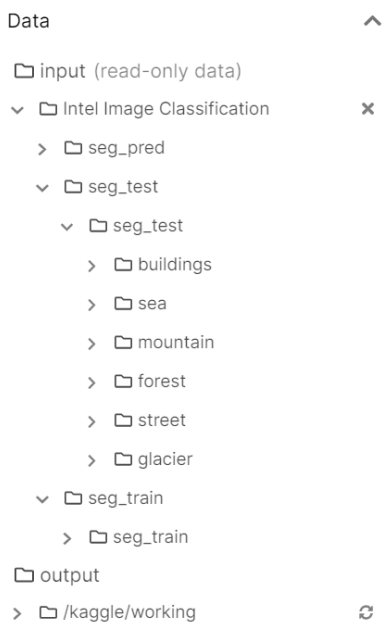
Images Classification

Load data



Từ bộ data có sẵn trên Kernel, ta import thư viện os để đọc dữ liệu từ datasets, cv2 để mở file ảnh cho vào các tập tương ứng

```
import numpy as np
import os
import cv2
```

```
class_names = ['buildings', 'sea', 'mountain', 'glacier', 'forest', 'street']
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}
IMAGE_SIZE = (150, 150)
```

- Load data từ dataset.
  1. Train_data lấy từ folder seg_train
  2. Test_data lất từ folder seg_test
  3. Val_data lấy 3000 hình ảnh ngẫu nhiên từ train_data
- Biến num lưu trữ dữ liệu về số ảnh tương ứng với mỗi class

```
def load_data():

    datasets = ['../input/intel-image-classification/seg_train/seg_train', '../input/intel-image-classifica
    output = []
    num = []

    for dataset in datasets:
        images = []
        labels = []

        for folder in os.listdir(dataset):
            curr_label = class_names_label[folder]

            for file in os.listdir(os.path.join(dataset, folder)):

                # Get the path name of the image
                img_path = os.path.join(os.path.join(dataset, folder), file)

                # Open and resize the img
                curr_img = cv2.imread(img_path)
                curr_img = cv2.resize(curr_img, IMAGE_SIZE)
```

```
                images.append(curr_img)
                labels.append(curr_label)
            if (len(num)) == 6:
                continue
            num.append([len(images), class_names[curr_label]])
        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')

        output.append((images, labels))

    images = []
    labels = []
    #lấy ngẫu nhiên 3000 hình trong 14000 hình của tập train là validation data
    for k in range(3000):
        i = np.random.randint(14000)
        images.append(output[0][0][i])
        labels.append(output[0][1][i])

    output.append((np.array(images, dtype='float32'), np.array(labels, dtype='int32')))
    return output, num
```

Function này trả về cho ta output là (images, labels) của tập train, tập test, tập validation và 1 list chứa thông tin về số lượng ảnh của mỗi class

* Có thể dùng np.permutation(len(…)) để trộn các classes vào nhau một cách ngẫu nhiên và validation_rate=… ở model.fit(…) để đánh giá model qua mỗi lần duyệt qua hết dữ liệu train

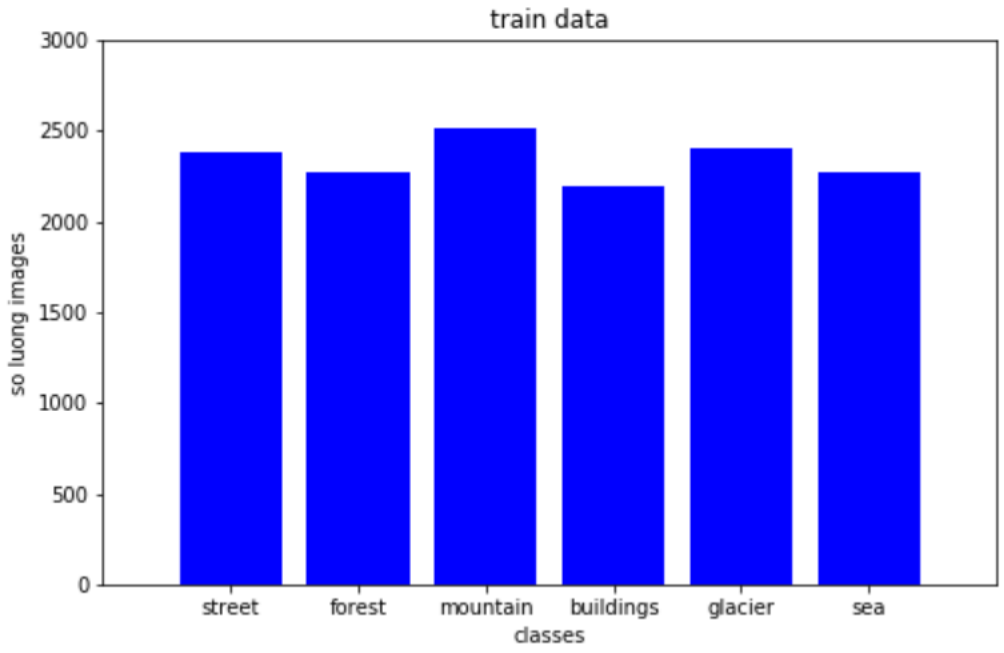
Build biểu đồ

```
import matplotlib.pyplot as plt
#xử lí dữ liệu để lấy số hình của mỗi class trong train_dataset
num[5][0] -= num[4][0]
num[4][0] -= num[3][0]
num[3][0] -= num[2][0]
num[2][0] -= num[1][0]
num[1][0] -= num[0][0]

classes = []
val = []

for i in num:
    classes.append(i[1])
    val.append(i[0])
#build biểu đồ
plt.subplots(figsize=(8,5))
plt.bar(classes, val, color='blue')
plt.axis([-1, 6, 0, 3000])
plt.ylabel('so luong images')
plt.xlabel('classes')
plt.title('train data')

plt.show()
```



Số lượng hình ảnh giữa các class tương đối đồng đều, không chênh lệch quá nhiều.

