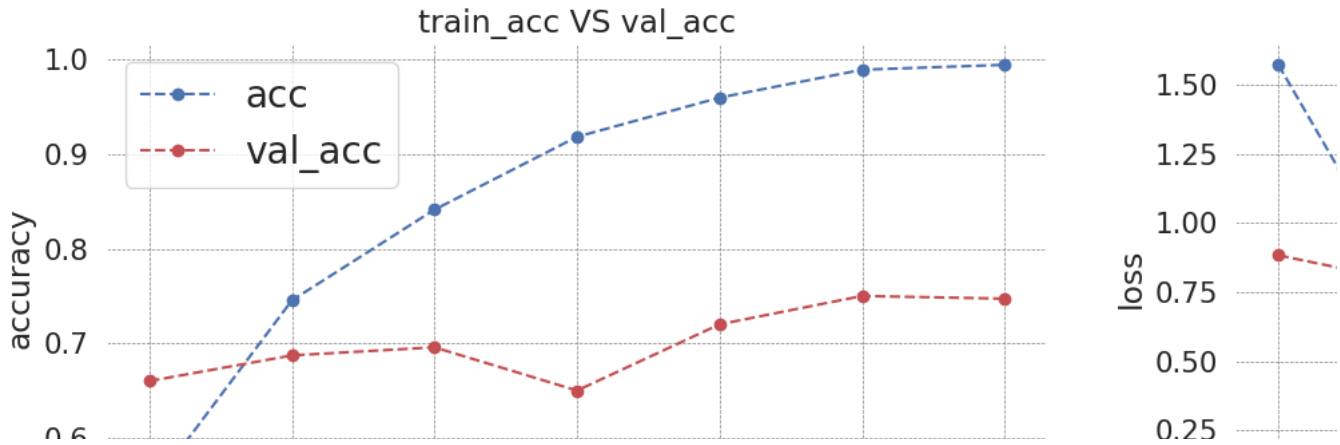
```
#bestHP.values


from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
import h5py


erl_stop = EarlyStopping(monitor='val_loss', patience = 5, restore_best_weights=True)
mod_chk1 = ModelCheckpoint(filepath='model1.hdf5', monitor='val_loss', save_best_only=True)
mod_chk2 = ModelCheckpoint(filepath='model2.hdf5', monitor='val_loss', save_best_only=True)
mod_chk3 = ModelCheckpoint(filepath='model2.hdf5', monitor='val_loss', save_best_only=True)
# checkpoint_path=work_path + "/model-{epoch:02d}-{val_loss:.4f}.hdf5"
# checkpoint=ModelCheckpoint(
#     filepath=checkpoint_path,
#     monitor='val_loss',
#     mode='max',
#     save_best_only=True,
#     verbose=1
# )
lr_rate = ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.1)


#This values was discovered in previus iteration and using Bayesian optimization
bestHP = {'num_blocks': 2, 'padding_0': 'same', 'filters_0': 32, 'dropout_0': 0.0, 'units': 256, 'learning_rate': 0.001, 'Optimizer': 'Ad


model_cnn = Sequential()
model_cnn.add(Input(shape=(150, 150, 3)))
for i in range(bestHP['num_blocks']):
  hp_padding=bestHP['padding_'+ str(i)]
  hp_filters=bestHP['filters_'+ str(i)]
  model_cnn.add(Conv2D(hp_filters, (3, 3), padding=hp_padding, activation='relu', kernel_initializer='he_uniform', input_shape=(150, 150,
  model_cnn.add(MaxPooling2D((2, 2)))
  model_cnn.add(Dropout(bestHP['dropout_'+ str(i)]))
model_cnn.add(Flatten())
model_cnn.add(Dense(bestHP['units'], activation='relu', kernel_initializer='he_uniform'))
model_cnn.add(Dense(6,activation="softmax"))
model_cnn.compile(optimizer=bestHP['Optimizer'],
                                            loss='categorical_crossentropy',
                                            metrics=['accuracy'])
print(model_cnn.summary())
history_cnn= model_cnn.fit(train_x, train_y, epochs=50, batch_size=32, validation_data=(val_x, val_y), callbacks=[erl_stop, mod_chk1, lr_
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 150, 150, 32)      896

 max_pooling2d_8 (MaxPooling  (None, 75, 75, 32)       0
 2D)

 dropout_11 (Dropout)        (None, 75, 75, 32)        0

 conv2d_9 (Conv2D)           (None, 73, 73, 32)        9248

 max_pooling2d_9 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 dropout_12 (Dropout)        (None, 36, 36, 32)        0

 flatten_4 (Flatten)         (None, 41472)             0

 dense_14 (Dense)            (None, 256)               10617088

 dense_15 (Dense)            (None, 6)                 1542

=================================================================
Total params: 10,628,774
Trainable params: 10,628,774
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
416/416 [==============================] - 5s 9ms/step - loss: 1.5734 - accuracy: 0.5442 - val_loss: 0.8832 - val_accuracy: 0.6600
Epoch 2/50
416/416 [==============================] - 3s 8ms/step - loss: 0.6806 - accuracy: 0.7453 - val_loss: 0.7767 - val_accuracy: 0.6871
Epoch 3/50
416/416 [==============================] - 3s 8ms/step - loss: 0.4390 - accuracy: 0.8412 - val_loss: 0.9948 - val_accuracy: 0.6957
Epoch 4/50
416/416 [==============================] - 3s 8ms/step - loss: 0.2348 - accuracy: 0.9182 - val_loss: 1.1036 - val_accuracy: 0.6500
Epoch 5/50
416/416 [==============================] - 3s 8ms/step - loss: 0.1316 - accuracy: 0.9596 - val_loss: 1.0628 - val_accuracy: 0.7200
Epoch 6/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0476 - accuracy: 0.9892 - val_loss: 1.0427 - val_accuracy: 0.7500
```

```
Epoch 7/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0305 - accuracy: 0.9944 - val_loss: 1.0729 - val_accuracy: 0.7471
```

```
def plot_accuracy_loss(history):
    fig=plt.figure(figsize=(20,10))
    plt.subplot(221)
    plt.plot(history.history['accuracy'], 'bo--', label='acc')
    plt.plot(history.history['val_accuracy'], 'ro--', label='val_acc')
    plt.title('train_acc VS val_acc')
    plt.ylabel('accuracy')
    plt.xlabel('epochs')
    plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
    plt.rcParams['axes.facecolor'] = 'white'
    plt.legend(fontsize = "large")
    plt.subplot(222)
    plt.plot(history.history['loss'], 'bo--', label='loss')
    plt.plot(history.history['val_loss'], 'ro--', label='val_loss')
    plt.title('train_loss VS val_loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.grid(color = 'grey', linestyle = '--', linewidth = 0.5)
    plt.rcParams['axes.facecolor'] = 'white'
    plt.legend()
    plt.show()
```

```
plot_accuracy_loss(history_cnn)
```




```
model_cnn.save(work_path + '/model1.h5')
```

```
print("We have " + str(len(train_x)) + " of images in " + str(len(class_names_label)) + " classes")
print("And " + str(len(test_x)) + " test images")
```

```
We have 13281 of images in 6 classes
And 2964 test images
```

