بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Imports

```
!pip install ipython-autotime
```

```
Requirement already satisfied: ipython-autotime in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: ipython in /usr/local/lib/python3.6/dist-packages (from i
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/lib/python
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.6/
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from trait
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages (from p
```

```
# necessary imports
import os
import cv2
import numpy as np
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
from tqdm import tqdm

import matplotlib.pyplot as plt
%matplotlib inline

from google.colab.patches import cv2_imshow

%load_ext autotime
```

```
time: 120 µs (started: 2021-01-07 06:35:34 +00:00)
```

```
img_width = 150
img_height = 150
```

```
time: 724 µs (started: 2021-01-07 06:35:34 +00:00)
```

InceptionResNetV2 Model

```
from keras.applications import InceptionResNetV2
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
```

time: 1.32 s (started: 2021-01-07 06:35:35 +00:00)

```
model = InceptionResNetV2(include_top=False, weights='imagenet', input_shape=(img_width, img_
```

time: 5.13 s (started: 2021-01-07 06:35:36 +00:00)

```
# add new classification layers
flat1 = Flatten()(model.layers[-1].output) # flatten last layer
class1 = Dense(1024, activation='relu')(flat1) # add FC layer on previous layer
class2 = Dense(1024, activation='relu')(class1) # add FC layer on previous layer
output = Dense(6, activation='softmax')(class2) # add softmax layer
```

time: 27.1 ms (started: 2021-01-07 06:35:41 +00:00)

```
# define the new model
model = Model(inputs=model.inputs, outputs=output)
model.summary()
```

```
block8_9_conv (Conv2D)          (None, 3, 3, 2080)   995920      block8_9_mixed[0][0]
__________________________________________________________________________________________
block8_9 (Lambda)               (None, 3, 3, 2080)   0           block8_8_ac[0][0]
                                                                 block8_9_conv[0][0]
__________________________________________________________________________________________
block8_9_ac (Activation)        (None, 3, 3, 2080)   0           block8_9[0][0]
__________________________________________________________________________________________
conv2d_200 (Conv2D)             (None, 3, 3, 192)    399360      block8_9_ac[0][0]
__________________________________________________________________________________________
batch_normalization_200 (BatchN (None, 3, 3, 192)    576         conv2d_200[0][0]
__________________________________________________________________________________________
activation_200 (Activation)     (None, 3, 3, 192)    0           batch_normalization_
__________________________________________________________________________________________
conv2d_201 (Conv2D)             (None, 3, 3, 224)    129024      activation_200[0][0]
__________________________________________________________________________________________
batch_normalization_201 (BatchN (None, 3, 3, 224)    672         conv2d_201[0][0]
__________________________________________________________________________________________
activation_201 (Activation)     (None, 3, 3, 224)    0           batch_normalization_
__________________________________________________________________________________________
conv2d_199 (Conv2D)             (None, 3, 3, 192)    399360      block8_9_ac[0][0]
__________________________________________________________________________________________
conv2d_202 (Conv2D)             (None, 3, 3, 256)    172032      activation_201[0][0]
__________________________________________________________________________________________
batch_normalization_199 (BatchN (None, 3, 3, 192)    576         conv2d_199[0][0]
__________________________________________________________________________________________
```

```
batch_normalization_202 (BatchN (None, 3, 3, 256)    768         conv2d_202[0][0]
__________________________________________________________________________________________
activation_199 (Activation)     (None, 3, 3, 192)    0           batch_normalization_
__________________________________________________________________________________________
activation_202 (Activation)     (None, 3, 3, 256)    0           batch_normalization_
__________________________________________________________________________________________
block8_10_mixed (Concatenate)   (None, 3, 3, 448)    0           activation_199[0][0]
                                                                 activation_202[0][0]
__________________________________________________________________________________________
block8_10_conv (Conv2D)         (None, 3, 3, 2080)   933920      block8_10_mixed[0][0
__________________________________________________________________________________________
block8_10 (Lambda)              (None, 3, 3, 2080)   0           block8_9_ac[0][0]
                                                                 block8_10_conv[0][0]
__________________________________________________________________________________________
conv_7b (Conv2D)                (None, 3, 3, 1536)   3194880     block8_10[0][0]
__________________________________________________________________________________________
conv_7b_bn (BatchNormalization) (None, 3, 3, 1536)   4608        conv_7b[0][0]
__________________________________________________________________________________________
conv_7b_ac (Activation)         (None, 3, 3, 1536)   0           conv_7b_bn[0][0]
__________________________________________________________________________________________
flatten (Flatten)               (None, 13824)        0           conv_7b_ac[0][0]
__________________________________________________________________________________________
dense (Dense)                   (None, 1024)         14156800    flatten[0][0]
__________________________________________________________________________________________
dense_1 (Dense)                 (None, 1024)         1049600     dense[0][0]
__________________________________________________________________________________________
dense_2 (Dense)                 (None, 6)            6150        dense_1[0][0]
==========================================================================================
Total params: 69,549,286
Trainable params: 69,488,742
Non-trainable params: 60,544
__________________________________________________________________________________________
time: 275 ms (started: 2021-01-07 06:35:41 +00:00)
```

