Chuẩn bị dữ liệu

Lưu trữ và truy cập dữ liệu từ Google Drive (vì dùng Google Colab)
Mô hình sẽ được xây dựng để phân loại 3 loại động vật khác nhau bao gồm: chó, mèo, nhện dựa trên 3 tệp dữ liệu có sẵn cho chúng

Tiền xử lý dữ liệu

Dữ liệu có nhãn của mèo: 1668 ảnh, 4836 ảnh của chó và 4821 ảnh của nhện
Có sự chênh lệch giữa kích cỡ dữ liệu có nhãn của mèo so với 2 dạng còn lại là chó và nhện
Đưa tất cả ảnh từ tập dữ liệu về dạng 128x128

Mô hình hóa

Cấu trúc mạng tích chập gồm:

- Đầu vào: ảnh màu 3 kênh (RGB) có kích thước 128x128
- Gồm 3 tầng tích chập, 3 tầng gộp (pooling), 2 tầng truyền thẳng (fully connected), và 1 lớp Dropout để tránh tình trạng quá khớp (overfitting)
- Đầu ra: xác suất cho mỗi lớp (ở đây là 3 lớp cho chó, mèo và nhện)

Chi tiết:

Tầng khởi tạo:

Lớp tích chập đầu tiên:

Đầu vào: 3 kênh màu (RGB)
Đầu ra 16 feature map
Kích thước kernel là 3x3 với padding là 1 để giữ kích thước không đổi
Kết quả đầu ra sẽ là [16, 128, 128]
Áp dụng “Max Pooling” 2x2 -> giảm đầu ra xuống còn [16, 64, 64]

Lớp tích chập thứ hai:

Đầu vào: [16, 64, 64] -> [32, 64, 64]
Áp dụng MaxPool -> [32, 32, 32]

Lớp tích chập thứ ba:

Đầu vào: [32, 32, 32] -> đầu ra: [64, 32, 32]
Áp dụng MaxPool -> [64, 16, 16]

Lớp Max Pooling:

Gộp 2x2 pixel thành 1 -> giúp giảm 50% chiều rộng và chiều cao
Giúp giảm số lượng tham số, tăng tốc, giữ đặc trưng chính của dữ liệu dạng ảnh

Lớp kết nối đầy đủ (Fully Connected):

Sau khi dữ liệu đi qua 3 tầng mạng tích chập cùng với gộp, kết quả đầu ra có kích thước [64, 16, 16]

Sau đó, ta trải thẳng chúng thành 1 véc tơ, sau đó đi qua 2 tầng kết nối đầy đủ

Tầng kết nối đầy đủ thứ nhất: Từ 16384 -> 128 (biến ảnh thành véc tơ đặc trưng nhỏ hơn)

Tầng kết nối đầy đủ thứ hai: Từ 128 -> 3 (tương ứng với 3 lớp phân loại)

Lớp Dropout:

Ngẫu nhiên tắt 30% mạng nơ ron trong lớp kết nối đầy đủ thứ nhất trong quá trình huấn luyện

=> Nhằm tránh mô hình trở nên quá khớp (overfitting)

Hàm forward():

Kết quả cuối cùng là 3 giá trị (logits) tương ứng với từng lớp

Nếu ta dùng “nn.CrossEntropyLoss()” thì không cần sử dụng hàm softmax ở đây (vì bản thân nó đã có sẵn)

Huấn luyện / Kiểm thử mô hình

Trong mỗi epoch (ở đây là 10 epochs):

- Lấy dữ liệu ảnh + nhãn
- Loại bỏ gradient ở phần tính toán trước
- Dự đoán nhãn từ ảnh qua mô hình
- Tính toán độ sai lệch (loss) giữa dự đoán và nhãn thật
- Tính đạo hàm cho gradient của loss theo các trọng số (backward)
- Cập nhật trọng số để giảm loss (qua optimizer Adam hoặc AdamW với tỉ lệ học ban đầu là 0.001)

Sau mỗi epoch (hoặc cuối cùng), ta dùng dữ liệu mà mô hình chưa từng được nhìn thấy (test dataset) để kiểm tra độ chính xác:

- Không dùng bộ tối ưu hóa (optimizer.step()), không cập nhật trọng số
- Không cần tính toán gradient => dùng torch.nograd() để tiết kiệm bộ nhớ