MACHINE LEARNING (MAT3533)

BÀI THỰC HÀNH 9 – CONVOLUTION NEURAL NETWORK

A. SỬ DỤNG THƯ VIỆN TENSORFLOW-KERAS

Do kiến trúc và các bước tính toán trong một mạng CNN rất phức tạp, chúng ta chủ yếu thực nghiệm việc xây dựng mạng từ các tầng (Layers) có sẵn trong một số thư viện thông dụng gồm TensorFlow – Keras và Pytorch.

Một cách lý thuyết, chúng ta có thể tham khảo code xây dựng một mô hình CNN đơn giản từ thư viện cơ sở trong tệp nguồn CNN_numpy.ipynb đình kèm.

BINARY CLASSIFICATION

Ví dụ 1. Phân loại ảnh Chó – Mèo bằng CNN sử dụng thư viện Keras TensorFlow.

Trong bài thực hành thứ nhất, chúng ta sẽ sử dụng một mạng CNN áp dụng cho bài toán phân loại ảnh thành 02 lớp (Binary Classification), thông qua thư viện keras – tensorflow. Chú ý có một số phần code dưới đây được viết dựa trên các phiên bản thư viện TensorFlow thấp nên có thể không còn được chấp nhận trên các phiên bản cao hiện tại. Các bạn cần tự tìm kiếm giải pháp thay thế.

Dữ liệu training cho ví dụ này được lấy từ https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip (Lấy một phần từ dữ liệu https://www.kaggle.com/c/dogs-vs-cats/data), trong đó có 1000 bức ảnh chó và 1000 bức ảnh mèo.

Các bức ảnh đều là ảnh màu dạng jpg và kích thước có thể khác nhau (dữ liệu gốc).

1. Chuẩn bị hệ thống

Để thực hành ví dụ này, chúng ta cần cài đặt thư viện tương ứng.

Hãy thử import các thư viện sau để kiểm tra xem chúng đã được cài sẵn sàng hay chưa:

```
import tensorflow as tf
```

nếu không báo lỗi, hãy kiểm tra phiên bản hiện có của thư viện:

```
print(tf.__version__)
```

Những phần code dưới đây được xây dựng trên thư viện tensorflow ver. 1.8 nên có thể xảy ra không tương thích. Các bạn cần kiểm tra thông báo lỗi để xử lý hoặc thực nghiệm trên gg. Colab.

Để cài đặt trong Windows, các bạn có thể cái đặt Anaconda trước, sau đó trong cửa sổ lệnh, sử dụng lần lượt các lệnh sau để cài đặt:

```
conda create --name tensorflow python=3.5
activate tensorflow
conda install jupyter
conda install scipy
pip install tensorflow-gpu
```

Nếu các bạn không có GPU, các bạn có thể cài bản CPU only bằng cách thay dòng lệnh cuối cùng thành

```
pip install tensorflow
```

Trường hợp có thông báo không được quyền truy cập thư mục (EnvironmentError: [WinError 5] Access is denied: …), các bạn có thể thử cài đặt bằng cách thêm tùy biến –user:

```
pip3 install --upgrade tensorflow-gpu --user
```

Tương tự cho trường hợp chỉ cài bản CPU only.

Để cài đặt trong Linux (Ubuntu): Giả sử đã có Python 3 và pip, các bạn bổ sung Keras TensorFlow qua lệnh

```
pip3 install --user --upgrade tensorflow
```

2. Chuẩn bị dữ liệu

Các bạn có thể download tệp dữ liệu đã cho ở trên về máy, giải nén vào thư mục chưa chương trình, sau khi giải nén sẽ có các thư mục **train** và **validation**. Trong mỗi thư mục này chúng ta sẽ có hai thư mục là **cats** và **dogs**.