Loading Data

```
# !unzip "/content/drive/MyDrive/CV/Assignment 3/intel-image-classification.zip" -d "/content
```

```
time: 680 µs (started: 2021-01-07 06:35:41 +00:00)
```

```
# !unzip "/content/drive/MyDrive/CV/Assignment 3/Test_data.zip" -d "/content/drive/MyDrive/CV
```

```
time: 436 µs (started: 2021-01-07 06:35:41 +00:00)
```

```
# A function to load data from a given directory
def load_data(data_dir):
  data = []
  labels = []
  class_dirs = os.listdir(data_dir)
```

```
  for direc in class_dirs:
    # i=0
    class_dir = os.path.join(data_dir, direc)
    for imagepath in tqdm(list(paths.list_images(class_dir))):
      image = cv2.imread(imagepath)
      image = cv2.resize(image, (img_width, img_height))  # incase images not of same size
      data.append(image)
      labels.append(direc)
      # i = i+1
      # if (i==10):
      #   break
  # normalizing and converting to numpy array format
  data = np.array(data, dtype='float')/255.0
  labels = np.array(labels)
  return data, labels
```

time: 9.32 ms (started: 2021-01-07 06:35:41 +00:00)

```
train_dir = "/content/drive/MyDrive/CV/Assignment 3/seg_train/seg_train/"
test_dir = "/content/drive/MyDrive/CV/Assignment 3/seg_test/seg_test/"
pred_dir = "/content/drive/MyDrive/CV/Assignment 3/pred/seg_pred/seg_pred/"
```

time: 770 µs (started: 2021-01-07 06:35:41 +00:00)

Compile the model

```
import tensorflow as tf
import keras
from keras.optimizers import RMSprop
```

time: 718 µs (started: 2021-01-07 06:35:41 +00:00)

```
initial_learning_rate = 0.1
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=7000,
    decay_rate=0.96,
    staircase=True)

opt = keras.optimizers.RMSprop(learning_rate=lr_schedule)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

time: 38 ms (started: 2021-01-07 06:35:41 +00:00)

```
# from keras.optimizers import SGD
# sgd = SGD(lr=0.001, decay=1e-7, momentum=.9)
# model.compile(loss='categorical_crossentropy',
#               optimizer=sgd,
```

```
#               metrics=['accuracy'])
```

```
     time: 634 µs (started: 2021-01-07 06:35:41 +00:00)
```

```
from keras.preprocessing.image import ImageDataGenerator
```

```
     time: 7.17 ms (started: 2021-01-07 06:35:41 +00:00)
```

```
img_height = 150
img_width = 150
batch_size = 16
nb_epochs = 10
```

```
     time: 715 µs (started: 2021-01-07 06:35:41 +00:00)
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2, # zoom
    rotation_range=10, # rotation
    width_shift_range=0.2, # horizontal shift
    height_shift_range=0.2, # vertical shift
    horizontal_flip=True) # horizontal flip
    # channel_shift_range = [-0.1, 0.1]
    # )
    # ,validation_split=0.3) # set validation split
```

```
     time: 1.78 ms (started: 2021-01-07 06:35:41 +00:00)
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=True,
    class_mode='categorical',
    interpolation="nearest")
    # subset='training') # set as training data
```

```
     Found 14034 images belonging to 6 classes.
     time: 343 ms (started: 2021-01-07 06:35:41 +00:00)
```

```
val_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
     time: 2.43 ms (started: 2021-01-07 06:35:42 +00:00)
```

