ĐẠI HỌC QUỐC GIA TP.HCM
ĐẠI HỌC BÁCH KHOA TP.HCM
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

ĐỒ ÁN TỔNG HỢP - HƯỚNG TRÍ TUỆ NHÂN TẠO

Đề tài

Nhận diện loài chim bằng phương pháp CNN

GVHD: Vũ Văn Tiến
Sinh viên: Nguyễn Thành Đạt - 2111018.

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 3/2024

Mục lục

1 Cơ sở lý thuyết

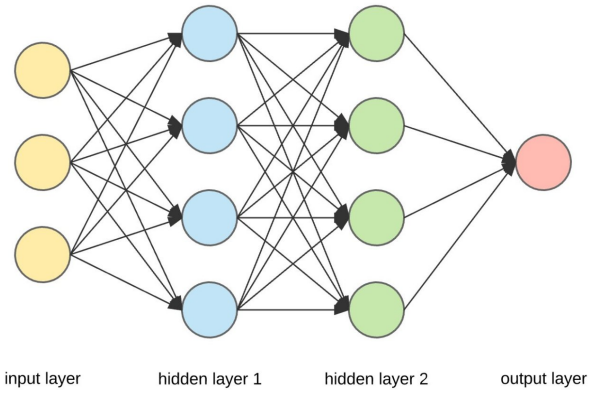
1.1 Mạng lưới Nơ-ron (Neural Networks)

1.1.1 Định nghĩa

Mạng lưới Nơ-ron (Neural Networks - NN) trong học máy (machine learning) là một mô hình toán học dựa trên mạng nơ-ron sinh học. Nó biểu diễn dữ liệu theo tầng, bao gồm mạng lưới các nốt nơ-ron chứa dữ liệu được kết nối với nhau và có thể truyền tín hiệu dữ liệu từ nốt này sang nốt kia để tính toán dữ liệu mới tại các nút.

Mạng Nơ-ron thường sẽ được tổ chức theo tầng (layers), với thông tin đi từ tầng thứ nhất thông qua một hoặc nhiều tầng trung gian (còn gọi là tầng ẩn - hidden layers) để đến lớp cuối cùng, cho ra kết quả bài toán. Do đó, có thể coi mạng lưới nơ-ron giống như một ánh xạ, nối giá trị từ input của tầng đầu tiên đến kết quả output của tầng cuối cùng.

Mạng Nơ-ron được ứng dụng rộng rãi trong trí tuệ nhân tạo, bao gồm nhiều khía cạnh khác nhau như dự đoán mô hình, bài toán nhận diện, bài toán gợi ý,...



Hình 1: Mô phỏng một Neural network

1.1.2 Cách hoạt động

Một kết nối giữa 2 nốt bất kì bao gồm một nốt ở tầng dưới và một nốt ở tầng trên. Mỗi kết nối như thế sẽ có một con số biểu diễn trọng lượng (weights), đại diện cho ảnh hưởng của nốt ở tầng trên đối với nốt ở tầng dưới. Một nốt ở tầng dưới sẽ được kết nối với mọi nốt ở tầng trên, hay còn được gọi là Full-connected/Dense layer. Bên cạnh đó, mỗi tầng đều sẽ có một bias, có thể coi đây là giá trị ảnh hưởng của toàn bộ tầng trên đối với mỗi nốt ở tầng dưới nó. Khi đó, giá trị của một nốt ở tầng dưới sẽ là tổng trọng lượng của mỗi nốt ở tầng trên đối với nó và giá trị bias. Giá trị đó được định nghĩa như sau:

$$Y = (\sum_{i=0}^{n} w_i x_i) + b \tag{1}$$

Trong đó:

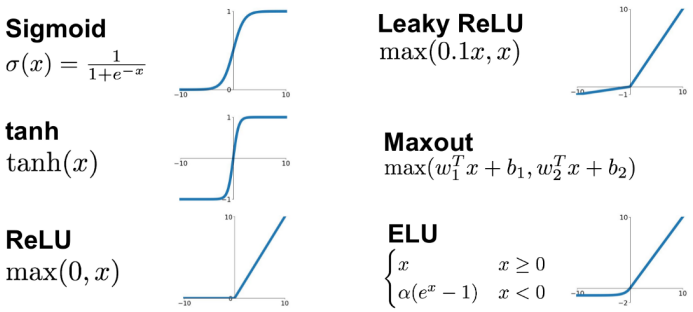
- w là trọng lượng của mỗi kết nốt đến một nơ-ron.