Ta cũng có thể load và giải nén dữ liệu trực tiếp bằng đoạn lệnh như sau:

```
!wget --no-check-certificate https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip

# download dataset

import zipfile

# Unzip the archive
local_zip = './cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall()

zip_ref.close()
```

Tiếp theo, ta thiết lập các đường dẫn đến các tệp dữ liệu

```
import os

base_dir =
'D:\\Teach_n_Train\\Advanced_Lessons_CV\\LABS\\cnn_binary\\cats_and_dogs'
# Change the base_dir to where you put dataset
print("Contents of base directory:")
print(os.listdir(base_dir))

print("\nContents of train directory:")
print(os.listdir(f'{base_dir}\\train'))

print("\nContents of validation directory:")
print(os.listdir(f'{base_dir}\\validation'))
```

Tham chiếu các phân lớp cho dữ liệu train và validation (tên thư mục là tên phân lớp)

```
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with training cat/dog pictures
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with validation cat/dog pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

print("\nContents of train directory:")
print(os.listdir(f'{base_dir}\\train'))

print("\nContents of validation directory:")
print(os.listdir(f'{base_dir}\\validation'))
```

Chúng ta sẽ lấy danh sách tên file từng loại ảnh train/validation và in thử một số tên file cũng như thống kê số file ảnh mỗi phân lớp để kiểm tra

```
train_cat_fnames = os.listdir( train_cats_dir )
train_dog_fnames = os.listdir( train_dogs_dir )

print(train_cat_fnames[:10])
print(train_dog_fnames[:10])

print('total training cat images :', len(os.listdir(      train_cats_dir ) ))
print('total training dog images :', len(os.listdir(      train_dogs_dir ) ))

print('total validation cat images :', len(os.listdir( validation_cats_dir ) ))
print('total validation dog images :', len(os.listdir( validation_dogs_dir ) ))
```

Chúng ta thử in ra một số ảnh của mỗi loại để kiểm tra

```
%matplotlib inline

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

pic_index = 0 # Index for iterating over images
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)

pic_index+=8

next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[ pic_index-8:pic_index]
               ]

next_dog_pix = [os.path.join(train_dogs_dir, fname)
                for fname in train_dog_fnames[ pic_index-8:pic_index]
               ]

for i, img_path in enumerate(next_cat_pix+next_dog_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

3. Khởi tạo mô hình CNN

Chúng ta sẽ sử dụng một mô hình CNN từ thư viện TensorFlow, với kiến trúc như sau:

- 03 tầng tích chập (convolution layers) kết hợp với Pooling (MaxPooling layers), kích thước mặt nạ tích chập là 3x3, kích thước pooling là 2x2.
- Sau mỗi tầng tích chập, hàm kích hoạt được sử dụng là ReLU.
- Tiếp theo là một tầng Full Connection và tầng tiếp theo là Flatten (duỗi các kết quả đầu ra thành vector).
- Cuối cùng là một tầng phân loại ở đầu ra, sử dụng hàm Sigmoid (logistic – vì đây chỉ có 02 lớp)

Khởi tạo một CNN như trên:

```
import tensorflow as tf
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes color
    tf.keras.layers.InputLayer(shape=[150, 150, 3]), # Specify the input shape
    tf.keras.layers.Conv2D(16, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron.
    # It will contain a value from 0-1 where 0 for 1 class ('cats')
    # and 1 for the other ('dogs')
```

```
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Ta có thể xem lại kiến trúc mô hình qua lệnh

```
model.summary()
```

4. Huấn luyện mô hình

Trước hết ta thiết lập phương pháp giải bài toán tối ưu. Ở đây ta dùng RMSprop thay cho SGD (do có thể tự động chọn tham số học)

```
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

Tiếp theo chúng ta chuẩn hóa dữ liệu bằng các đưa cường độ pixel về khoảng [0, 1], chỉnh kích thước ảnh về 150x150 và điều hướng các thư mục chứa dữ liệu training

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

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))
# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=20,
                                                         class_mode  = 'binary',
                                                         target_size = (150, 150))
```

Gọi lệnh huấn luyện mô hình

```
history = model.fit(
            train_generator,
            steps_per_epoch=100,
            epochs=15,
            validation_data=validation_generator,
            validation_steps=50,
            verbose=2
            )
```