```
validation_generator = val_datagen.flow_from_directory(
    test_dir, # directory for validation data
    target_size=(img_height, img_width),
    batch_size=batch_size,
```

```
    class_mode='categorical')
    # subset='validation') # set as validation data
```

```
Found 3000 images belonging to 6 classes.
time: 113 ms (started: 2021-01-07 06:35:42 +00:00)
```

```
# validation_generator[0]
```

```
time: 1.3 ms (started: 2021-01-07 06:35:42 +00:00)
```

```
H = model.fit(
    train_generator,
    steps_per_epoch = train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs = nb_epochs)
```

```
Epoch 1/10
877/877 [==============================] - 3905s 4s/step - loss: 22897.4580 - accuracy:
Epoch 2/10
877/877 [==============================] - 164s 187ms/step - loss: 1.8664 - accuracy: 0
Epoch 3/10
877/877 [==============================] - 166s 189ms/step - loss: 1.8042 - accuracy: 0
Epoch 4/10
877/877 [==============================] - 166s 189ms/step - loss: 1.8070 - accuracy: 0
Epoch 5/10
877/877 [==============================] - 167s 190ms/step - loss: 1.8064 - accuracy: 0
Epoch 6/10
877/877 [==============================] - 166s 189ms/step - loss: 1.8063 - accuracy: 0
Epoch 7/10
877/877 [==============================] - 166s 190ms/step - loss: 1.8057 - accuracy: 0
Epoch 8/10
877/877 [==============================] - 166s 189ms/step - loss: 1.8044 - accuracy: 0
Epoch 9/10
877/877 [==============================] - 165s 188ms/step - loss: 1.8051 - accuracy: 0
Epoch 10/10
877/877 [==============================] - 166s 189ms/step - loss: 1.8051 - accuracy: 0
time: 1h 29min 58s (started: 2021-01-07 06:35:42 +00:00)
```

```
save_path = '/content/drive/MyDrive/CV/Assignment 3/InceptionResNetV2_Aug_Adam'
```

```
time: 1.19 ms (started: 2021-01-07 08:05:40 +00:00)
```

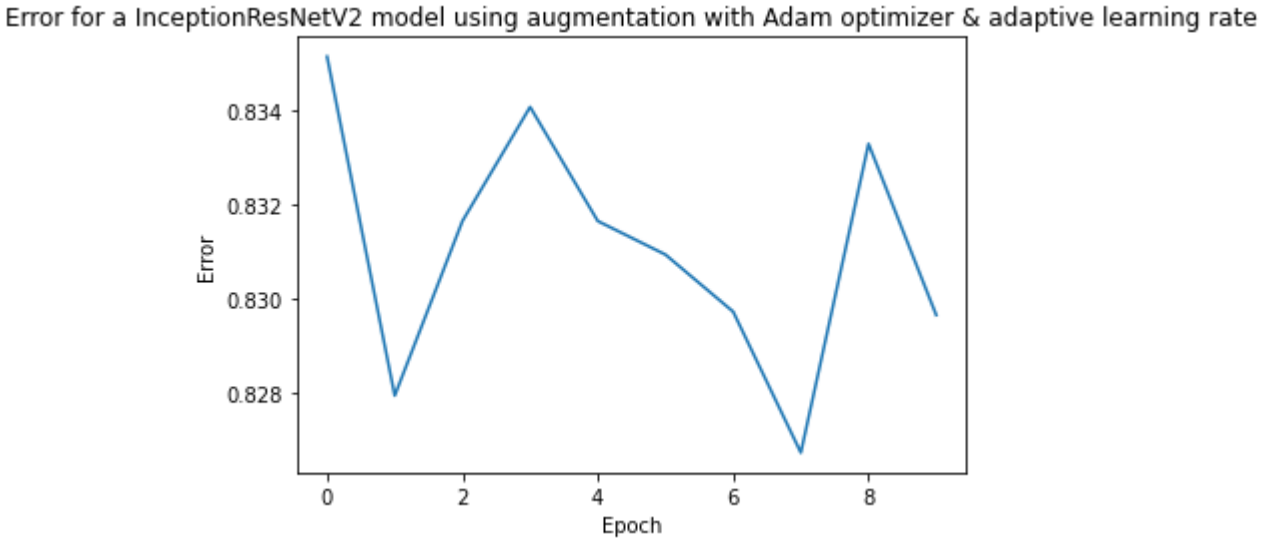
```
# save the model's trained weights
model.save_weights(save_path+"transfer_trained_wts.h5")
```

