

Báo cáo về Model phân tích số viết tay trong tập dữ liệu MNIST bằng LeNet-5

Nguyễn Trường Giang

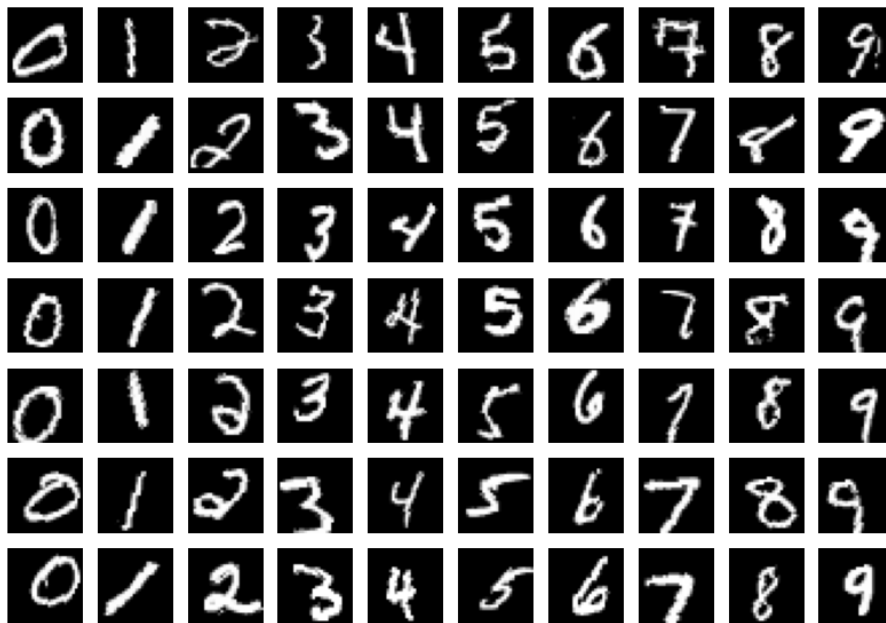
Ngày 8 tháng 2 năm 2025

1 Tập dataset:

1.1 Các Label

- Tập dữ liệu MNIST có 10 label bao gồm các số 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Input của mỗi mẫu dữ liệu là một ảnh 32×32 , mỗi pixel có giá trị từ 0 \rightarrow 255 mô tả ảnh của một số viết tay.

1.2 Visualize mẫu dữ liệu:



Hình 1: Visualization dataset

1.3 Data augmentation:

- Có **data augmentation** bằng cách sử dụng `transforms.Compose()` bao gồm:

- Góc xoay ảnh: 10° .
- Tịnh tiến trên dưới trái phải: 10%.
- Tỷ lệ ảnh so với ảnh gốc: 10%.
- Làm méo ảnh (shear) đi: 10° .

2 Kiến trúc mô hình:

2.1 Các layer

- Mô hình có tổng cộng 7 layer chính, bao gồm:

1. Convolution 1.
2. Pooling (Average) 1.
3. Convolution 2.
4. Pooling (Average) 2.
5. Convolution 3.
6. Fully connected 1.
7. Fully connected 2.

- Ngoài ra còn sử dụng các hàm kích hoạt [Tanh\(\)](#) và sử dụng [Softmax\(\)](#) cho layer cuối cùng, và sử dụng [Flatten\(\)](#) để trải phẳng hình ảnh ra trước khi vào lớp **fully connected**.

2.2 Cách tính đầu vào, đầu ra từng layer (quan trọng).

- Ban đầu mỗi batch có kích thước $(32 \times 1 \times 28 \times 28)$.

- Sau khi qua lớp **Convolution 1** với ($in_channels = 1, out_channels = 9, kernel_size = 3, padding = 1$)
 - Chiều cao $H = \frac{H - kernel_size + 2 \times padding}{stride} + 1 = \frac{28 - 3 + 2 \times 1}{1} + 1 = 28$
 - Chiều rộng $W = \frac{W - kernel_size + 2 \times padding}{stride} + 1 = \frac{28 - 3 + 2 \times 1}{1} + 1 = 28$
- Khi qua lớp **Pooling 1** ($kernel_size = 2, stride = 2$)
 - Chiều cao $H = \frac{H - kernel_size}{stride} + 1 = \frac{28 - 2}{2} + 1 = 14$
 - Chiều rộng $W = \frac{W - kernel_size}{stride} + 1 = \frac{28 - 2}{2} + 1 = 14$
 - Số channels: 9
- Sau khi qua lớp **Convolution 2** với ($in_channels = 9, out_channels = 36, kernel_size = 5$)
 - Chiều cao $H = \frac{14 - 5}{1} + 1 = 10$
 - Chiều rộng $W = \frac{14 - 5}{1} + 1 = 10$
 - Số channels: 36
 - (Không có *padding* thì *padding* sẽ mặc định bằng 0).
- Khi qua lớp **Pooling 2** ($kernel_size = 2, stride = 2$)
 - Chiều cao $H = \frac{10 - 2}{2} + 1 = 5$
 - Chiều rộng $W = \frac{10 - 2}{2} + 1 = 5$
- Khi đi qua lớp **Convolution 3** với ($in_channels = 36, out_channels = 72, kernel_size = 2$)
 - Chiều cao $H = \frac{5 - 2}{1} + 1 = 4$
 - Chiều rộng $W = \frac{5 - 2}{1} + 1 = 4$
 - Số channels: 72