- x là giá trị của nơ-ron ở tầng trên kết nối với nơ-ron hiện tại.
- b là giá trị bias của tầng trên
- n là số lượng kết nối.
- Y là kết quả của nơ-ron đang được kết nối.

Phương trình ở trên là tổng trọng lượng (weight sum). Mỗi nơ-ron ở tầng dưới sẽ được tính toán bằng phương trình (1) và tiếp tục truyền dữ liệu đến cho những nốt ở tầng dưới. Những giá trị này, bao gồm weight và bias, sẽ được mô hình tối ưu, thay đổi sao cho trả đúng kết quả output nhất mong đợi từ input. Đó là quá trình học (training) của mô hình.

Tuy nhiên, phương trình (1) là một phương trình tuyến tính, do đó để tăng độ phức tạp và sự phi tuyết tính vào mô hình, **hàm kích hoạt** được đưa vào nhằm giúp mạng lưới có thể học hỏi những mô hình và quan hệ phức tạp. Một số hàm kích hoạt thông dụng như ReLU (Rectified Linear Unit), Sigmoid, Tanh,... Phương trình (1) sẽ được viết lại với $F(x)$ là hàm kích hoạt:

$$Y = F((\sum_{i=0}^{n} w_i x_i) + b) \tag{2}$$



Hình 2: Một số activation function thông dụng

Ban đầu, mạng lưới sẽ bắt đầu với hàm kích hoạt được định nghĩa trước nhưng với trọng lượng và bias ngẫu nhiên. Khi cho máy học bằng việc tiếp nhận thông tin, nó sẽ chỉnh sửa trọng lượng và bias đồng thời thay đổi mạng lưới dựa theo một kĩ thuật gọi là Backpropagation. Một khi quá trình học đã hoàn tất, ta sẽ đánh giá mô hình thông qua việc quan sát kết quả đầu ra của nó tại tầng cuối cùng.

1.1.3 Lan truyền ngược - Backpropagation

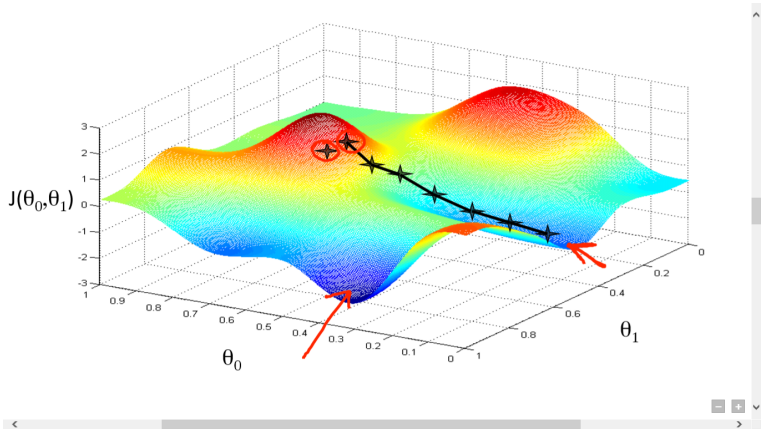
Mục tiêu của Neural-networks là dự đoán chính xác kết quả đầu vào, bằng việc thay đổi các thông số weight và bias như đã đề cập. Đó chính là lan truyền ngược (backpropagation). Lan truyền ngược là giải thuật cốt lõi cho việc training của neural-network. Trước khi đi vào nó, ta cần phải biết được **hàm mất mát** (cost/loss function)

1.1.3.1 Hàm mất mát: Hàm mất mát là hàm so sánh sự khác biệt giữa giá trị đầu ra của mô hình với giá trị cần dự đoán. Mục đích của nó là để đánh giá giá trị đầu ra của neural-network, từ đó cho biết mạng lưới mô hình dữ liệu training tốt như thế nào. Do đó, công việc thực sự của neural-network chính là tối thiểu hàm mất mát này, cũng có nghĩa là dự đoán đúng giá trị mong đợi. Một số hàm mất mát thường thấy là Mean Squared Error, Mean Absolute Error,...