```
time: 1.95 s (started: 2021-01-07 08:14:26 +00:00)
```

```
# model.load_weights('/content/drive/MyDrive/CV/Assignment 3/vgg_aug_transfer_trained_wts.h5'
```

```
time: 680 µs (started: 2021-01-07 08:14:29 +00:00)
```

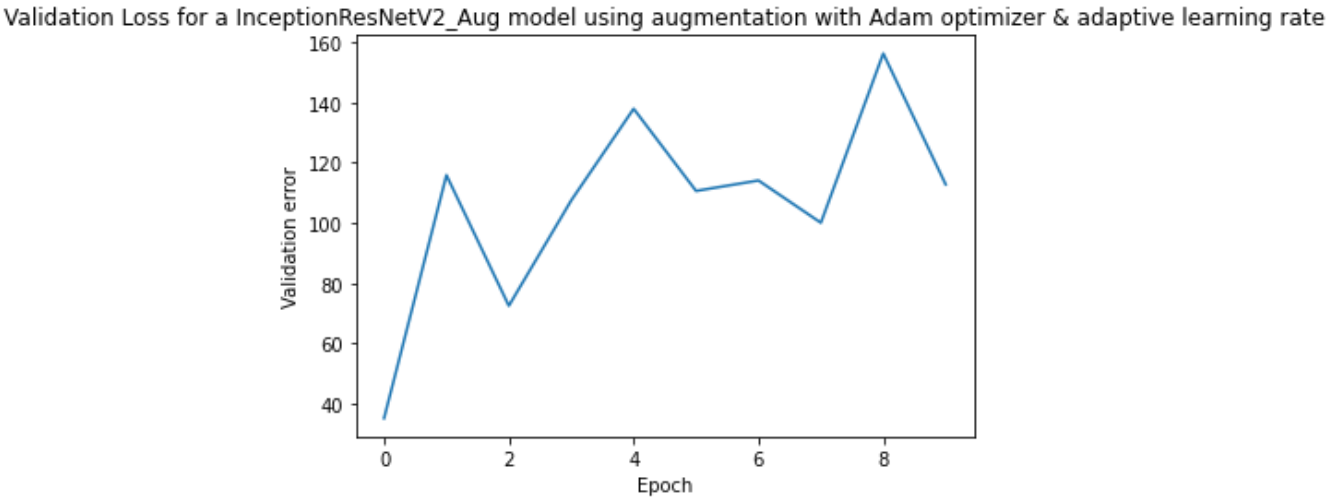
```
simple_acc = H.history['accuracy']
plt.plot([1 - acc for acc in simple_acc])
plt.title('Error for a InceptionResNetV2 model using augmentation with Adam optimizer & adapt
plt.ylabel('Error')
plt.xlabel('Epoch')
plt.savefig(save_path+'/simple_acc_error.png')
plt.show()
```




```
time: 184 ms (started: 2021-01-07 08:15:20 +00:00)
```

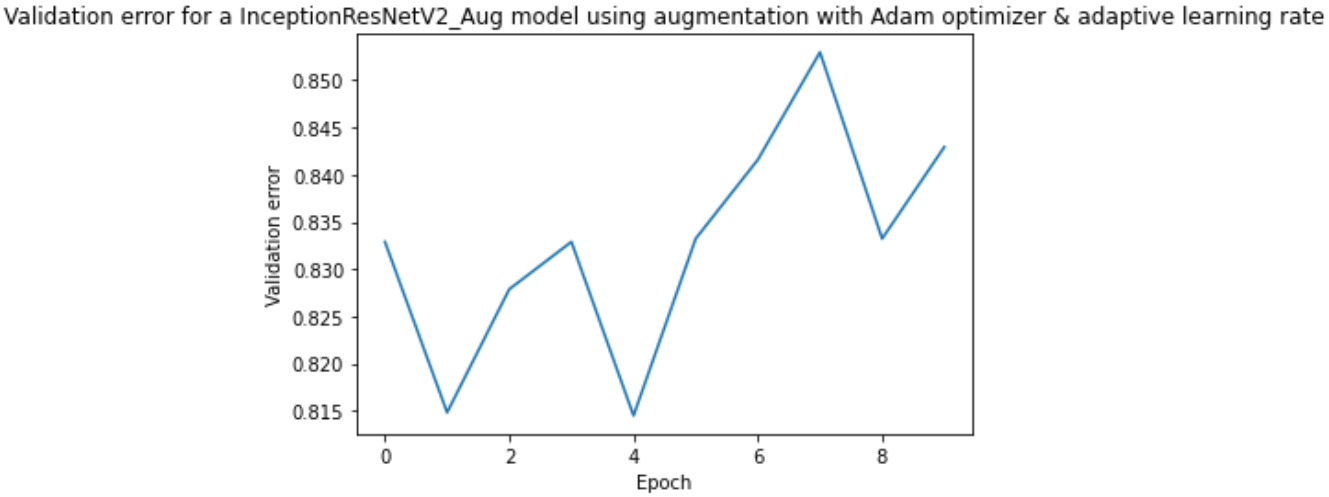
```
simple_loss = H.history['loss']
plt.plot([los for los in simple_loss])
plt.title('Loss for a InceptionResNetV2_Aug model using augmentation with Adam optimizer & ad
plt.ylabel('Error')
plt.xlabel('Epoch')
plt.savefig(save_path+'/simple_loss_error.png')
plt.show()
```