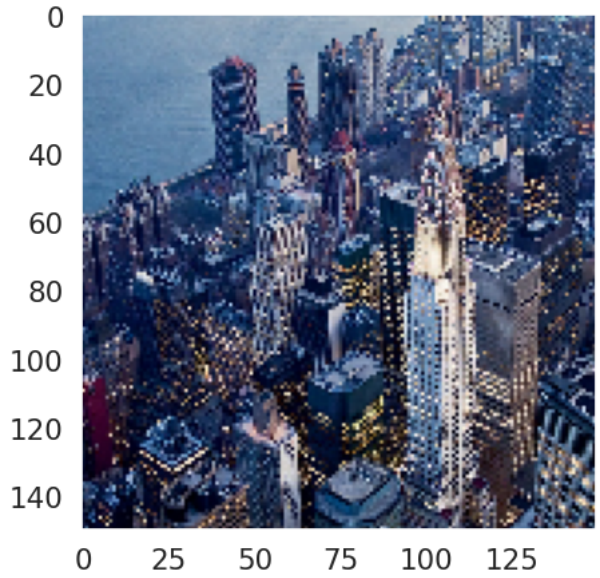
```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
pred_images = model_cnn.predict(test_images) #this is vector of probabilities
pred_labels = np.argmax(pred_images, axis=1) #take the highest one prob.
print(classification_report(test_labels, pred_labels))
```

```
93/93 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

           0       0.55      0.81      0.65       436
           1       0.96      0.75      0.84       472
           2       0.62      0.88      0.73       537
           3       0.72      0.57      0.64       525
           4       0.74      0.47      0.57       494
           5       0.74      0.67      0.71       500

    accuracy                           0.69      2964
   macro avg       0.72      0.69      0.69      2964
weighted avg       0.72      0.69      0.69      2964
```

```
#Precision is the ratio of the correctly +ve labeled by our program to all +ve labeled
#Precision answers the following: How many of those who we labeled as diabetic are actually diabetic
#Recall is the ratio of the correctly +ve labeled by our program to all who are diabetic in reality.
#Recall answers the following question: Of all the people who are diabetic, how many of those we correctly predict?
```

```
#F1 Score considers both precision and recall.
#It is the harmonic mean(average) of the precision and recall


directory=os.path.abspath("")
test_image_filename= work_path + '/testimage1.jpg'
img_path=os.path.join(directory,test_image_filename)
print(img_path)
image=cv2.imread(img_path)
image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image=cv2.resize(image,image_size)

figsize=(5,5)
fig=plt.figure(figsize=figsize)
plt.imshow(image)
plt.grid(False)
plt.show
image.reshape

y_pred=model_cnn.predict(image.reshape(1, 150,150,3))
pred_label = np.argmax(y_pred, axis=1)
print("Results of image classification:", y_pred)
np.set_printoptions(suppress=True)
print("Rounded predict values:", np.around(y_pred, decimals=2))
print("Our image belongs to class:", class_names[pred_label[0]])
```

```
/content/drive/Othercomputers/Thinkpad/Project/testimage1.jpg
1/1 [==============================] - 0s 117ms/step
Results of image classification: [[1.0000000e+00 0.0000000e+00 5.7962183e-38 9.0958275e-25 0.0000000e+00
  0.0000000e+00]]
Rounded predict values: [[1. 0. 0. 0. 0. 0.]]
Our image belongs to class: buildings
```