Giảm giá trị của dữ liệu xuống, dễ hàng hơn cho việc tính toán

```
train_images = train_images / 255.0
test_images = test_images / 255.0
val_images = val_images / 255.0
```

Build model

- Sử dụng convulsion filter 3×3 để lấy những đặc trưng của hình ảnh.
- Giảm kích thước của hình bằng MaxPooling có kích thước 2×2 (lấy giá trị lớn nhất trong phạm vi 2×2)
- Hàm kích hoạt của các hidden layer là ReUL (max(0, z)), trừ lớp cuối cùng dử dụng softmax để lấy được class theo yêu cầu của bài toán
- Sau khi làm phẳng (Flatten) ma trận, dùng fully conection layer (có dropout ở 1 layer)

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import optimizers
from keras.utils import np_utils

model = Sequential()
model.add(Conv2D(16, (3, 3), padding='valid', activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), padding='valid', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dense(6, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h = model.fit(train_images, train_labels, batch_size=128, epochs=15, validation_data=(val_images, val_labels
```

- padding=’valid’ kích thước output giảm so với input tương ứng với kích thước của conv_filter

- padding=’same’ kích thước của output và input bằng nhau bằng cách sử dụng zero-padding (thêm vector 0 ở ngoài tương ứng kích thước của conv_filter)

- activation: hàm kích hoạt, ở đây dùng ReUL (max(0, z)) và softmax ($\exp(z)/\text{sum}(\exp(z_i))$)

- epochs: số lần lặp qua toàn bộ điểm dữ liệu

- Batch_size: số lượng hình ảnh lấy ra để train trong một lần ở mỗi epoch

- Chọn kích thước filter 3x3 để giảm khối lượng tính toán so với filter 5x5 or 7x7. Tuy nhiên hiêu quả chưa chắc sẽ cao hơn những TH kia

- Maxpooling2D(2x2) kích thước pooling vừa phải để giảm kích thước ban đầu, không làm mất quá nhiều thông tin của hình ảnh

- Optimizer: adam, có thể dùng sgd nhưng tốc độ học chậm hơn thể hiện qua val_accuracy qua mỗi epoch

- Loss function: Vì đầu ra là xác suất thuộc class nào nên dùng crossentropy là hợp lí hơn mean square error. Sparse_categorical_crossentropy: dùng cho output labels là nhãn của images tương ứng, nếu output labels là one-hot coding thì dùng categorical_crossentropy.

- Metric = accuracy: pred/labels: Lấy số predict đúng chia cho tổng số.

```
Train on 14034 samples, validate on 3000 samples
Epoch 1/15
14034/14034 [==============================] - 229s 16ms/step - loss: 1.2223 - accuracy: 0.4979 - val_loss: 0.8993 - val_accuracy: 0.6307
Epoch 2/15
14034/14034 [==============================] - 233s 17ms/step - loss: 0.9348 - accuracy: 0.6298 - val_loss: 0.8106 - val_accuracy: 0.6820
Epoch 3/15
14034/14034 [==============================] - 229s 16ms/step - loss: 0.7873 - accuracy: 0.6957 - val_loss: 0.6890 - val_accuracy: 0.7377
Epoch 4/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.6916 - accuracy: 0.7395 - val_loss: 0.6479 - val_accuracy: 0.7400
Epoch 5/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.6365 - accuracy: 0.7646 - val_loss: 0.5861 - val_accuracy: 0.7887
Epoch 6/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.5811 - accuracy: 0.7889 - val_loss: 0.5400 - val_accuracy: 0.8077
Epoch 7/15
14034/14034 [==============================] - 231s 16ms/step - loss: 0.4928 - accuracy: 0.8252 - val_loss: 0.3784 - val_accuracy: 0.8700
Epoch 8/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.4526 - accuracy: 0.8469 - val_loss: 0.4072 - val_accuracy: 0.8487
Epoch 9/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.4203 - accuracy: 0.8539 - val_loss: 0.3359 - val_accuracy: 0.8807
Epoch 10/15
14034/14034 [==============================] - 234s 17ms/step - loss: 0.3735 - accuracy: 0.8685 - val_loss: 0.3056 - val_accuracy: 0.8910
Epoch 11/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.3435 - accuracy: 0.8831 - val_loss: 0.2935 - val_accuracy: 0.8897
Epoch 12/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.3123 - accuracy: 0.8918 - val_loss: 0.2239 - val_accuracy: 0.9200
Epoch 13/15
14034/14034 [==============================] - 231s 16ms/step - loss: 0.3009 - accuracy: 0.8945 - val_loss: 0.2294 - val_accuracy: 0.9173
Epoch 14/15
14034/14034 [==============================] - 229s 16ms/step - loss: 0.2761 - accuracy: 0.9060 - val_loss: 0.1961 - val_accuracy: 0.9260
Epoch 15/15
14034/14034 [==============================] - 230s 16ms/step - loss: 0.2341 - accuracy: 0.9171 - val_loss: 0.1738 - val_accuracy: 0.9330
```

```
score = model.evaluate(test_images, test_labels, verbose=0)
print(score)
```

```
[0.4965379662315051, 0.8526666760444641]
```

Kết quả trên tập test là: 85.27%