```
simple_val_loss = H.history['val_loss']
plt.plot([los for los in simple_val_loss])
plt.title('Validation Loss for a InceptionResNetV2_Aug model using augmentation with Adam opt
plt.ylabel('Validation error')
plt.xlabel('Epoch')
plt.savefig(save_path+'/simple_Validation_loss_error.png')
plt.show()
```




```
time: 198 ms (started: 2021-01-07 08:15:24 +00:00)
```

```
simple_val_acc = H.history['val_accuracy']
plt.plot([1 - acc for acc in simple_val_acc])
plt.title('Validation error for a InceptionResNetV2_Aug model using augmentation with Adam op
plt.ylabel('Validation error')
plt.xlabel('Epoch')
plt.savefig(save_path+'/simple_Validation_error.png')
plt.show()
```




```
time: 196 ms (started: 2021-01-07 08:15:25 +00:00)
```

```
print('loading pred images')
X_test, y_test = load_data(pred_dir)
```

```
loading pred images
100%|██████████| 1128/1128 [08:34<00:00,  2.19it/s]
100%|██████████| 1297/1297 [10:04<00:00,  2.15it/s]
100%|██████████| 1330/1330 [10:14<00:00,  2.17it/s]
100%|██████████| 1166/1166 [09:06<00:00,  2.13it/s]
100%|██████████| 1144/1144 [09:30<00:00,  2.01it/s]
100%|██████████| 1236/1236 [09:43<00:00,  2.12it/s]
time: 57min 23s (started: 2021-01-07 08:15:25 +00:00)
```

```
lb = LabelBinarizer()
y_test = lb.fit_transform(y_test)
```

```
time: 11.8 ms (started: 2021-01-07 09:12:49 +00:00)
```

```
score = model.evaluate(X_test, y_test, batch_size=64)
print('Test Loss = ', score[0])
print('Test Accuracy = ', score[1])
```

```
115/115 [==============================] - 20s 144ms/step - loss: 368.8753 - accuracy: 0
Test Loss =  368.8753356933594
Test Accuracy =  0.1606629192829132
time: 22.1 s (started: 2021-01-07 09:12:49 +00:00)
```

```
'''CONFUSION MATRIX'''
# Making prediction
y_pred = model.predict(X_test)
y_true = np.argmax(y_test, axis=-1)

# Plotting the confusion matrix
from sklearn.metrics import confusion_matrix
confusion_mtx = confusion_matrix(y_true, np.argmax(y_pred, axis=1))
```