1.1.3.2 Gradient Descent: Gradient Descent là thuật toán dùng cho việc tìm các tham số tối ưu (weights và bias) cho mạng lưới. Mục đích của Gradient Descent chính là để tìm cực tiểu địa phương (local minimun) cho hàm mất mát. Nó làm điều này bằng việc di chuyển theo hướng ngược lại với giá trị Gradient của hàm mục tiêu. Gradient được định nghĩa như sau: Gradient của hàm $f(v)$ với $v = (v_1, v_2, ..., v_n)$ là một vector

$$\nabla f(v) = \begin{bmatrix} \frac{\partial f}{\partial v_1} \\ \frac{\partial f}{\partial v_2} \\ \vdots \\ \frac{\partial f}{\partial v_n} \end{bmatrix} \tag{3}$$

Trong đó, $\frac{\partial f}{\partial v_i}$ là đạo hàm riêng phần của $f(v)$ với $v_i, i = 1, ..n$. Gradient cho biết độ dốc của một điểm thuộc hàm mục tiêu trong không gian n chiều. Có thể hiểu rằng, Gradient cho biết hướng đi sao cho hàm mục tiêu tăng dần giá trị, nên ngược lại đó là hướng đi giảm dần giá trị của hàm mục tiêu, cũng chính là mục đích của giải thuật Gradient descent. Trong neural-network, Gradient descent là giải thuật được Backpropagation sử dụng nhằm cập nhật các tham số của mô hình.



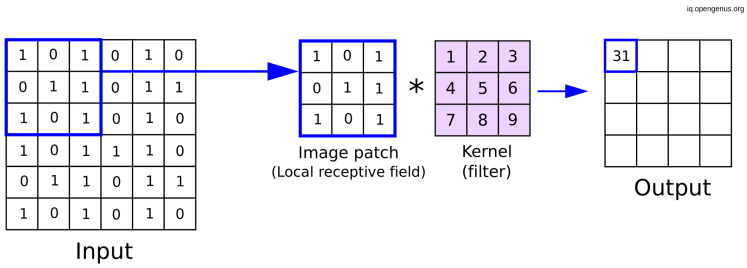
Hình 3: Trực quang hóa của Gradient descent

1.1.3.3 Optimizer: Optimizer là hàm thực hiện giải thuật Backpropagation như ở trên. Một số hàm optimizer bao gồm Gradient descent, Stochastic Gradient descent, Momentum, Adam, ...

1.2 Mạng nơ-ron tích chập - Convolutional Neural Network (CNN)

1.2.1 Tích chập - Convolution

Tích chập là phép toán lấy từng ma trận con của ma trận được tích chập nhân với ma trận tích chập (hay còn được gọi là kernel, filter hoặc sliding windows). Các ma trận con này có cùng kích thước với sliding windows và sẽ trượt dọc 1 ô theo từng hàng và từng cột của ma trận gốc. Kết quả là một ma trận tích chập được sinh ra từ việc nhận ma trận filter với ma trận gốc.