Now lets do some another model, and after that lets combine them to Ensemble

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import MaxPooling2D, GlobalAveragePooling2D

base_model=VGG16(
    input_shape=(150,150,3),
    weights='imagenet',
    include_top=False
    )

for layer in base_model.layers[:10]:
    layer.trainable=False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.2)(x)
predictions = Dense(6, activation='softmax')(x)
model2 = Model(inputs=base_model.inputs, outputs=predictions)
```

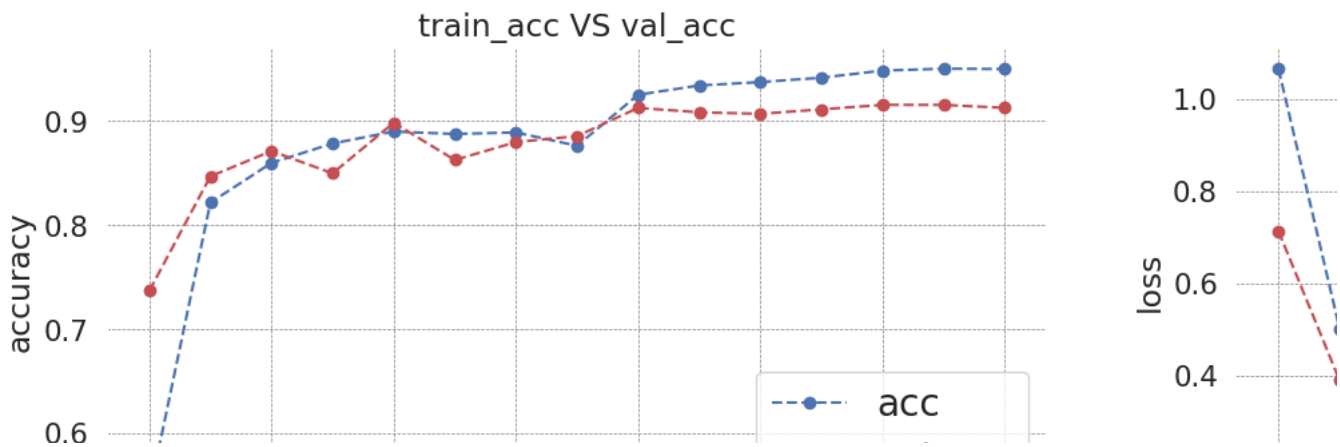
```
model2.save(work_path + '/model2.h5')
```

```
model2.compile(optimizer=bestHP['Optimizer'], loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history2= model2.fit(train_x, train_y, epochs=50, batch_size=32, validation_data=(val_x, val_y), callbacks=[erl_stop, mod_chk2, lr_rate])
```

