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



#!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
print('Data loaded...')
```

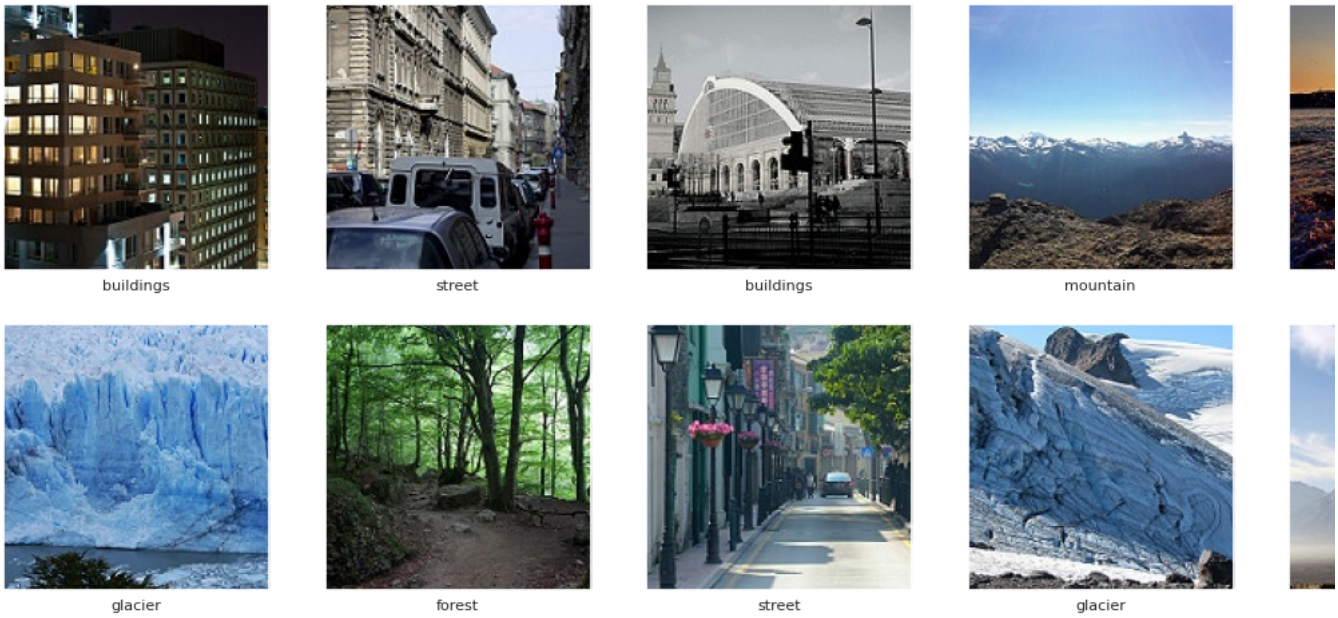
```
Data loaded...
```

```
(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, 2)
```

```
(700, 150, 150, 3)
(700, 6, 2)
(2964, 150, 150, 3)
(2964, 6, 2)
```

```
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


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)
```

```
________
<keras_tuner.engine.hyperparameters.hyperparameters.HyperParameters object at 0x7f1f121d6980>
```

```
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}
```

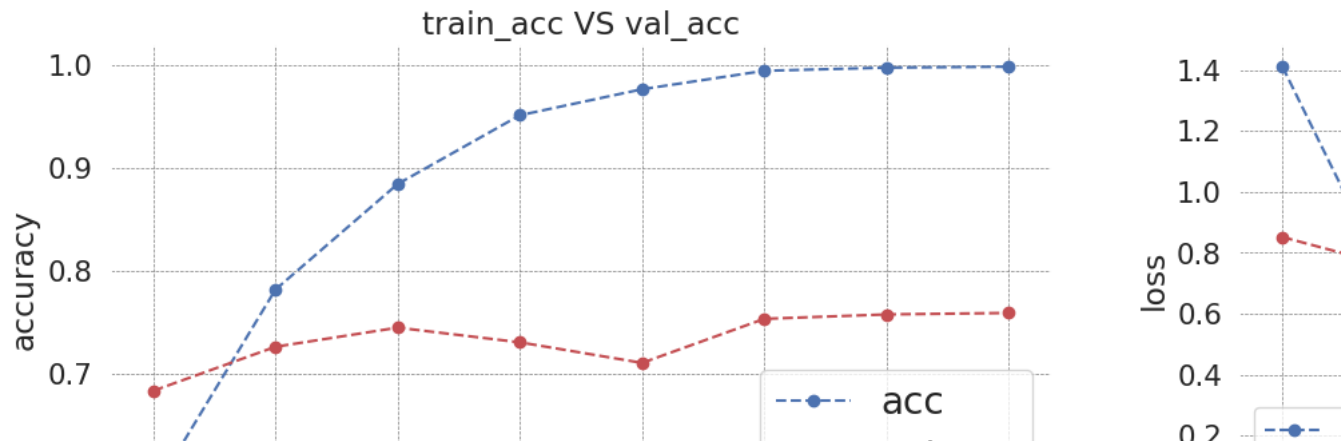
```
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
import h5py


