# THỰC HÀNH BUỔI 3: MÔ HÌNH CNN

```
import numpy as np #tinh toan ma tran
import pandas as pd #tien xu ly du lieu, input output
import matplotlib.pyplot as plt #ve ra bieu do, hinh anh
import os #he dieu hanh, doc file, ghi file, ghi o dia, thu muc
import cv2 #opencv: xu ly thi giac may tinh
from sklearn.model_selection import train_test_split #chia tap train, test
from keras.models import Sequential #khai bao tính tuần tự của mảng
#Activatio: hàm kích hoạt, Flatten: làm phẳng 2d,3d
from keras.layers import Dense, Dropout, Activation, Flatten
#Trích đặc trưng
from keras.layers import Convolution2D, MaxPooling2D
from tensorflow.keras import utils #xử lý thông tin, hồi quy
import tensorflow as tf
from tensorflow import keras

#image path
path = "lab2/"
#animal categories
categories = ['dogs', 'panda', 'cats']
```

```
#Hien thi hinh anh
for category in categories:
  fig, _ = plt.subplots(3,4) #dạng lưới 3 dòng, 4 cột
  fig.suptitle(category)
  for k, v in enumerate(os.listdir(path+category)[:12]): #enumarate lấy ngẫu nhiên
    img = plt.imread(path+category+'/'+v)
    plt.subplot(3, 4, k+1) #in 3 trong 4, từng cái tăng dần
    plt.axis('off') #trục ngang, xuống hàng
    plt.imshow(img) #show mỗi ảnh con
  plt.show()
```

```
from string import hexdigits
#Tien xu ly va han dau vao
data = []
labels = [] #nhãn
imagePaths = [] #đọc all đường dẫn của tấm hình
# HEIGHT = 32
# WIDTH = 55
HEIGHT = 128
WIDTH = 128

N_CHANNELS = 3

#grab the image path and random shuffle them
#đọc lên các hình và đường dẫn
for k, category in enumerate(categories):
  for f in os.listdir(path+category):
    imagePaths.append([path+category+'/'+f, k]) #k0:dogs, k1:panda, k2:cats

#Xáo trộn ảnh, để ảnh không theo thứ tự
import random
random.shuffle(imagePaths)
print(imagePaths[:10])

#loop over the input images
for imagePath in imagePaths:
  #load image, resize to HEIGHT*WIDTH pixels (ignore aspect ratio), store image
  #in data list
  image = cv2.imread(imagePath[0])
  image = cv2.resize(image, (WIDTH,HEIGHT)) #.flatten()
  data.append(image)

  #extract the class label form the image path and update label list
  label = imagePath[1]
  labels.append(label)

#scale hinh anh cung dang trong numpy
#scale the raw pixel intensities to range [0,1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
```

## Chia dữ liệu

```
#chia du lieu huan luyen
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2, random_state=42)

#Tien xu ly lop
trainY = utils.to_categorical(trainY, 3)
```

```
#----------------------
EPOCHS = 5
INIT_LR = 1e-3
BS = 32
#----------------------
```

**Mô hình Lenet**

```
#Xay dung kien truc mo hinh CNN
model = Sequential()
model.add(Convolution2D(32, (2,2), activation='relu', input_shape=(HEIGHT, WIDTH, N_CHANNELS)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Convolution2D(32, (2,2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation= 'relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

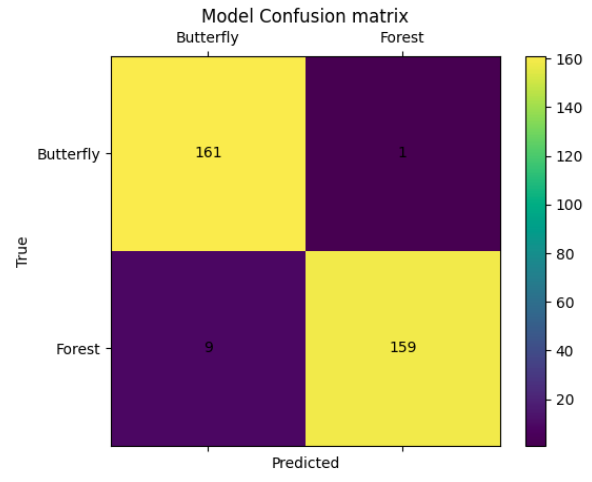
---