```
Epoch 1/50
414/416 [============================>.] - ETA: 0s - loss: 1.0665 - accuracy: 0.5418
Epoch 1: val_loss did not improve from 1.79081
416/416 [==============================] - 10s 17ms/step - loss: 1.0652 - accuracy: 0.5424 - val_loss: 0.7134 - val_accuracy
Epoch 2/50
416/416 [==============================] - ETA: 0s - loss: 0.5025 - accuracy: 0.8219
Epoch 2: val_loss did not improve from 1.79081
416/416 [==============================] - 7s 16ms/step - loss: 0.5025 - accuracy: 0.8219 - val_loss: 0.3901 - val_accuracy:
Epoch 3/50
415/416 [============================>.] - ETA: 0s - loss: 0.4056 - accuracy: 0.8600
Epoch 3: val_loss did not improve from 1.79081
416/416 [==============================] - 7s 16ms/step - loss: 0.4056 - accuracy: 0.8600 - val_loss: 0.3757 - val_accuracy:
Epoch 4/50
412/416 [============================>.] - ETA: 0s - loss: 0.3543 - accuracy: 0.8787
Epoch 4: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.3544 - accuracy: 0.8787 - val_loss: 0.4389 - val_accuracy:
Epoch 5/50
413/416 [============================>.] - ETA: 0s - loss: 0.3253 - accuracy: 0.8901
Epoch 5: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.3251 - accuracy: 0.8901 - val_loss: 0.3329 - val_accuracy:
Epoch 6/50
412/416 [============================>.] - ETA: 0s - loss: 0.3293 - accuracy: 0.8878
Epoch 6: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.3286 - accuracy: 0.8878 - val_loss: 0.4153 - val_accuracy:
Epoch 7/50
415/416 [============================>.] - ETA: 0s - loss: 0.3324 - accuracy: 0.8894
Epoch 7: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.3325 - accuracy: 0.8893 - val_loss: 0.3769 - val_accuracy:
Epoch 8/50
416/416 [==============================] - ETA: 0s - loss: 0.3730 - accuracy: 0.8764
Epoch 8: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.3730 - accuracy: 0.8764 - val_loss: 0.3332 - val_accuracy:
Epoch 9/50
413/416 [============================>.] - ETA: 0s - loss: 0.2074 - accuracy: 0.9255
Epoch 9: val_loss did not improve from 1.79081
416/416 [==============================] - 7s 16ms/step - loss: 0.2072 - accuracy: 0.9255 - val_loss: 0.2768 - val_accuracy:
Epoch 10/50
416/416 [==============================] - ETA: 0s - loss: 0.1904 - accuracy: 0.9343
Epoch 10: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.1904 - accuracy: 0.9343 - val_loss: 0.2911 - val_accuracy:
Epoch 11/50
415/416 [============================>.] - ETA: 0s - loss: 0.1769 - accuracy: 0.9376
Epoch 11: val_loss did not improve from 1.79081
416/416 [==============================] - 7s 16ms/step - loss: 0.1769 - accuracy: 0.9376 - val_loss: 0.2887 - val_accuracy:
Epoch 12/50
413/416 [============================>.] - ETA: 0s - loss: 0.1656 - accuracy: 0.9416
Epoch 12: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.1652 - accuracy: 0.9418 - val_loss: 0.2816 - val_accuracy:
Epoch 13/50
413/416 [============================>.] - ETA: 0s - loss: 0.1487 - accuracy: 0.9486
Epoch 13: val_loss did not improve from 1.79081
416/416 [==============================] - 6s 16ms/step - loss: 0.1490 - accuracy: 0.9486 - val_loss: 0.2795 - val_accuracy:
Epoch 14/50
416/416 [==============================] - ETA: 0s - loss: 0.1456 - accuracy: 0.9505
Epoch 14: val_loss did not improve from 1.79081
```

