```
!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 4.4 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
```

Double-click (or enter) to edit

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

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

```
Mounted at /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-5-122b315d6062>:10: 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'
    category=['seg_train','seg_test']
    output=[]
    for eachcategory in category:
        path=os.path.join(directory,eachcategory)
        images=[]
        labels=[]

        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)
                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, 150, 3)))
#         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)
#     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)
```

```
Trial 5 Complete [00h 00m 31s]
val_loss: 0.6786908507347107

Best val_loss So Far: 0.6786908507347107
Total elapsed time: 00h 02m 21s

Search: Running Trial #6

Value             |Best Value So Far |Hyperparameter
1                 |2                 |num_blocks
valid             |valid             |padding_0
32                |32                |filters_0
0.1               |0.1               |dropout_0
192               |224               |units
0.001             |0.001             |learning_rate
Adam              |SGD               |Optimizer
valid             |same              |padding_1
64                |32                |filters_1
0                 |0                 |dropout_1

Epoch 1/30
--------------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-23-4344b9a89c58> in <cell line: 11>()
      9     project_name='tuning-cnn')
     10
---> 11 tuner_cnn.search(train_x, train_y,
     12              validation_data= (val_x,val_y),
     13              epochs=30,

                                 14 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ct
     50   try:
     51     ctx.ensure_initialized()
---> 52     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                         inputs, attrs, num_outputs)
     54   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

```
#bestHP.values
```

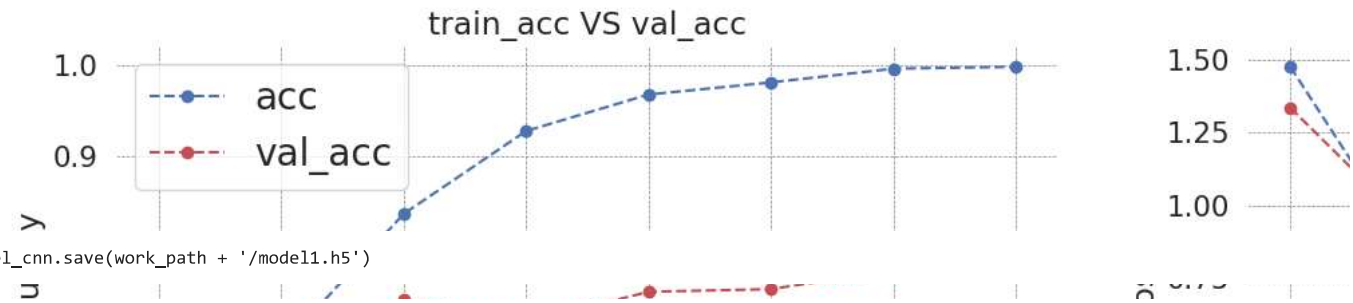
```
{'num_blocks': 2,
 'padding_0': 'same',
 'filters_0': 32,
 'dropout_0': 0.0,
 'units': 256,
 'learning_rate': 0.001,
 'Optimizer': 'Adam',
 'padding_1': 'valid',
 'filters_1': 32,
 'dropout_1': 0.1}
```

```
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)
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, 3)))
  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_rate])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

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

 conv2d_1 (Conv2D)           (None, 73, 73, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 36, 36, 32)        0