```
#----------------------
class_names = categories
#-------------------------

from tensorflow.keras.applications import InceptionV3, MobileNet, VGG16, DenseNet121, EfficientNetB7
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from keras import layers
from keras import models

print("[INFO] compiling model...")
inceptionv3 = InceptionV3(input_shape=(128,128,3), include_top=False, weights='imagenet')
for layer in inceptionv3.layers:
  layer.trainable = False

model = Sequential()
model.add(inceptionv3)
#model.add(layers.AveragePooling2D(8,8), padding= 'valid', name='average_pool'))
model.add(GlobalAveragePooling2D())
model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(len(class_names), activation='softmax'))

opt = tf.keras.optimizers.legacy.Adam(learning_rate=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
model.summary()
```

## Vẽ Confusion Matrix

```
from numpy import argmax
from sklearn.metrics import confusion_matrix, accuracy_score

pred = model.predict(testX)
predictions = argmax(pred, axis=1)

cm = confusion_matrix(testY, predictions)

fig = plt.figure()
ax= fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Model confusion matrix')
fig.colorbar(cax)
ax.set_xticklabels([''] + categories)
ax.set_yticklabels([''] + categories)

for i in range(3):
  for j in range(3):
    ax.text(i,j,cm[j,i], va='center', ha='center')

plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

accuracy = accuracy_score(testY, predictions)
print("Accuracy : %.2f%%" % (accuracy*100.0))
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step
```




```
Accuracy: 96.97%
```

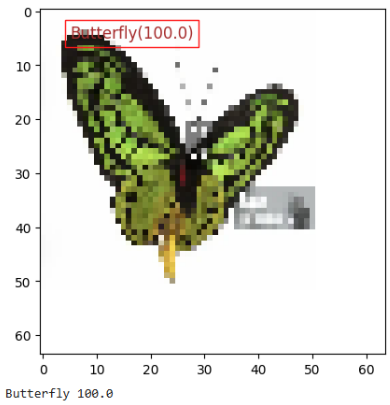
## Lưu mô hình

```
model.save("lab2.h5")
```

## Kiểm tra - Inference:

```
from numpy import argmax
import PIL
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image


img_path="/content/Untitled.jpg"

img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Chuyển BGR -> RGB
img = cv2.resize(img, (height,width))  # Resize về đúng kích thước mô hình cần (VD: 224x224)

img_preprocessed = np.array(img, dtype="float32") / 255.0
img_preprocessed = np.expand_dims(img_preprocessed, axis=0)

pred = model.predict(img_preprocessed)
Res = argmax(pred, axis=1) # return to label
print(pred)

Result_Text = "{0}({1})".format(categories[Res[0]],round(pred[0][Res[0]]*100,ndigits=2))

plt.text(5, 5, Result_Text, color="brown",fontsize="large",bbox=dict(fill=False, edgecolor='red', linewidth=1))
plt.imshow(img)
plt.show()
print(categories[Res[0]],pred[0][Res[0]]*100)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
[[1.0000000e+00 5.8842675e-10]]
```




```
Butterfly 100.0
```

## TỰ CÀI ĐẶT

```
# ----------------------------
# Các hàm tiện ích
# ----------------------------
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

def softmax(x):
    exp_x = np.exp(x - np.max(x))
    return exp_x / exp_x.sum(axis=0, keepdims=True)

def conv2d(X, kernel):
    k_h, k_w = kernel.shape
    h, w = X.shape
    out_h, out_w = h - k_h + 1, w - k_w + 1
    out = np.zeros((out_h, out_w))
    for i in range(out_h):
        for j in range(out_w):
            out[i, j] = np.sum(X[i:i+k_h, j:j+k_w] * kernel)
    return out

def average_pool(X, size=2):
    h, w = X.shape
    new_h, new_w = h // size, w // size
    pooled = np.zeros((new_h, new_w))
    for i in range(new_h):
        for j in range(new_w):
            pooled[i, j] = np.mean(X[i*size:(i+1)*size, j*size:(j+1)*size])
    return pooled
```