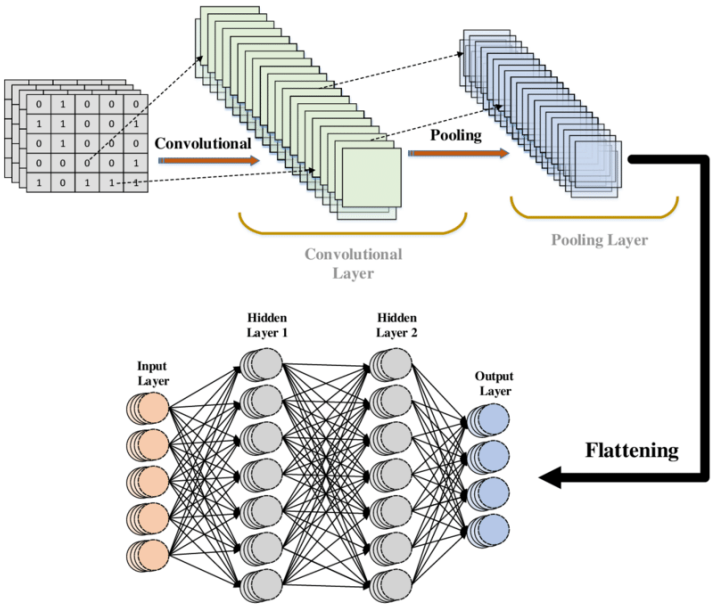


Hình 4: Phép tích chập hai ma trận

1.2.2 Mạng tích chập - CNN

Bên cạnh các tầng đã được nhắc đến trong phần Neural-network, mạng CNN có thêm các lớp tích chập (Convolutional layers) chồng lên nhau. Ở Convolutional layer, giá trị của neural không phải là kết quả tính toán weights sum như ở phương trình (1), mà thay vào đó nó sẽ là kết quả từ việc thực hiện phép tích chập với một ma trận filter. Ma trận được tích chập thường được gọi là feature map, một convolutional layer sẽ sử dụng nhiều filter khác nhau lên feature map, tạo nên nhiều kết quả. Do vậy, ngay sau tầng tích chập thường sẽ có một tầng pooling để chắc lọc ra các thông tin hữu ích hơn, loại bỏ thông tin gây nhiễu.

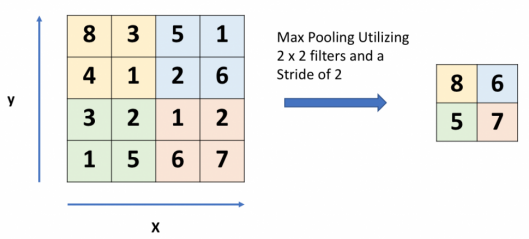
Mục đích của convolutional layers là để lọc ra các dấu hiệu khác nhau với mỗi filter. Trong xử lí ảnh thì nó có nghĩa là làm mờ, làm rõ, hoặc lấy nét cạnh.



Hình 5: Mô hình một CNN

1.2.3 Pooling layers

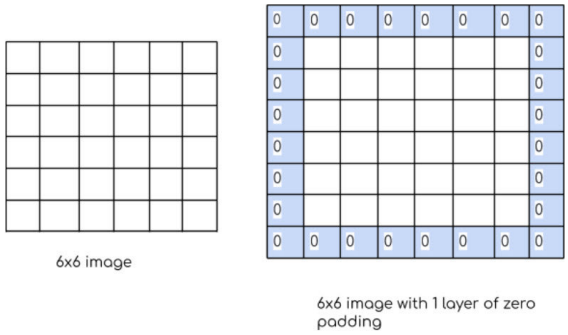
Lớp pooling thường được sử dụng ngay sau lớp convolution để đơn giản hóa thông tin đầu ra, giảm bớt số lượng neuron. Một thủ tục pooling phổ biến là max-pooling - chọn ra giá trị lớn nhất trong mỗi ma trận 2x2 của ma trận đầu vào.



Hình 6: Lớp MaxPooling

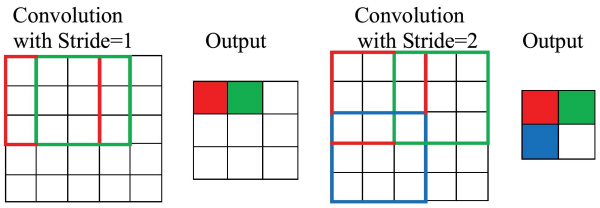
1.2.4 Padding và Strides

Dễ thấy việc convolution làm giảm kích thước của ma trận gốc, bởi một số ô ở cạnh không nằm ở trung tâm của ma trận filter .Padding là hành động thêm các hàng, các cột với giá trị 0 vào ma trận gốc nhằm hạn chế hiện tượng này.



Hình 7: Kĩ thuật Padding

Thông thường, ma trận filter sẽ trượt 1 ô cho mỗi lần tính toán. Stride là thông số đại diện cho số ô mỗi lần trượt của filter.



Hình 8: Kĩ thuật Stride

1.2.5 Ưu điểm của CNN

CNN sử dụng trường tiếp nhận cục bộ (local receptive field), giúp nhận biết các dấu hiệu (pattern, feature) trong cục bộ từng bộ phận, thay vì toàn cảnh như Fully-connected layer. Do đó, CNN có tính bất biến khi đối tượng được chiếu từ các góc độ khác nhau như dịch chuyển (translation), rotation (xoay chuyển), scaling (co giãn). Tính kết hợp cục bộ cho ta nhiều cấp độ biểu diễn thông tin, trừu tượng hơn thông qua các lớp convolution.

CNN rất hiệu quả trong việc xử lí ảnh, với độ chính xác rất cao, do đó thường được sử dụng trong việc nhận dạng ảnh.