 flatten (Flatten)           (None, 41472)             0

 dense (Dense)               (None, 256)               10617088

 dense_1 (Dense)             (None, 6)                 1542

=================================================================
Total params: 10,628,774
Trainable params: 10,628,774
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
416/416 [==============================] - 19s 16ms/step - loss: 1.4760 - accuracy: 0.5811 - val_loss: 1.3352 - val_accuracy: 0.528
Epoch 2/50
416/416 [==============================] - 6s 14ms/step - loss: 0.8268 - accuracy: 0.6885 - val_loss: 0.9202 - val_accuracy: 0.6486
Epoch 3/50
416/416 [==============================] - 6s 14ms/step - loss: 0.4597 - accuracy: 0.8358 - val_loss: 0.8153 - val_accuracy: 0.7414
Epoch 4/50
416/416 [==============================] - 6s 13ms/step - loss: 0.2202 - accuracy: 0.9271 - val_loss: 1.0664 - val_accuracy: 0.7157
Epoch 5/50
416/416 [==============================] - 6s 13ms/step - loss: 0.1101 - accuracy: 0.9674 - val_loss: 1.0491 - val_accuracy: 0.7500
Epoch 6/50
416/416 [==============================] - 6s 13ms/step - loss: 0.0738 - accuracy: 0.9807 - val_loss: 1.2252 - val_accuracy: 0.7529
Epoch 7/50
416/416 [==============================] - 6s 13ms/step - loss: 0.0228 - accuracy: 0.9959 - val_loss: 1.1554 - val_accuracy: 0.7743
Epoch 8/50
416/416 [==============================] - 6s 14ms/step - loss: 0.0141 - accuracy: 0.9979 - val_loss: 1.1915 - val_accuracy: 0.7757
```

```
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')
```

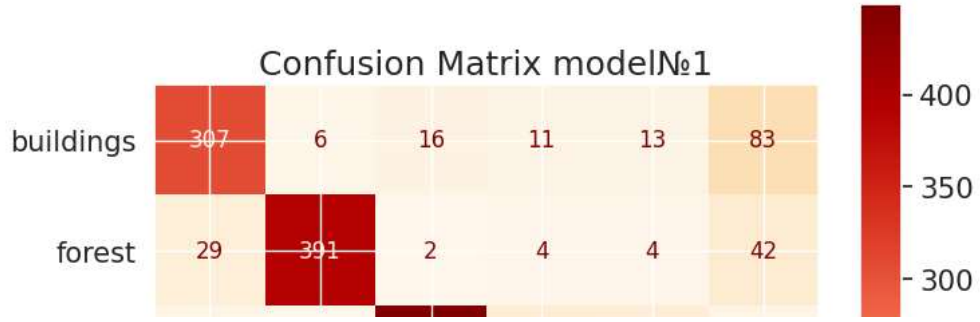
```
print("We have " + str(len(train_x)) + " of images in " + str(len(class_names_label)) + " classes")
print("And " + str(len(test_x)) + " test images")
```

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

```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix, ConfusionMatrixDisplay
pred_images = model_cnn.predict(test_images)
pred_labels = np.argmax(pred_images, axis=1)
cm = confusion_matrix(test_labels, pred_labels)
print(cm)
print('\n')
print(classification_report(test_labels, pred_labels))
```

```
93/93 [==============================] - 1s 6ms/step
[[307   6  16  11  13  83]
 [ 29 391   2   4   4  42]
 [  8   1 449  38  33   8]
 [ 37   2 140 232 100  14]
 [ 42   1 135  52 247  17]
 [ 86  16  15   1   5 377]]


              precision    recall  f1-score   support

           0       0.60      0.70      0.65       436
           1       0.94      0.83      0.88       472
           2       0.59      0.84      0.69       537
           3       0.69      0.44      0.54       525
           4       0.61      0.50      0.55       494
           5       0.70      0.75      0.72       500

    accuracy                           0.68      2964
   macro avg       0.69      0.68      0.67      2964
weighted avg       0.69      0.68      0.67      2964
```

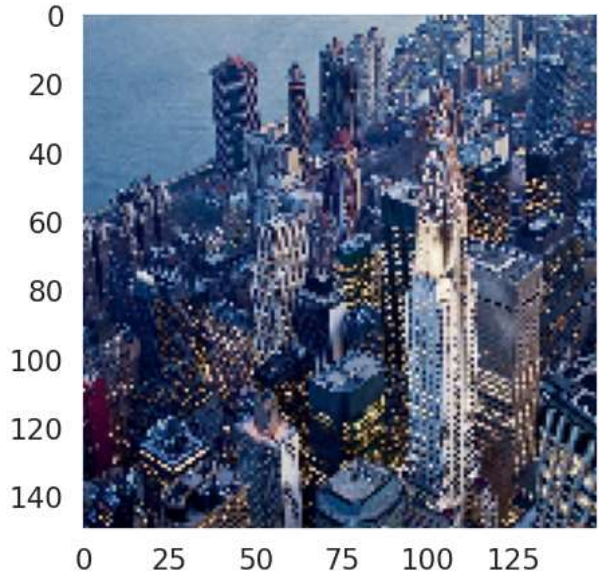
```
plt.rcParams['figure.figsize'] = (8, 8)
plt.rcParams['font.size'] = 12
display_c_m = ConfusionMatrixDisplay(cm, display_labels=class_names)
display_c_m.plot(cmap='OrRd', xticks_rotation=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Confusion Matrix model№1', fontsize=18)
plt.savefig('confusion_matrix_model1.png', transparent=True, dpi=300)
plt.show()
```