Xem xét một số kết quả: Trong đoạn lệnh dưới đây ta sẽ xem kết quả các feature map ở đầu ra của mỗi layer tích chập, và quan sát những đặc trưng đã được giữ lại như thế nào.

```
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img
# Define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model
successive_outputs = [layer.output for layer in model.layers]
model(tf.keras.Input((150, 150, 3)))
visualization_model = tf.keras.models.Model([model.inputs], outputs =successive_outputs)
```

```
# Prepare a random input image from the training set.
cat_img_files = [os.path.join(train_cats_dir, f) for f in train_cat_fnames]
dog_img_files = [os.path.join(train_dogs_dir, f) for f in train_dog_fnames]
img_path = random.choice(cat_img_files + dog_img_files)
img = load_img(img_path, target_size=(150, 150)) # this is a PIL image
x = img_to_array(img) # Numpy array with shape (150, 150,3)
x = x.reshape((1,) + x.shape) # Numpy array with shape (1, 150,150, 3)
# Scale by 1/255
x /= 255.0
# Run the image through the network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)# These are the names of the
layers, so we can have them as part of our plot
layer_names = [layer.name for layer in model.layers]
# Display the representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
    if len(feature_map.shape) == 4:
        #----------------------------------------
        # Just do this for the conv / maxpool layers, not the fully-connected layers
        #----------------------------------------
        n_features = feature_map.shape[-1] # number of features in the feature map
        size = feature_map.shape[ 1] # feature map shape (1, size, size, n_features)
        # Tile the images in this matrix
        display_grid = np.zeros((size, size * n_features))
        #-------------------------------------------------
        # Postprocess the feature to be visually palatable
        #-------------------------------------------------
        for i in range(n_features):
            x = feature_map[0, :, :, i]
            x -= x.mean()
            if (x.std () != 0):
                x /= x.std ()
            x *= 64
            x += 128
            x = np.clip(x, 0, 255).astype('uint8')
            display_grid[:, i * size : (i + 1) * size] = x
        # Tile each filter into a horizontal grid
        #-----------------
        # Display the grid
        #-----------------
        scale = 20. / n_features
        plt.figure( figsize=(scale * n_features, scale) )
        plt.title ( layer_name )
        plt.grid ( False )
        plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```

Trong đoạn lệnh trên, chúng ta sẽ in ra kết quả sau mỗi tầng (“ảnh” đầu ra sau mỗi tầng convolution và maxpool) để hiểu thêm cách hoạt động của CNN để cho ra dãy so sánh được ở tầng flatten. **Chú ý đoạn lệnh này có thể phát sinh lỗi đối với một vài phương thức trong thư viện phiên bản cao của TensorFlow.**

Đoạn lệnh tiếp theo chúng ta thử sử dụng mô hình với các tham số đã được huấn luyện (train) để dự đoán (predict) cho một số ảnh đầu vào tùy ý (chú ý cần có chó hoặc mèo). Chúng ta hãy sửa đoạn lệnh này để áp dụng cho một list các tên file ảnh đầu vào, thay vì từng ảnh như trong hướng dẫn.

```
import numpy as np

#from google.colab import files
from keras.preprocessing import image

#uploaded=files.upload()

#for fn in uploaded.keys():
fn =  '4.jpg' # change it to your image file
```

```
# predicting images
path='D:\\Teach_n_Train\\Advanced_Lessons_CV\\LABS\\'+
        'cnn_binary\\cats_and_dogs\\validation\\4.jpg' # change it to your image
img=image.load_img(path, target_size=(150, 150))

x=image.img_to_array(img)
x /= 255
x=np.expand_dims(x, axis=0)
images = np.vstack([x])

classes = model.predict(images, batch_size=10)

print(classes[0])

if classes[0]>0.5:
    print(fn + " is a dog" )
else:
    print(fn + " is a cat" )
```

Tính toán các độ đo độ chính xác (accuracy) và hàm tổn thất, sau đó vẽ đồ thị để minh họa.