```
time: 21.8 s (started: 2021-01-07 09:13:11 +00:00)
```

```
mask = y_pred==y_test
correct = np.count_nonzero(mask)
print (correct*100.0/y_pred.size)
result = y_pred.astype(int)
```

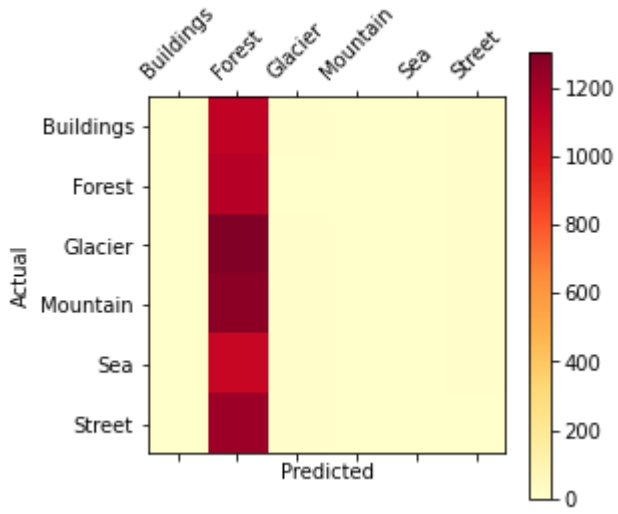
```
1.1185682326621924
time: 5.16 ms (started: 2021-01-07 09:13:33 +00:00)
```

```
confusion_mtx
```

```
array([[   0, 1125,    8,    0,    0,   11],
       [   0, 1157,    3,    0,    0,    6],
```

```
       [   0, 1305,   12,    0,    0,   13],
       [   0, 1267,   15,    0,    0,   15],
       [   0, 1101,   14,    0,    0,   13],
       [   0, 1225,    7,    0,    0,    4]])time: 2.58 ms (started: 2021-01-07 09:13:33
```

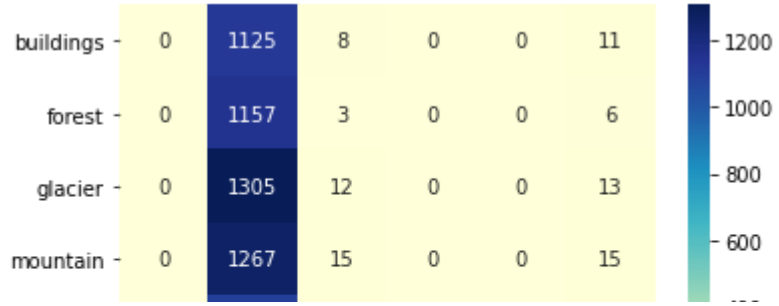
```
def plot_confusion_matrix(df_confusion, title='Confusion matrix', cmap=plt.cm.YlOrRd):
  plt.matshow(df_confusion, cmap=cmap) # imshow
  plt.colorbar()
  tick_marks = np.arange(6)
  names = ["Buildings", "Forest", "Glacier", "Mountain", "Sea", "Street"]
  plt.xticks(tick_marks, names, rotation=45)
  plt.yticks(tick_marks, names)
  plt.ylabel("Actual")
  plt.xlabel("Predicted")
#call function
plot_confusion_matrix(confusion_mtx)
```




```
time: 205 ms (started: 2021-01-07 09:13:33 +00:00)
```

```
import seaborn as sns

class_names = ['buildings','street','forest','glacier','mountain','sea']

class_names = sorted(class_names)

sns.heatmap(confusion_mtx, xticklabels=class_names, yticklabels=class_names,

            annot=True, fmt='d', cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6978e74320>
```




```
fig, axis = plt.subplots(1, 2, figsize=(20, 4))


axis[0].plot(H.history['accuracy'],
         label='Train accuracy with augmentation',
         c='tomato', ls='-')
axis[0].plot(H.history['val_accuracy'],
         label='Validation accuracy with augmentation',
         c='magenta', ls='-')

axis[0].set_xlabel('Epoch')
axis[0].set_ylabel('Accuracy')
axis[0].legend(loc='upper left')


axis[1].plot(H.history['loss'],
         label='Train loss with augmentation',
         c='tomato', ls='-')
axis[1].plot(H.history['val_loss'],
         label='Validation loss with augmentation',
         c='magenta', ls='-')

axis[1].set_xlabel('Epoch')
axis[1].set_ylabel('loss')
axis[1].legend(loc='upper left')
plt.savefig(save_path+'/simple_Validation_error&loss.png')
plt.show()
```

```
  File "<ipython-input-42-5fcda59f6d64>", line 27
    plt.show()
      ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

```
def visualize_data(images, categories, class_names):
    fig = plt.figure(figsize=(14, 6))
    fig.patch.set_facecolor('white')
    for i in range(3 * 6):
        plt.subplot(3, 6, i+1)
        plt.xticks([])
```

```
            plt.yticks([])
            plt.imshow(images[i])
            class_index = categories[i].argmax()
            plt.xlabel(class_names[class_index])
        plt.show()


    visualize_data(X_test*255, y_test, names)
```