```
#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 86ms/step
Results of image classification: [[1. 0. 0. 0. 0. 0.]]
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
```

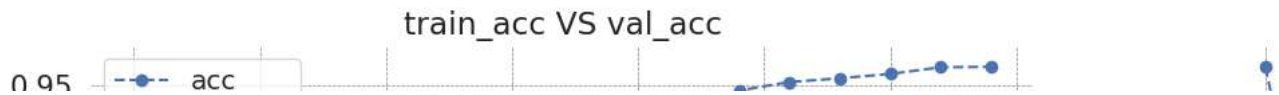
```
base_model=VGG16(
    input_shape=(150,150,3),
    weights='imagenet',
    include_top=False
    )

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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n
58889256/58889256 [==============================] - 0s 0us/step
```

```
#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
416/416 [==============================] - 23s 44ms/step - loss: 0.9491 - accuracy: 0.6100 - val_loss: 0.5603 - val_accuracy: 0.817
Epoch 2/50
416/416 [==============================] - 17s 40ms/step - loss: 0.4817 - accuracy: 0.8306 - val_loss: 0.4431 - val_accuracy: 0.842
Epoch 3/50
416/416 [==============================] - 17s 40ms/step - loss: 0.3972 - accuracy: 0.8618 - val_loss: 0.3604 - val_accuracy: 0.884
Epoch 4/50
416/416 [==============================] - 16s 39ms/step - loss: 0.3541 - accuracy: 0.8801 - val_loss: 0.4066 - val_accuracy: 0.861
Epoch 5/50
416/416 [==============================] - 16s 39ms/step - loss: 0.3298 - accuracy: 0.8886 - val_loss: 0.4095 - val_accuracy: 0.867
Epoch 6/50
416/416 [==============================] - 17s 40ms/step - loss: 0.3006 - accuracy: 0.8963 - val_loss: 0.3297 - val_accuracy: 0.895
Epoch 7/50
416/416 [==============================] - 16s 39ms/step - loss: 0.3121 - accuracy: 0.8924 - val_loss: 0.5989 - val_accuracy: 0.815
Epoch 8/50
416/416 [==============================] - 17s 40ms/step - loss: 0.3444 - accuracy: 0.8812 - val_loss: 0.3186 - val_accuracy: 0.905
Epoch 9/50
416/416 [==============================] - 17s 40ms/step - loss: 0.2808 - accuracy: 0.9060 - val_loss: 0.2848 - val_accuracy: 0.911
Epoch 10/50
416/416 [==============================] - 16s 40ms/step - loss: 0.2587 - accuracy: 0.9100 - val_loss: 0.3052 - val_accuracy: 0.902
Epoch 11/50
416/416 [==============================] - 16s 39ms/step - loss: 0.2280 - accuracy: 0.9221 - val_loss: 0.3356 - val_accuracy: 0.900