```python
# ----------------------------
# Định nghĩa mô hình LeNet-5
# ----------------------------
class LeNet5:
    def __init__(self):
        # Khởi tạo các kernel ngẫu nhiên
        self.C1_kernels = np.random.randn(6, 5, 5) * 0.1
        self.C3_kernels = np.random.randn(16, 6, 5, 5) * 0.1
        self.FC1_weights = np.random.randn(120, 16 * 5 * 5) * 0.1
        self.FC2_weights = np.random.randn(84, 120) * 0.1
        self.FC3_weights = np.random.randn(3, 84) * 0.1

    def forward(self, X):
        # C1
        self.C1 = np.array([conv2d(X, k) for k in self.C1_kernels])
        self.C1_act = sigmoid(self.C1)
        # S2
        self.S2 = np.array([average_pool(x) for x in self.C1_act])
        # C3
        self.C3 = np.array([
            sum(conv2d(self.S2[i], self.C3_kernels[o, i]) for i in range(6))
            for o in range(16)
        ])
        self.C3_act = sigmoid(self.C3)
        # S4
        self.S4 = np.array([average_pool(x) for x in self.C3_act])
        # Flatten
        self.flatten = self.S4.reshape(-1, 1)
        # FC1
        self.FC1 = sigmoid(np.dot(self.FC1_weights, self.flatten))
        self.FC2 = sigmoid(np.dot(self.FC2_weights, self.FC1))
        self.FC3 = softmax(np.dot(self.FC3_weights, self.FC2))
        return s
```

# CÀI ĐẶT VỚI TORCH

```
# ============================================================
# CÀI ĐẶT THƯ VIỆN CẦN THIẾT
# ============================================================
import os
import random
import numpy as np
import cv2
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import torch.nn.functional as F


# ============================================================
# THIẾT LẬP DỮ LIỆU
# ============================================================
path = "/content/drive/MyDrive/Classroom/Thigiacmaytinh/Phan_lop/"
categories = ['Butterfly', 'Forest']

height, width, N_CHANNELS = 64, 64, 3

data = []
labels = []

# Lấy ảnh và gán nhãn
for k, category in enumerate(categories):
    for f in os.listdir(os.path.join(path, category)):
        img_path = os.path.join(path, category, f)
        img = cv2.imread(img_path)
        if img is None:
            continue
        img = cv2.resize(img, (width, height))
        data.append(img)
        labels.append(k)
```

```
    # Chuyển sang numpy và chuẩn hóa
    data = np.array(data, dtype=np.float32) / 255.0
    labels = np.array(labels)

    # Chia dữ liệu train/test
    trainX, testX, trainY, testY = train_test_split(
        data, labels, test_size=0.2, random_state=42
    )

    # Chuyển sang Tensor PyTorch
    trainX = torch.tensor(trainX.transpose((0, 3, 1, 2)))  # (N, C, H, W)
    testX = torch.tensor(testX.transpose((0, 3, 1, 2)))
    trainY = torch.tensor(trainY, dtype=torch.long)
    testY = torch.tensor(testY, dtype=torch.long)

    # Tạo DataLoader
    BS = 64
    train_dataset = TensorDataset(trainX, trainY)
    test_dataset = TensorDataset(testX, testY)
    train_loader = DataLoader(train_dataset, batch_size=BS, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=BS, shuffle=False)

# ============================================================
# XÂY DỰNG KIẾN TRÚC CNN (theo Keras gốc)
# ============================================================
class CNN_Model(nn.Module):
    def __init__(self, num_classes):
        super(CNN_Model, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=2)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=2)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout1 = nn.Dropout(0.25)
        self.flatten_dim = 32 * 14 * 14  # sau hai lần pooling (64 -> 31 -> 15)
        self.fc1 = nn.Linear(self.flatten_dim, 128)
        self.dropout2 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = self.dropout1(x)
        x = x.view(x.size(0), -1)  # flatten
        x = F.relu(self.fc1(x))
        x = self.dropout2(x)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

# Khởi tạo mô hình
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN_Model(num_classes=len(categories)).to(device)
print(model)
```

```python
# ============================================================
# CẤU HÌNH HUẤN LUYỆN
# ============================================================
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
EPOCHS = 25

# ============================================================
# HUẤN LUYỆN MÔ HÌNH
# ============================================================
for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch [{epoch+1}/{EPOCHS}] - Loss: {running_loss/len(train_loader):.4f}")

print("Training hoàn tất!")
```

```python
# ============================================================
# ĐÁNH GIÁ MÔ HÌNH
# ============================================================
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.numpy())

