BÀI THỰC HÀNH 5: ỨNG DỤNG MẠNG NEURAL TÍCH CHẬP

Hướng dẫn nộp bài: Nộp file jupyter notebook (đuôi là .jpynb), đặt tên là MSSV_BaiThucHanh5.jpynb + file PDF báo cáo trả lời các câu hỏi: MSSV_BaiThucHanh5.pdf
Nộp qua course, giảng viên sẽ tạo submission sau mỗi buổi học.

1. Các bộ dữ liệu.

Bộ dữ liệu 1: Bộ dữ liệu ảnh X-quang phổi.

+ Số lượng dữ liệu: 5216 ảnh cho huấn luyện, 16 cho dev và 624 cho test.

+ Định dạng: jpg.

+ Số lượng nhãn: 2. Gồm 2 nhãn: NORMAL - bình thường và PNEUMONIA - nhiễm bệnh (phân lớp nhị phân).

+ Link: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Bộ dữ liệu 2: Bộ dữ liệu về trang sức (Jewellery).

+ Số lượng dữ liệu: 1569 cho train, 250 cho test.

+ Định dạng: jpg.

+ Số lượng nhãn: 5 (phân lớp đa lớp).

BRACELET.

EARRINGS.

NECKLACE.

RINGS.

WRISTWATCH.

+ Link: https://github.com/princesegzy01/Jewellery-Classification

Cả 2 bộ dữ liệu trên đều được tổ chức thành dạng các thư mục, mỗi thư mục sẽ đại diện cho một nhãn (lớp) mà ảnh đó thuộc về.

VD: Với bộ dữ liệu X-ray.

Train:

NORMAL: pic1.jpg, pic2.jpg, ...

PNEUMONIA: pic3.jpg, pic4.jpg.

Dev: tương tự.

Test: tương tự.

2. Đọc dữ liệu.

Để đọc dữ liệu ảnh như trên, ta dùng hàm ***image_dataset_from_directory*** được hỗ trợ sẵn bởi keras. Hàm này gồm các thông số quan trọng sau:

+ *directory*: đường dẫn đến thư mục chứa dữ liệu.

+ *labels*: nhãn được dùng trong bộ dữ liệu. Mặc định là "infered", tức là lấy từ tên thư mục trong bộ dữ liệu.

+ *label_mode*: loại nhãn. 'int' dùng cho phân lớp đa nhãn, 'binary' dùng cho phân lớp nhị phân, và 'categorical' là nhãn được biểu diễn ở dạng vector (tương tự to_categorical trước đó).

+ *color_mode*: hệ màu. Mặc định là rgb.

+ *batch_size*: size của data. Mặc định là 32.

Xem thêm: https://keras.io/api/preprocessing/image/

Ví dụ: đoạn lệnh sau sẽ đọc dữ liệu từ tập huấn luyện của bộ Chest-X-ray:

```
train_set = image_dataset_from_directory(
    "drive/My Drive/Chest-X-ray/dataset/train/",
    labels="inferred",
    label_mode="binary",
    class_names=['NORMAL', 'PNEUMONIA'],
    color_mode=COLOR_MODE,
    batch_size=BATCH_SIZE,
    image_size=(IMG_SIZE, IMG_SIZE),
    interpolation="bilinear",
)
```

Lưu ý: hàm ***image_dataset_from_directory*** sẽ trả về dữ liệu có dạng (X, y) theo từng batch nhỏ với kích thước xác định (dựa vào BATCH_SIZE trước đó). Do đó, ta có thể trực tiếp sử dụng từng batch này vào huấn luyện mô hình.

Nếu không thích dùng hàm ***image_dataset_from_directory***, các bạn có thể dùng các thư viện đọc hình ảnh truyền thống bằng *PIL image* hay *OpenCV*, sau đó chuyển thành dạng ma trận.

3. Xây dựng mô hình.

Các bạn có thể tự xây dựng các mô hình mạng neural tích chập như đã thực hành trong *Bài thực hành 3*.

Trong bài thực hành này giới thiệu một kỹ thuật mới là học chuyển tiếp (transfer learning). Kỹ thuật này sẽ tận dụng các tham số của mô hình đã được huấn luyện trước đó như VGG16, ResNET-50, sau đó tinh chỉnh lại các layer để áp dụng cho bài toán đang xử lý.