```
#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
acc      = history.history[     'accuracy' ]
val_acc  = history.history[ 'val_accuracy' ]
loss     = history.history[     'loss' ]
val_loss = history.history['val_loss' ]

epochs   = range(len(acc)) # Get number of epochs

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot  ( epochs,     acc )
plt.plot  ( epochs, val_acc )
plt.title ('Training and validation accuracy')
plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot  ( epochs,     loss )
plt.plot  ( epochs, val_loss )
plt.title ('Training and validation loss'   )
```

Bài tập tự thực hành 1

- Quan sát Loss Function và Accuracy trên tập Training và trên tập Validation. Giải thích những kết quả quan sát được.
- Trong thư viện tensorflow.keras.layer có một loại layer là Dropout(rate = <rate_drop>), trong đó giá trị của rate_drop ∈ [0, 1). Layer Dropout cho phép bỏ bớt dữ liệu train với tỉ lệ là rate_drop, nhằm mục đích tránh overfit. Hãy bổ sung các tầng Dropout này vào giữa các tầng Convolution, rate_drop khoảng từ 0.3 đến 0.5 và quan sát kết quả sau khi thay đổi.
- Hãy đưa kích thước các ảnh (resize) về 150x150:
- Thực hiện chuyển ảnh thành vector, sau đó sử dụng mô hình Hồi quy Logistic để phân loại ảnh. Đánh giá độ chính xác bằng các độ đo.
- Dùng PCA giảm số chiều về còn 225. Sau đó sử dụng mô hình ANN đã có để phân loại (có thể tham khảo chương trình đã được cung cấp cho bộ ảnh mặt người). So sánh độ chính xác so với các phương pháp: Logistic, ANN và CNN.
- Sưu tầm một số ảnh về 02 loài động vật trên, đặt trong cùng thư mục (>= 10 ảnh) .

- Chạy predict cho toàn bộ các ảnh sưu tập và đánh giá độ chính xác.

MULTINOMIAL CLASSIFICATION

Ví dụ dưới đây sẽ minh họa trường hợp phân loại nhiều lớp.

Ví dụ 2. Phân loại ảnh Chó – Mèo – Gấu trúc bằng CNN sử dụng thư viện Keras TensorFlow.

Trong bài thực hành thứ nhất, chúng ta sẽ sử dụng một mạng CNN áp dụng cho bài toán phân loại ảnh thành 03 lớp (Multi Classification), thông qua thư viện keras – tensorflow. Dữ liệu training cho ví dụ này được lấy từ https://www.kaggle.com/datasets/ashishsaxena2209/animal-image-datasetdog-cat-and-panda trong đó có 1000 bức ảnh chó, 1000 ảnh panda và 1000 bức ảnh mèo – tức là có 03 phân lớp.

Các bức ảnh đều là ảnh màu dạng jpg và kích thước có thể khác nhau (dữ liệu gốc)

1. Chuẩn bị hệ thống

Nếu chúng ta đã cài đặt từ ví dụ 1 thì bỏ qua phần này. Chúng ta sẽ tham khảo chương trình ví dụ trong Kaggle về việc sử dụng thư viện TensorFlow.Keras, do đó ta cần nhúng một số thư viện bổ sung như dưới đây:

```
import tensorflow as tf # Thư viện tensor.flow

from tensorflow import keras # Lớp keras và các công cụ liên quan

import tensorflow_datasets as tfds # Gọi và xử lý dữ liệu (với label là tên folder chứa dữ liệu)
```

Ngoài ra chúng ta cũng sử dụng một số thư viện khác để đọc dữ liệu, đánh giá độ chính xác và xuất hình ảnh như

```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
```

2. Mã chương trình

Đoạn code sau đây thực hiện gọi các thư viện cần thiết:

```
#!pip install tensorflow_datasets

import os
import numpy as np # linear algebra
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import tensorflow as tf
import PIL
import PIL.Image
from tensorflow import keras
import tensorflow_datasets as tfds
```

Đoạn code tiếp theo thực hiện việc khởi dựng mô hình CNN với kiến trúc gồm:

- 3 tầng Convolution & MaxPooling nối tiếp: Tầng đầu gồm 16 filters; Tầng 2 gồm 32 filters và tầng ba gồm 64 filters, tất cả có cùng cỡ 3x3;
- Tiếp theo tầng Convolution thứ ba là tầng Flatten;
- Tiếp theo đó ta có tầng FullyConnected (FullConnection) FC với đầu ra là vector đặc trưng 512 phần tử. Tất cả các tầng Conv và FC đều sử dụng activation là ReLU;
- Cuối cùng, ta có tầng đầu ra với activation là SoftMax chia 03 loại.