- Sau đó dữ liệu sẽ được tính qua hàm [Flatten\(\)](#), khi đó dữ liệu sẽ đổi *shape* từ $(32, 72, 4, 4)$ thành *shape*: $(32, 72 \times 4 \times 4)$ tức là $(32, 1152)$.

- Khi qua **Fully connected 1** sẽ đổi *shape* thành $(32, 96)$.
- Khi qua **Fully connected 2** sẽ đổi *shape* thành $(32, 10)$ (10 tương ứng với số lượng labels của dataset).

2.3 Về Early stopping:

- **Early stopping** trong model được cài *patience* = 100, tức là sau khi chạy 100 epoch liên tiếp mà vẫn không thể cải thiện **validation loss** thì sẽ dừng train model (thực tế là không cần thiết phải lớn như vậy, chỉ cần khoảng *patience* = 20 là quá đủ rồi).
- Còn đối với *delta* = 0 để có thể "vắt kiệt" từng *loss* một (thực tế cũng không cần thiết, nên để *delta* = 0.0001 hoặc loanh quanh đó).

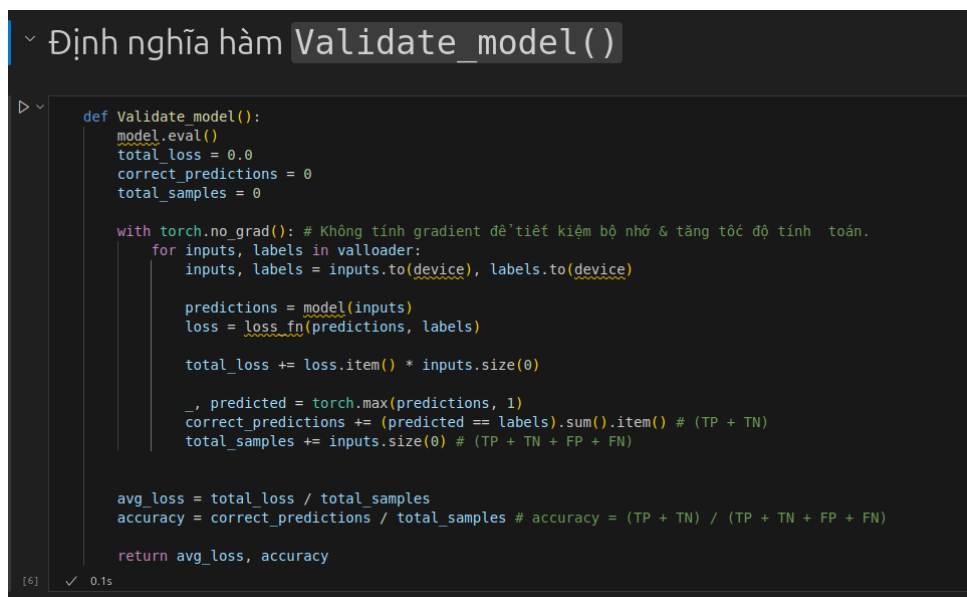
2.4 Về Learning rate schedule:

- **Learning rate schedule** trong model sử dụng `torch.optim.lr_scheduler.ReduceLROnPlateau()` với các tham số là:

- *optimizer* của *Adam*.
- *patience* = 5 tức là nếu mô hình không thể tối ưu **validation loss** thêm được sau 5 epoch thì **learning rate** sẽ giảm.
- *mode* = 'min' có nghĩa là chữ "tối ưu" được nhắc ở trên là **validation loss** **Không thể giảm** được nữa.
- *factor* = 0.1 là mỗi lần **learning rate** giảm, nó sẽ chỉ còn lại tỉ lệ 0.1 so với giá trị ban đầu.