erl_stop = EarlyStopping(monitor='val_loss', patience = 6, restore_best_weights=True)
mod_chk = ModelCheckpoint(filepath='my_model.hdf5', monitor='val_loss', save_best_only=True)
lr_rate = ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.1)


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_chk, lr_r
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 150, 150, 32)      896

 max_pooling2d_12 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 dropout_12 (Dropout)        (None, 75, 75, 32)        0

 conv2d_13 (Conv2D)          (None, 73, 73, 32)        9248

 max_pooling2d_13 (MaxPoolin  (None, 36, 36, 32)       0
 g2D)

 dropout_13 (Dropout)        (None, 36, 36, 32)        0

 flatten_6 (Flatten)         (None, 41472)             0

 dense_12 (Dense)            (None, 256)               10617088

 dense_13 (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: 1.4117 - accuracy: 0.5846 - val_loss: 0.8519 - val_accuracy: 0.6829
Epoch 2/50
416/416 [==============================] - 4s 8ms/step - loss: 0.5950 - accuracy: 0.7813 - val_loss: 0.7443 - val_accuracy: 0.7257
Epoch 3/50
416/416 [==============================] - 3s 8ms/step - loss: 0.3353 - accuracy: 0.8840 - val_loss: 0.8726 - val_accuracy: 0.7443
Epoch 4/50
416/416 [==============================] - 3s 8ms/step - loss: 0.1604 - accuracy: 0.9508 - val_loss: 0.9716 - val_accuracy: 0.7300
Epoch 5/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0887 - accuracy: 0.9761 - val_loss: 1.2005 - val_accuracy: 0.7100
Epoch 6/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0333 - accuracy: 0.9941 - val_loss: 1.0811 - val_accuracy: 0.7529
Epoch 7/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0199 - accuracy: 0.9971 - val_loss: 1.1132 - val_accuracy: 0.7571
Epoch 8/50
416/416 [==============================] - 3s 8ms/step - loss: 0.0147 - accuracy: 0.9980 - val_loss: 1.1339 - val_accuracy: 0.7586
```

```
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 + '/model/')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 2 of
```

```
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
```

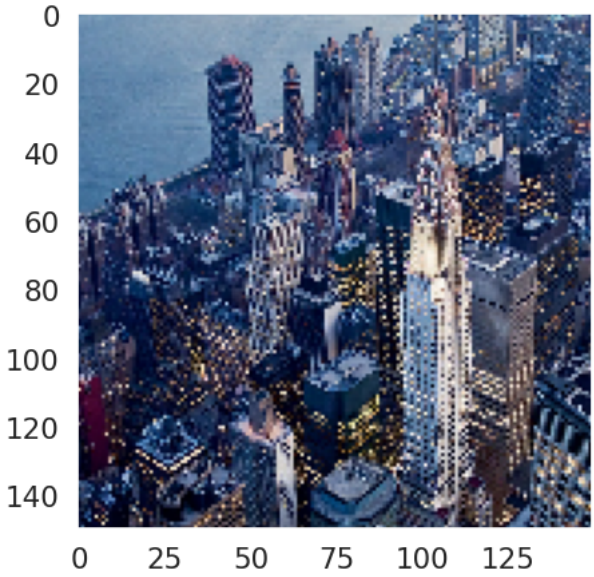
```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
pred_images = model_cnn.predict(test_images) #this is vector of probabilities
pred_labels = np.argmax(pred_images, axis=1) #take the highest one prob.
print(classification_report(test_labels, pred_labels))
```

```
93/93 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

           0       0.78      0.50      0.61       436
           1       0.98      0.77      0.86       472
           2       0.61      0.81      0.69       537
           3       0.74      0.54      0.62       525
           4       0.56      0.60      0.58       494
           5       0.63      0.87      0.73       500

    accuracy                           0.68      2964
   macro avg       0.71      0.68      0.68      2964
weighted avg       0.71      0.68      0.68      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 34ms/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
```



✓ 1s completed at 7:14 PM