2 Giải quyết vấn đề

2.1 Đặt vấn đề

Trong các vườn quốc gia, nhân viên kiểm lâm sẽ thường xuyên chụp ảnh của các loài chim nhằm theo dõi tình hình quần thể cũng như đánh giá mức độ nguy hiểm của chúng. Việc có thể phân loại được các loài chim khác nhau thông qua ảnh chụp sẽ giúp ích rất nhiều trong việc

quản lí tài nguyên thiên nhiên, bảo vệ đa dạng sinh học. Do đó, nhóm đã chọn nhận diện loài chim làm đề tài cho đồ án này.

Input của bài toán sẽ là ảnh chụp một con chim, với output trả về chủng loại của loài chim đó.

2.2 Phương pháp hiện thực

Bài toán thuộc nhóm vấn đề nhận dạng ảnh. Trong nhóm bài toán này có nhiều kĩ thuật khac nhau được công bố như SVM, HMM, AGES, GMM .Nhóm lựa chọn thực hiện việc xây dựng mô hình theo phương pháp CNN nhờ tính hiệu quả và các ngôn ngữ, framework hỗ trợ.

Công cụ sử dụng:

- Ngôn ngữ lập trình: Python 3.11
- Framework: Tensorflow 2.16.1

Nhóm sẽ sử dụng accuracy (tỉ lệ đoán đúng output của mô hình) làm metric chính để đánh giá mô hình.

2.3 Nguồn dữ liệu (dataset)

Dữ liệu được lấy từ đường link https://www.kaggle.com/datasets/gpiosenka/100-bird-species. Nguồn dữ liệu bao gồm hơn 525 loài chim, 84635 ảnh training, 2625 ảnh cho validation (khi train) và 2625 ảnh cho test (sử dụng để làm prediction sau này). Mỗi ảnh màu có kích thước 224x224x3 với định dạng jpg. Mỗi ảnh chỉ chứa một con chim và nó sẽ chiếm ít nhất 50% lượng pixel của tấm ảnh.

Hình 9: Một số hình ảnh từ dataset

2.4 Xây dựng mô hình

2.4.1 Định nghĩa mô hình

```
model = Sequential([
    layers.Resizing(112, 112, input_shape= (224, 224, 3) ),
    layers.Rescaling(1./255),
    # Yoinking from tutorial, not optimized for accuracy
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

2.4.2 Compile mô hình

```
# Compile the model
model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])
```

2.4.3 Train mô hình

```
# Train the model using train_ds
history = model.fit(
    train_ds,
    validation_data = test_ds,
    epochs = 10
)
```

2.4.4 Kết quả training

```
Epoch 1/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 255s 95ms/step - accuracy: 0.0530 - loss: 5.4703 - val_accuracy: 0.3265 - val_loss: 3.1780
Epoch 2/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 289s 109ms/step - accuracy: 0.3175 - loss: 3.2482 - val_accuracy: 0.4648 - val_loss: 2.3965
Epoch 3/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 305s 115ms/step - accuracy: 0.4875 - loss: 2.2755 - val_accuracy: 0.5158 - val_loss: 2.1904
Epoch 4/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 287s 108ms/step - accuracy: 0.5989 - loss: 1.7042 - val_accuracy: 0.5310 - val_loss: 2.1924
Epoch 5/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 277s 105ms/step - accuracy: 0.6865 - loss: 1.2800 - val_accuracy: 0.5299 - val_loss: 2.4916
Epoch 6/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 267s 101ms/step - accuracy: 0.7604 - loss: 0.9382 - val_accuracy: 0.5208 - val_loss: 2.8367
Epoch 7/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 286s 108ms/step - accuracy: 0.8157 - loss: 0.6858 - val_accuracy: 0.5139 - val_loss: 3.3299
Epoch 8/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 285s 108ms/step - accuracy: 0.8546 - loss: 0.5110 - val_accuracy: 0.5128 - val_loss: 3.7232
Epoch 9/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 271s 102ms/step - accuracy: 0.8837 - loss: 0.3999 - val_accuracy: 0.4937 - val_loss: 4.1229
Epoch 10/10
2645/2645 ━━━━━━━━━━━━━━━━━━━━ 273s 103ms/step - accuracy: 0.9014 - loss: 0.3328 - val_accuracy: 0.5093 - val_loss: 4.5191
```

Hình 10: Kết quả training