Epoch 12/50
416/416 [==============================] - 16s 39ms/step - loss: 0.2326 - accuracy: 0.9204 - val_loss: 0.3329 - val_accuracy: 0.910
Epoch 13/50
416/416 [==============================] - 17s 40ms/step - loss: 0.1563 - accuracy: 0.9457 - val_loss: 0.2522 - val_accuracy: 0.921
Epoch 14/50
416/416 [==============================] - 16s 39ms/step - loss: 0.1381 - accuracy: 0.9529 - val_loss: 0.2553 - val_accuracy: 0.927
Epoch 15/50
416/416 [==============================] - 16s 39ms/step - loss: 0.1250 - accuracy: 0.9560 - val_loss: 0.2795 - val_accuracy: 0.921
Epoch 16/50
416/416 [==============================] - 16s 39ms/step - loss: 0.1162 - accuracy: 0.9596 - val_loss: 0.2720 - val_accuracy: 0.920
Epoch 17/50
416/416 [==============================] - 16s 39ms/step - loss: 0.0991 - accuracy: 0.9652 - val_loss: 0.2702 - val_accuracy: 0.927
Epoch 18/50
416/416 [==============================] - 16s 39ms/step - loss: 0.0974 - accuracy: 0.9656 - val_loss: 0.2717 - val_accuracy: 0.927
```

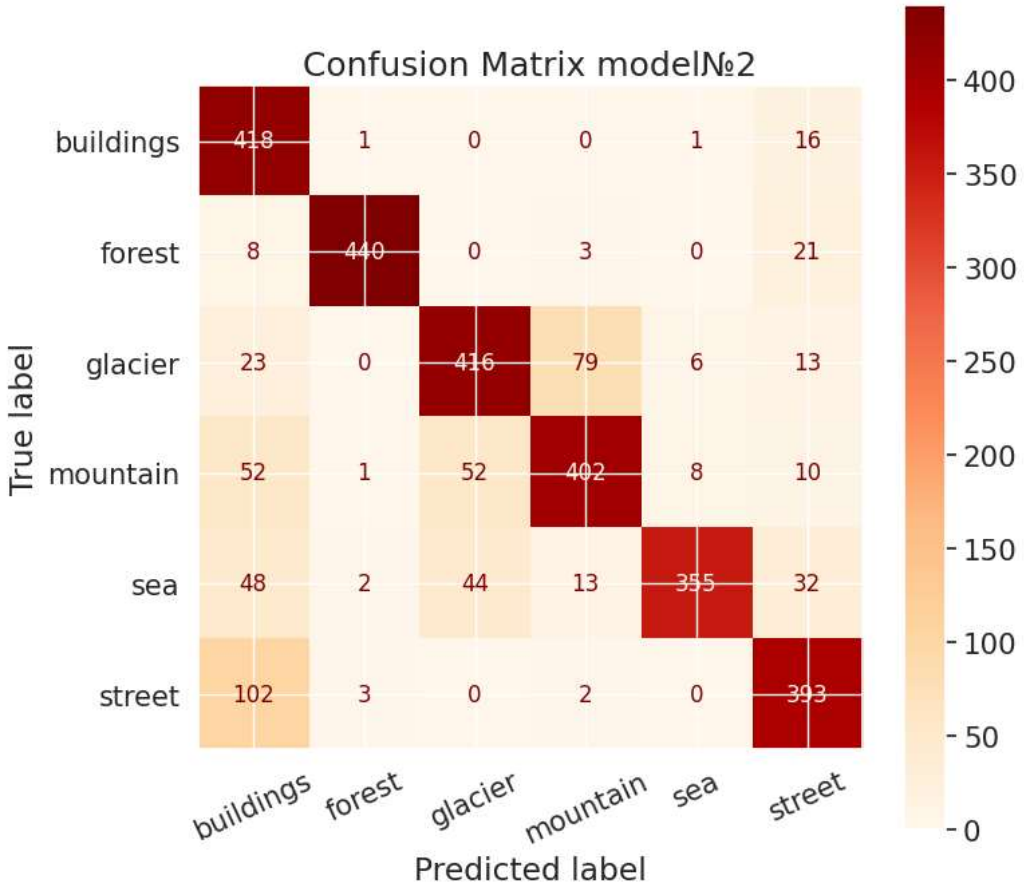
```
plot_accuracy_loss(history2)
```




```
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.
cm2 = confusion_matrix(test_labels, pred_labels)
print(cm2)
print('\n')
print(classification_report(test_labels, pred_labels))
```

```
93/93 [==============================] - 3s 29ms/step
[[418   1   0   0   1  16]
 [  8 440   0   3   0  21]
 [ 23   0 416  79   6  13]
 [ 52   1  52 402   8  10]
 [ 48   2  44  13 355  32]
 [102   3   0   2   0 393]]


              precision    recall  f1-score   support

           0       0.64      0.96      0.77       436
           1       0.98      0.93      0.96       472
           2       0.81      0.77      0.79       537
           3       0.81      0.77      0.79       525
           4       0.96      0.72      0.82       494
           5       0.81      0.79      0.80       500

    accuracy                           0.82      2964
   macro avg       0.84      0.82      0.82      2964
weighted avg       0.84      0.82      0.82      2964
```

