```
#!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
from sklearn.metrics import classification_report
import seaborn as sn; sn.set(font_scale=1.4)
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tqdm import tqdm
import os
import random

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
```

```
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...')
```

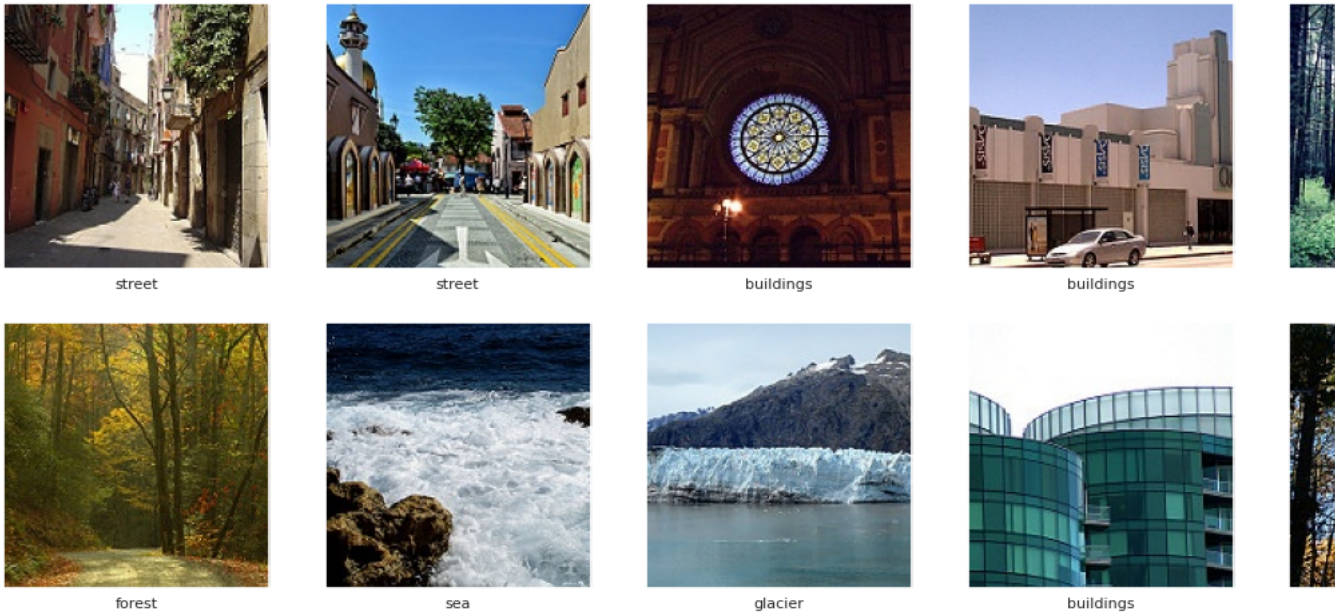
```
Data loaded...
```

```
(train_images, train_labels), (test_images, test_labels) = load_data()
```

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

```
13981
```

```
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)
```



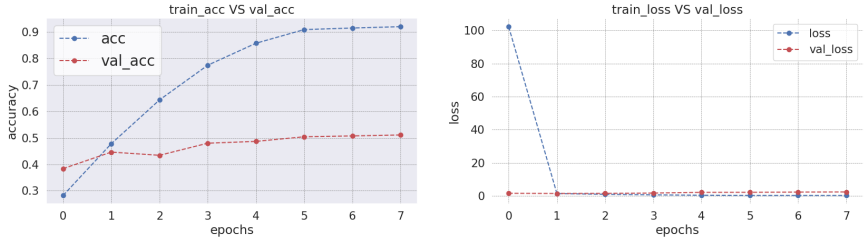

```
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
import h5py


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 = Sequential([
     tf.keras.layers.Conv2D(32,(3,3), activation='relu', input_shape=(150,150,3)),
     tf.keras.layers.MaxPooling2D(2,2),
     tf.keras.layers.Conv2D(32,(3,3), activation='relu'),
     tf.keras.layers.MaxPooling2D(2,2),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128,activation=tf.nn.relu),
     tf.keras.layers.Dense(6,activation=tf.nn.softmax)
     ])


#model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])


#model.fit(train_images, train_labels, batch_size=64,epochs=4)
history = model.fit(train_images, train_labels, batch_size=128,epochs=100,validation_split=0.2, callbacks=[erl_stop, mod_chk, lr_rate])
```

```
     Epoch 1/100
     88/88 [==============================] - 13s 37ms/step - loss: 102.3049 - accuracy: 0.2821 - val_loss: 1.5410 - val_accuracy: 0.383
     Epoch 2/100
     88/88 [==============================] - 2s 22ms/step - loss: 1.3714 - accuracy: 0.4779 - val_loss: 1.4241 - val_accuracy: 0.4462 -
     Epoch 3/100
     88/88 [==============================] - 2s 20ms/step - loss: 0.9428 - accuracy: 0.6434 - val_loss: 1.4883 - val_accuracy: 0.4340 -
     Epoch 4/100
     88/88 [==============================] - 2s 21ms/step - loss: 0.6010 - accuracy: 0.7746 - val_loss: 1.6884 - val_accuracy: 0.4798 -
     Epoch 5/100
     88/88 [==============================] - 2s 21ms/step - loss: 0.3814 - accuracy: 0.8577 - val_loss: 2.1086 - val_accuracy: 0.4866 -
     Epoch 6/100
     88/88 [==============================] - 2s 20ms/step - loss: 0.2518 - accuracy: 0.9092 - val_loss: 2.1411 - val_accuracy: 0.5041 -
     Epoch 7/100
     88/88 [==============================] - 2s 20ms/step - loss: 0.2294 - accuracy: 0.9151 - val_loss: 2.2834 - val_accuracy: 0.5073 -
```

```
Epoch 8/100
88/88 [==============================] - 2s 20ms/step - loss: 0.2153 - accuracy: 0.9202 - val_loss: 2.3596 - val_accuracy: 0.5109 -
```

```
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)
```




```
model.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_images)) + " of images in " + str(len(class_names_label)) + " classes")
print("And " + str(len(test_images)) + " test images")
```

```
We have 13981 of images in 6 classes
And 2964 test images
```

```
pred_images = 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 4ms/step
              precision    recall  f1-score   support

           0       0.32      0.29      0.31       436
           1       0.55      0.77      0.64       472
           2       0.39      0.55      0.45       537
           3       0.37      0.33      0.35       525
           4       0.34      0.13      0.19       494
           5       0.49      0.47      0.48       500

    accuracy                           0.42      2964
   macro avg       0.41      0.42      0.40      2964
weighted avg       0.41      0.42      0.40      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.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 157ms/step
Results of image classification: [[0.23211876 0.1923747  0.00534715 0.013098   0.0164776  0.5405838 ]]
Rounded predict values: [[0.23 0.19 0.01 0.01 0.02 0.54]]
Our image belongs to class: street
```