```
# Trains a model to classify images of 3 classes: cat, dog, and panda
```

```
def gen_model():

    # Defines & compiles the model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150,
3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    keras.layers.Dropout(rate=0.15), #adding dropout regularization throughout the
model to deal with overfitting
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    keras.layers.Dropout(rate=0.1),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    keras.layers.Dropout(rate=0.10),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),

    # 3 output neuron for the 3 classes of Animal Images
    tf.keras.layers.Dense(3, activation='softmax')
    ])

    from tensorflow.keras.optimizers import RMSprop

    model.compile(loss='categorical_crossentropy',
              optimizer="adam",
              metrics=['acc'])
    return model
```

Tiếp theo chúng ta thiết lập dữ liệu và gọi mô hình để huấn luyện và dự báo

```
# Trains a model to classify images of 3 classes: cat, dog, and panda
def train_test_animals():
    # Creates an instance of an ImageDataGenerator called train_datagen, and a
train_generator, train_datagen.flow_from_directory

    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    #splits data into training and testing(validation) sets
    train_datagen =ImageDataGenerator(rescale=1./255, validation_split=0.25)

    import matplotlib.pyplot as plt


    #training data
    train_generator = train_datagen.flow_from_directory(

'D:\\Teach_n_Train\\Advanced_Lessons_CV\\LABS\\cnn_binary\\CNN_MultiClass\\animals',
# Source directory
        target_size=(150, 150),  # Resizes images
        batch_size=15,
        class_mode='categorical',subset = 'training')


    epochs = 2
    #Testing data
    validation_generator = train_datagen.flow_from_directory(
```

```
'D:\\Teach_n_Train\\Advanced_Lessons_CV\\LABS\\cnn_binary\\CNN_MultiClass\\validation'
,
    target_size=(150, 150),
    batch_size=15,
    class_mode='categorical',
    subset='validation') # set as validation data

    model = gen_model()
    #Model fitting for a number of epochs
    history = model.fit_generator(
      train_generator,
      steps_per_epoch=150,
      epochs=epochs,
      validation_data = validation_generator,
      validation_steps = 50,
      verbose=1)



    acc = history.history['acc']
    val_acc = history.history['val_acc']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    #This code is used to plot the training and validation accuracy
    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()

    # returns accuracy of training
    print("Training Accuracy:"), print(history.history['acc'][-1])
    print("Testing Accuracy:"), print (history.history['val_acc'][-1])
```

Cuối cùng chúng ta gọi thực hiện mô hình:

```
train_test_animals()
```

Bài tập tự thực hành:

1. **So sánh các mô hình**

- Hãy đưa kích thước các ảnh (resize) về 150x150:
- Thực hiện chuyển ảnh thành vector, sau đó sử dụng mô hình Hồi quy SOFTMAX để phân loại ảnh. Đánh giá độ chính xác bằng các độ đo.
- Dùng PCA giảm số chiều về còn 225. Sau đó sử dụng mô hình ANN đã có để phân loại. So sánh độ chính xác so với các phương pháp: SOFTMAX, ANN và CNN.

2. **Kiểm tra độ chính xác**

a) Hãy thay đổi dữ liệu và chương trình để chạy dự đoán cho một số ảnh mẫu riêng biệt và kiểm tra độ chính xác.

b) Hãy tự điều chỉnh chương trình để đưa về phân loại 02 lớp (chỉ cho chó – mèo) nhưng giữ nguyên hàm kích hoạt tầng đầu ra là SOFTMAX;

- Chỉ lấy 300 ảnh làm dữ liệu training và 100 ảnh làm dữ liệu validation
- Chạy dự đoán kết quả

c) Hãy bổ sung các tầng Dropout này vào giữa các tầng Convolution, rate_drop khoảng từ 0.3 đến 0.5 và quan sát kết quả sau khi thay đổi.

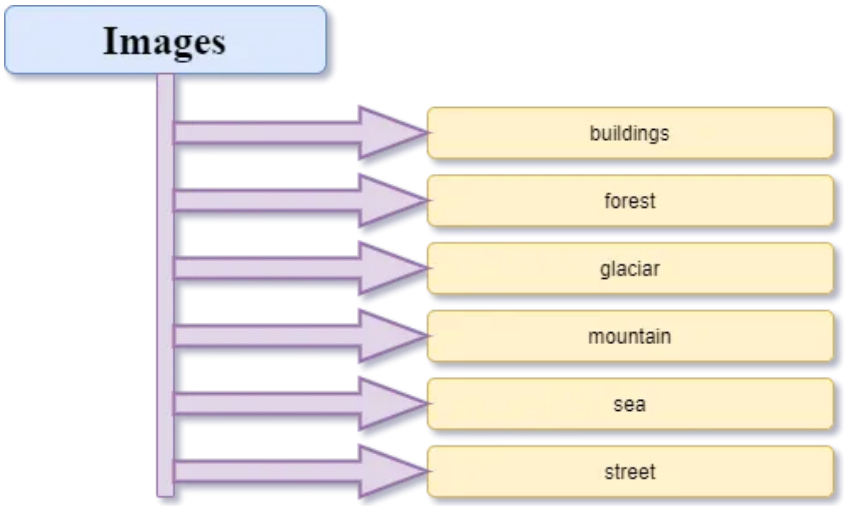
Ví dụ 3 (Bài tập tự thực hành). Hãy sử dụng lại bộ ảnh dữ liệu các khuôn mặt facedata với 11 phân loại đã có trong ví dụ trước. Chia dữ liệu thành các tập train:test tỷ lệ 8:2. Sau đó huấn luyện mô hình CNN đơn giản trong ví dụ 2 và thực hiện dự đoán cho tập test.

a) Tính độ chính xác của mô hình theo các độ đo Accuracy, Precision và Recall trên cả tập train và test.
b) Giải thích kết quả thu được
c) So sánh với kết quả khi kết hợp giảm chiều và sử dụng mô hình ANN hoặc Logistic để phân loại (đã có trong bài trước).
d) Thực hiện các mô hình CNN và mô hình kết hợp PCA-ANN đã đề cập ở trên, đã được huấn luyện với toàn bộ tập dữ liệu, sau đó chạy dự đoán cho khoảng 05 ảnh chân dung bất kỳ có định dạng như bài thực hành phần ANN (kích thước 320x243, khuôn mặt người ở vị trí hơi lệch bên phải với chiều nhìn vào).

B. SỬ DỤNG THƯ VIỆN PYTORCH

Trong ví dụ này chúng ta thực hiện bài toán phân loại ảnh cho tập dữ liệu Intel Image có từ Kaggle. Dữ liệu có thể lấy từ link sau: https://www.kaggle.com/datasets/puneet6060/intel-image-classification

Dữ liệu gồm khoảng 25 nghìn ảnh cỡ 150 x 150, phân bố trong 06 nhãn phân loại dữ liệu (label) tương ứng với chỉ số từ 0 đến 6 gồm:



Chúng ta lấy các phần dữ liệu seg_train và seg_test, trong đó mỗi phân loại dữ liệu được đặt trong một thư mục có tên phân loại như hình trên, tương tự như ví dụ phân loại ảnh ở phần A.

Đoạn chương trình dưới đây liên kết với thư viện pytorch và đọc dữ liệu từ các thư mục ảnh tương ứng. Chú ý code tương thích với torch 1.3 và torchvision 0.4.1, các version khác có thể không tương thích, các bạn tự tìm cách điều chỉnh phiên bản phù hợp.