# Confusion Matrix & Accuracy
cm = confusion_matrix(all_labels, all_preds)
acc = accuracy_score(all_labels, all_preds)
print("Accuracy: %.2f%%" % (acc * 100.0))

plt.imshow(cm, cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.xticks(np.arange(len(categories)), categories)
plt.yticks(np.arange(len(categories)), categories)
for i in range(len(categories)):
    for j in range(len(categories)):
        plt.text(j, i, cm[i, j], ha="center", va="center", color="red")
plt.show()
```

```python
# ============================================================
# THỬ DỰ ĐOÁN ẢNH MỚI
# ============================================================
img_path = "/content/Untitled.jpg"
img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (width, height))
img_tensor = torch.tensor(img.transpose((2, 0, 1)), dtype=torch.float32).unsqueeze(0) / 255.0
img_tensor = img_tensor.to(device)

with torch.no_grad():
    output = model(img_tensor)
    _, predicted = torch.max(output, 1)
    pred_class = categories[predicted.item()]
    confidence = torch.exp(output[0][predicted]).item() * 100

plt.imshow(img)
plt.title(f"{pred_class} ({confidence:.2f}%)")
plt.axis("off")
plt.show()

print(f"Kết quả dự đoán: {pred_class} ({confidence:.2f}%)")
```

# ĐẶC TRƯNG CNN VÀ DÙNG SVM ĐỂ HUẤN LUYỆN

```
# ======================================================
# 1. Chuẩn bị dữ liệu như trong mã gốc
# ======================================================
import numpy as np
import cv2
import os
import random
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

# Đường dẫn dữ liệu
path = "/content/drive/MyDrive/Classroom/Thigiacmaytinh/Phan_lop/"
categories = ['Butterfly', 'Forest']

height, width, N_CHANNELS = 64, 64, 3

# Tải ảnh và gán nhãn
data, labels = [], []
for k, category in enumerate(categories):
    for f in os.listdir(os.path.join(path, category)):
        img_path = os.path.join(path, category, f)
        img = cv2.imread(img_path)
        if img is None:
            continue
        img = cv2.resize(img, (width, height))
        data.append(img)
        labels.append(k)
```

```
# Chuyển thành mảng NumPy
data = np.array(data, dtype="float32") / 255.0
labels = np.array(labels)

# Chia train/test
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2, random_state=42)
trainY_cat = to_categorical(trainY, len(categories))
testY_cat = to_categorical(testY, len(categories))

print(trainX.shape, testX.shape)
```

```
# ======================================================
# 2. Xây CNN như cũ, nhưng chỉ train để trích đặc trưng
# ======================================================
model = Sequential([
    Conv2D(32, (2,2), activation='relu', input_shape=(height, width, N_CHANNELS)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(32, (2,2), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),   # đặc trưng chúng ta sẽ lấy ở đây
    Dropout(0.5),
    Dense(len(categories), activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

# Không train lâu – chỉ vài epoch hoặc dừng sớm để lấy đặc trưng
model.fit(trainX, trainY_cat, epochs=3, batch_size=64, verbose=1)
```

**Hoặc**

```
# Khi xây model, đặt tên cho layer bạn muốn lấy feature
model = Sequential([
    Conv2D(32, (2,2), activation='relu', input_shape=(64,64,3)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu', name='feature_layer'),   # đặt tên rõ ràng
    Dense(2, activation='softmax')
])

# Tạo model trích đặc trưng
feature_extractor = Model(
    inputs=model.input,
    outputs=model.get_layer('feature_layer').output
)
```

```
# ======================================================
# 4. Huấn luyện mô hình SVM
# ======================================================
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix

svm_clf = SVC(kernel='rbf', C=100000, gamma=0.001)  # bạn có thể đổi kernel: 'linear', 'poly', ...
svm_clf.fit(X_train_features, trainY)

# Dự đoán
y_pred = svm_clf.predict(X_test_features)
```

```python
# ======================================================
# 5. Đánh giá
# ======================================================
acc = accuracy_score(testY, y_pred)
print("Accuracy with SVM on CNN features: %.2f%%" % (acc * 100))

cm = confusion_matrix(testY, y_pred)
plt.imshow(cm, cmap='Blues')
plt.title("Confusion Matrix (CNN + SVM)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.xticks(np.arange(len(categories)), categories)
plt.yticks(np.arange(len(categories)), categories)
for i in range(len(categories)):
    for j in range(len(categories)):
        plt.text(j, i, cm[i, j], ha="center", va="center", color="red")
plt.show()
```