Danh sách các mô hình được huấn luyện sẵn có thể tham khảo tại link sau:

https://keras.io/api/applications/

Ví dụ với VGG-16:

+ Load mô hình đã có trước đó:

```
from keras.applications.vgg16 import VGG16

vgg = VGG16(include_top=False, input_shape=(IMG_SIZE,
IMG_SIZE, IMG_CHANNEL))
```

Ghi chú: IMG_SIZE là kích thước ảnh, IMG_CHANNEL là số kênh của ảnh.

+ Chúng ta sẽ "đóng băng" (freeze) các lớp trước đó. Việc này sẽ giúp cho các trọng số của mô hình VGG-16 trước đó được bảo toàn.

```
for layer in vgg.layers:
    layer.trainable = False
```

+ Tinh chỉnh đầu ra: Với bài toán phân lớp nhị phân, chúng ta tinh chỉnh lại mô hình VGG-16 bằng cách điều chỉnh các lớp ở đầu ra: đầu ra của bài toán chỉ gồm 2 giá trị là 0 hoặc 1. Do đó ta dùng lớp Dense để làm lớp FC cho output.

```
flat = Flatten()(vgg.layers[-1].output)
fc1 = Dense(1024, activation='relu')(flat)
output = Dense(1, activation='sigmoid')(fc1)

model = Model(inputs=vgg.inputs, outputs=output)
```

Kết quả build mô hình:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
input_2 (InputLayer)         [(None, 227, 227, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 227, 227, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 227, 227, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 113, 113, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 113, 113, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 113, 113, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              25691136
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 1025
=================================================================
Total params: 40,406,849
Trainable params: 25,692,161
Non-trainable params: 14,714,688
_________________________________________________________________
```

Huấn luyện mô hình:

```
model.fit(train_set,epochs = 10,validation_data=dev_set)
```

Có thể sử dụng thêm kỹ thuật *EarlyStopping* để tránh overfit.

Dự đoán mô hình: Do dữ liệu ban đầu đọc bằng hàm ***image_dataset_from_directory*** được chia thành từng batch, do đó, ta sẽ dự đoán dữ liệu trên mỗi batch, sau đó merge kết quả dự đoán (y_pred) và nhãn của tập test (y_true) ở mỗi batch lại thành một list.

```
y_pred_total = []
y_true = []

for img, label in test_set:
    y_pred = model.predict(img)
    y_pred_total += [1 if i > 0.5 else 0 for i in y_pred]
    y_true += np.array(label).flatten().tolist()

accuracy_score(y_true, y_pred_total)*100
```

Tới đây, các bạn có thể tính độ chính xác của mô hình dự đoán bằng các độ đo khác như precision, recall và F1-score.

BÀI TẬP:

Bài 1: Thực hiện huấn luyện mô hình mạng neural dùng cho phân lớp nhị phân cho bộ dữ liệu Chest X-ray bằng kỹ thuật transfer learning. Tinh chỉnh lại các mô hình sau: VGG-16, VGG-19, ResNet-50. Đánh giá mô hình bằng các độ đo: *accuracy, precision, recall và F1-score*. Có kết luận gì về khả năng phân lớp của các mô hình?

Bài 2: Thực hiện các yêu cầu của Bài 1 với bộ dữ liệu Jewellery.

Gợi ý:

+ Dùng hàm loss là: *categorical_crossentropy*.

+ Dùng lớp Dense cuối với unit là 5.

+ Tạo thêm tập validation từ tập train bằng cách lấy 10% dữ liệu từ tập train:

```
dev_set = image_dataset_from_directory(
    "drive/MyDrive/DL/dataset/Jewellery/train/",
    labels="inferred",
    label_mode="categorical",
    class_names=['BRACELET','EARRINGS','NECKLACE',
'RINGS', 'WRISTWATCH'],
    color_mode=COLOR_MODE,
    batch_size=BATCH_SIZE,
```

```
    image_size=(IMG_SIZE, IMG_SIZE),
    interpolation="bilinear",
    subset="validation",
    seed=1,
    validation_split=0.1
)
```