```
plot_accuracy_loss(history2)
```



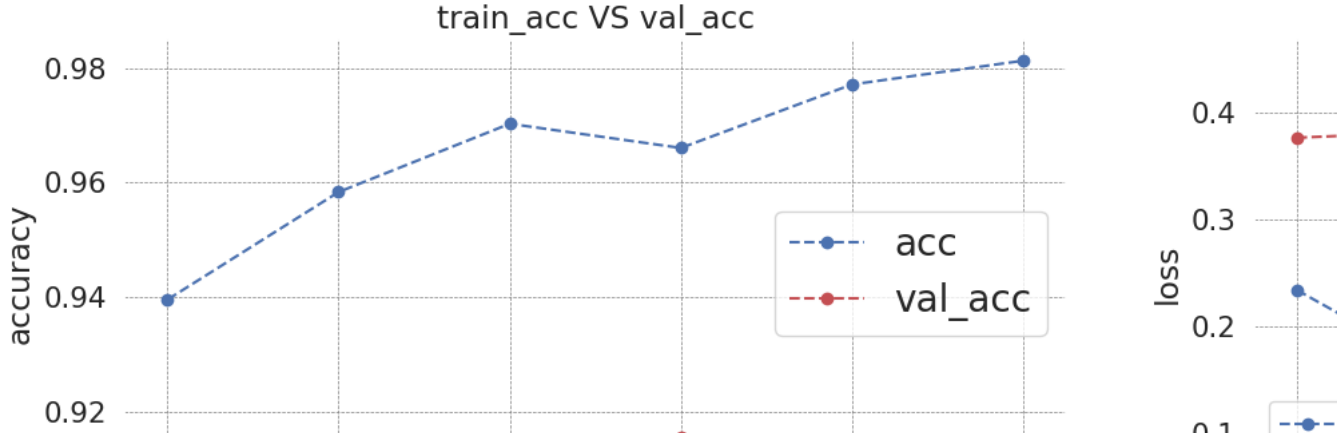
```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
pred_images = model2.predict(test_images) #this is vector of probabilities
pred_labels = np.argmax(pred_images, axis=1) #take the highest one prob.
print(classification_report(test_labels, pred_labels))
```

```
93/93 [==============================] - 2s 15ms/step
              precision    recall  f1-score   support

           0       0.64      0.97      0.77       436
           1       0.99      0.90      0.94       472
           2       0.82      0.72      0.77       537
           3       0.72      0.85      0.78       525
           4       0.96      0.66      0.78       494
           5       0.83      0.73      0.78       500

    accuracy                           0.80      2964
   macro avg       0.83      0.81      0.80      2964
weighted avg       0.83      0.80      0.80      2964
```

```
#now lets load two different models (model_cnn and model2)


from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Input, Average
#look at the logs correct model names!!!
model_1=load_model(work_path + '/model1.h5')
model_1 = Model(inputs=model_1.inputs,
                outputs=model_1.outputs,
                name='model1')

model_2=load_model(work_path + '/model2.h5')
model_2 = Model(inputs=model_2.inputs,
                outputs=model_2.outputs,
                name='model2')

models = [model_1, model_2]
model_input = Input(shape=(150, 150, 3))
model_outputs = [model(model_input) for model in models]
ensemble_output = Average()(model_outputs)
ensemble_model = Model(inputs=model_input, outputs=ensemble_output, name='ensemble')

ensemble_model.compile(optimizer=bestHP['Optimizer'], loss='categorical_crossentropy', metrics=['accuracy'])


history=ensemble_model.fit(train_x, train_y, epochs=50, batch_size=256, validation_data=(val_x, val_y), callbacks=[erl_stop, mod_chk3, lr
```

```
Epoch 1/50
52/52 [==============================] - 12s 143ms/step - loss: 0.2340 - accuracy: 0.9396 - val_loss: 0.3764 - val_accuracy: 0.9086
Epoch 2/50
52/52 [==============================] - 6s 123ms/step - loss: 0.1445 - accuracy: 0.9584 - val_loss: 0.3832 - val_accuracy: 0.9129
Epoch 3/50
52/52 [==============================] - 6s 123ms/step - loss: 0.1000 - accuracy: 0.9703 - val_loss: 0.4009 - val_accuracy: 0.9071
Epoch 4/50
52/52 [==============================] - 6s 123ms/step - loss: 0.0950 - accuracy: 0.9660 - val_loss: 0.4338 - val_accuracy: 0.9157
Epoch 5/50
52/52 [==============================] - 6s 123ms/step - loss: 0.0619 - accuracy: 0.9771 - val_loss: 0.4484 - val_accuracy: 0.9100
Epoch 6/50
52/52 [==============================] - 6s 124ms/step - loss: 0.0510 - accuracy: 0.9813 - val_loss: 0.4347 - val_accuracy: 0.9100
```

```
plot_accuracy_loss(history)
```




```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
```

```
pred_images = ensemble_model.predict(test_images) #this is vector of probabilities
pred_labels = np.argmax(pred_images, axis=1) #take the highest one prob.
print(classification_report(test_labels, pred_labels))
```

```
93/93 [==============================] - 1s 12ms/step
              precision    recall  f1-score   support

           0       0.55      0.98      0.70       436
           1       0.95      0.91      0.93       472
           2       0.70      0.84      0.76       537
           3       0.76      0.70      0.73       525
           4       0.94      0.58      0.72       494
           5       0.93      0.56      0.70       500

    accuracy                           0.76      2964
   macro avg       0.81      0.76      0.76      2964
weighted avg       0.81      0.76      0.76      2964
```

✓ 1s completed at 10:15 AM