```
plt.rcParams['figure.figsize'] = (8, 8)
plt.rcParams['font.size'] = 12
display_c_m = ConfusionMatrixDisplay(cm2, display_labels=class_names)
display_c_m.plot(cmap='OrRd', xticks_rotation=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Confusion Matrix model№2', fontsize=18)
plt.savefig('confusion_matrix_model2.png', transparent=True, dpi=300)
plt.show()
```



```
#now lets compile together two different models (model1 and model2)
model_cnn=''
model2=''
cm=''
cm2=''
history=''
history2=''
y_pred=''
```

```
 !nvidia-smi -L
```

```
    GPU 0: Tesla V100-SXM2-16GB (UUID: GPU-2e18bb6f-54f2-7504-7e01-985f1080481b)
```

```
!free -h
```

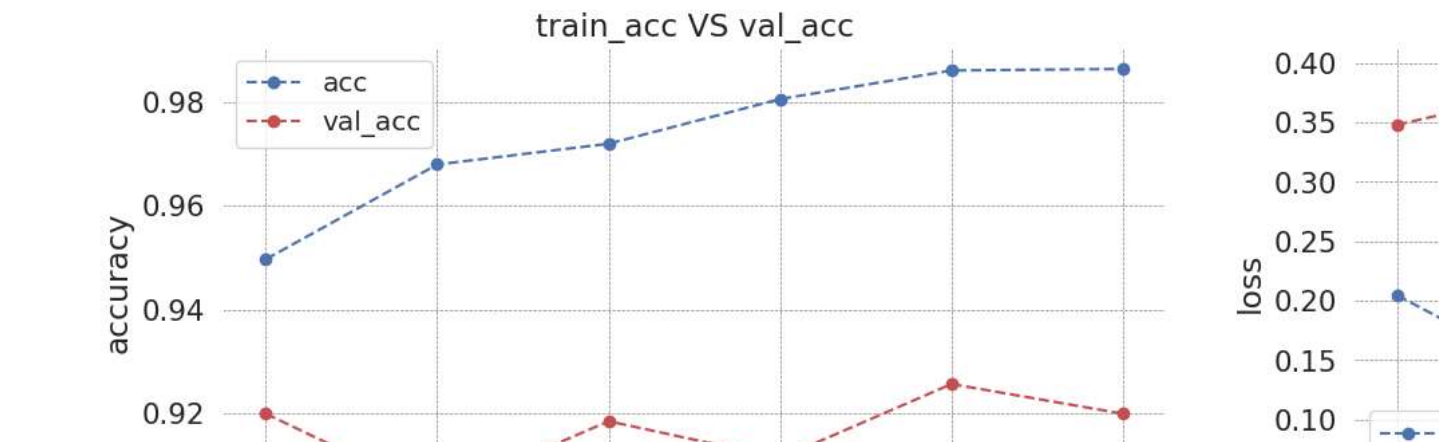
```
                total        used        free      shared  buff/cache   available
    Mem:           25Gi        22Gi       567Mi        34Mi       2.5Gi       2.6Gi
    Swap:            0B          0B          0B
```

```
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 [==============================] - 39s 530ms/step - loss: 0.2045 - accuracy: 0.9497 - val_loss: 0.3481 - val_accuracy: 0.9200
    Epoch 2/50
    52/52 [==============================] - 17s 333ms/step - loss: 0.1289 - accuracy: 0.9679 - val_loss: 0.3811 - val_accuracy: 0.9043
    Epoch 3/50
    52/52 [==============================] - 17s 334ms/step - loss: 0.0964 - accuracy: 0.9719 - val_loss: 0.3764 - val_accuracy: 0.9186
    Epoch 4/50
    52/52 [==============================] - 17s 334ms/step - loss: 0.0643 - accuracy: 0.9805 - val_loss: 0.3861 - val_accuracy: 0.9114
    Epoch 5/50
    52/52 [==============================] - 17s 334ms/step - loss: 0.0424 - accuracy: 0.9860 - val_loss: 0.3951 - val_accuracy: 0.9257
    Epoch 6/50
    52/52 [==============================] - 17s 335ms/step - loss: 0.0362 - accuracy: 0.9863 - val_loss: 0.3879 - val_accuracy: 0.9200
```