```
import torch
import torchvision
from torchvision import transforms
from torchvision.datasets import ImageFolder

DATA_PATH = "D:/Teach_n_TrainAdvanced_Lessons_CV/LABS/data//input/intel-image-
classification/"

#train and test data directory
data_dir = DATA_PATH + "seg_train/seg_train/"
test_data_dir = DATA_PATH + "seg_test/seg_test"


#load the train and test data
dataset = ImageFolder(data_dir,transform = transforms.Compose([
    transforms.Resize((150,150)),transforms.ToTensor()
]))
test_dataset = ImageFolder(test_data_dir,transforms.Compose([
    transforms.Resize((150,150)),transforms.ToTensor()
]))
```

Xây dựng một số chức năng hiển thị ảnh (theo nhóm) và label phân loại ảnh:

```
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
from torchvision.utils import make_grid
import matplotlib.pyplot as plt


img, label = dataset[0]
print(img.shape,label)
```

```
def display_img(img,label):
    print(f"Label : {dataset.classes[label]}")
    plt.imshow(img.permute(1,2,0))

def show_batch(dl):
    """Plot images grid of single batch"""
    for images, labels in dl:
        fig,ax = plt.subplots(figsize = (16,12))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images,nrow=16).permute(1,2,0))
        break

#display the first image in the dataset
display_img(*dataset[0])


batch_size = 128
val_size = 2000
train_size = len(dataset) - val_size

train_data,val_data = random_split(dataset,[train_size,val_size])
print(f"Length of Train Data : {len(train_data)}")
print(f"Length of Validation Data : {len(val_data)}")

#output
#Length of Train Data : 12034
#Length of Validation Data : 2000

#load the train and validation into batches.
train_dl = DataLoader(train_data, batch_size, shuffle = True, num_workers = 4, pin_memory
= True)
val_dl = DataLoader(val_data, batch_size*2, num_workers = 4, pin_memory = True)

show_batch(train_dl)
```

Xây dựng mạng CNN với 3 tầng Convolution + ReLU và MaxPool. Số filter là 16 kích thước 3x3, stride = (1,1), padding = 1. Cách xây dựng kiến trúc CNN là tương tự như của Keras, với gói chứa các kiến trúc layer có trong torch.nn

```
import torch.nn as nn
import torch.nn.functional as F

class NaturalSceneClassification(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(

            nn.Conv2d(3, 32, kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.Conv2d(32,64, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Conv2d(64, 128, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.Conv2d(128 ,128, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Conv2d(128, 256, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
```

```
            nn.Conv2d(256,256, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),

            nn.Flatten(),
            nn.Linear(82944,1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512,6)
        )

    def forward(self, xb):
        return self.network(xb)
```

Chúng ta xây dựng lớp cho mô hình phân loại ảnh sử dụng kiến trúc trên như sau:

```
class ImageClassificationBase(nn.Module):

    def training_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                      # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))
```

Trong phần train và validation, ta sử dụng loss dạng cross-entropy thông qua thiết lập:

loss = F.cross_entropy(out, labels)

Đoạn lệnh tiếp theo thiết lập các Hyperparameters cho phần Training Model, cũng như độ đo đánh giá:

```
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))


@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)


def fit(epochs, lr, model, train_loader, val_loader, opt_func = torch.optim.SGD):
```

```
    history = []
    optimizer = opt_func(model.parameters(),lr)
    for epoch in range(epochs):

        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)

    return history
```

Cuối cùng, thực hiện training thông qua việc gọi phương thức fit đã xây dựng ở trên, với số epoch = 30. Các bạn cần điều chỉnh tham số này theo cấu hình phù hợp:

```
num_epochs = 30
opt_func = torch.optim.Adam
lr = 0.001
#fitting the model on training data and record the result after each epoch
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

Bài tập tự thực hành:

- Từ các đoạn lệnh cơ sở nói trên, hãy hoàn chỉnh và thực hiện training, predict trên các tập dữ liệu train và validation tương ứng.
- Thực hiện phân loại cho bộ ảnh Dogs – Cats – Pandas bằng mô hình CNN sử dụng thư viện Pytorch.