```
plot_accuracy_loss(history)
```

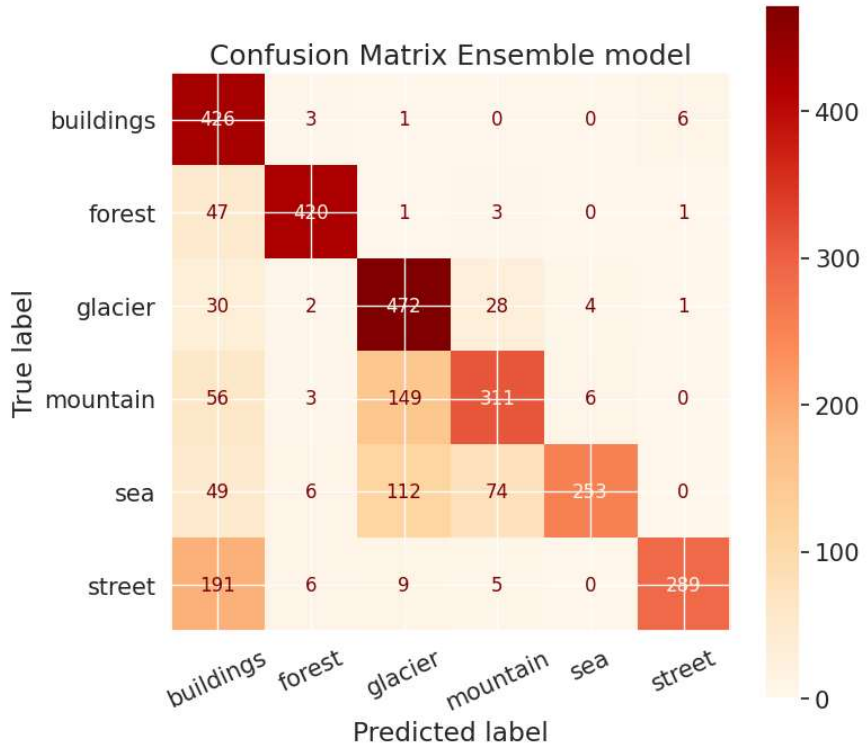


```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
pred_images = ensemble_model.predict(test_images) #this is vector of probabilities
pred_labels = np.argmax(pred_images, axis=1) #take the highest one prob.
ensemble_cm = confusion_matrix(test_labels, pred_labels)
print(ensemble_cm)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

```
93/93 [==============================] - 2s 23ms/step
[[426   3   1   0   0   6]
 [ 47 420   1   3   0   1]
 [ 30   2 472  28   4   1]
 [ 56   3 149 311   6   0]
 [ 49   6 112  74 253   0]
 [191   6   9   5   0 289]]


              precision    recall  f1-score   support

           0       0.53      0.98      0.69       436
           1       0.95      0.89      0.92       472
           2       0.63      0.88      0.74       537
           3       0.74      0.59      0.66       525
           4       0.96      0.51      0.67       494
           5       0.97      0.58      0.73       500

    accuracy                           0.73      2964
   macro avg       0.80      0.74      0.73      2964
weighted avg       0.80      0.73      0.73      2964
```

```
ensemble_model.save(work_path + '/model3.h5')
```

```
plt.rcParams['figure.figsize'] = (8, 8)
plt.rcParams['font.size'] = 12
display_c_m = ConfusionMatrixDisplay(ensemble_cm, display_labels=class_names)
display_c_m.plot(cmap='OrRd', xticks_rotation=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Confusion Matrix Ensemble model', fontsize=18)
plt.savefig('confusion_matrix_ensemble_model.png', transparent=True, dpi=300)
plt.show()
```



✓ 1s completed at 8:03 AM