3 Kết quả thực nghiệm:

3.1 Độ đo dùng để đánh giá và ý nghĩa:



```
~ Định nghĩa hàm Validate_model()

def Validate_model():
    model.eval()
    total_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    with torch.no_grad(): # Không tính gradient để tiết kiệm bộ nhớ & tăng tốc độ tính toán.
        for inputs, labels in valloader:
            inputs, labels = inputs.to(device), labels.to(device)

            predictions = model(inputs)
            loss = loss_fn(predictions, labels)

            total_loss += loss.item() * inputs.size(0)

            _, predicted = torch.max(predictions, 1)
            correct_predictions += (predicted == labels).sum().item() # (TP + TN)
            total_samples += inputs.size(0) # (TP + TN + FP + FN)

    avg_loss = total_loss / total_samples
    accuracy = correct_predictions / total_samples # accuracy = (TP + TN) / (TP + TN + FP + FN)

    return avg_loss, accuracy

[6] ✓ 0.1s
```

Hình 2: Validation function

- Độ đo dùng để đánh giá là: Accuracy. Lý do là vì bộ dữ liệu **MNIST** được cho là cân bằng cho mỗi label. Chi tiết hơn thì khi xét trên tập dữ liệu **testset** đối với mô hình đó, mỗi label trong số 10 labels đều có sấp xỉ 1000 samples trong tổng số 10000 samples của **testset** (với sai số chỉ khoảng ± 30 samples).

3.2 Accuracy, confusion matrix, classification report của mô hình.

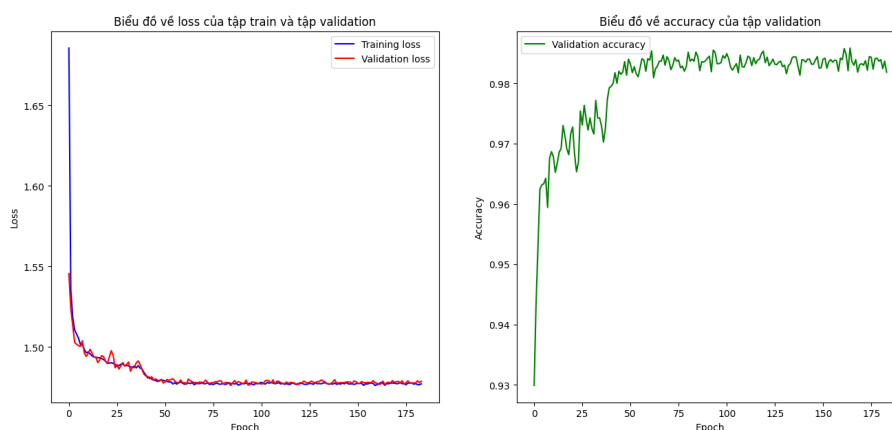
```
Accuracy on test set: 98.41%
Confusion matrix:
[[ 970   0   1   0   0   0   3   1   2   3]
 [   0 1122   4   3   1   0   1   2   2   0]
 [   1   4 1019   3   1   0   0   3   1   0]
 [   0   1   1  996   0   5   0   2   5   0]
 [   0   1   3   0  966   0   1   2   1   8]
 [   0   0   0   7   0  877   2   1   3   2]
 [   1   3   2   0   3   3  944   0   2   0]
 [   0   3  11   1   0   0  1011   0   2]
 [   2   0   4   2   2   3   0   2  956   3]
 [   3   0   0   4   9   2   0   4   7  980]]
Classification report:
              precision    recall  f1-score   support

 number 0      0.99      0.99      0.99       980
 number 1      0.99      0.99      0.99      1135
 number 2      0.98      0.99      0.98      1032
 number 3      0.98      0.99      0.98      1010
 number 4      0.98      0.98      0.98       982
 number 5      0.99      0.98      0.98       892
 number 6      0.99      0.99      0.99       958
 number 7      0.98      0.98      0.98      1028
 number 8      0.98      0.98      0.98       974
 number 9      0.98      0.97      0.98      1009

 accuracy              0.98      10000
 macro avg              0.98      0.98      0.98      10000
 weighted avg           0.98      0.98      0.98      10000
```

Hình 3: Độ chính xác của mô hình sau khi training

4 Kết luận, cần cải thiện gì, có overfit/underfit k



Hình 4: Biểu đồ về loss và accuracy trong khi mô hình train

- Dựa vào biểu đồ trên có thể biết được mô hình đã fitting từ đâu đó epoch thứ 60 rồi, nhưng vì "quá tay" nên mô hình trên vẫn tiếp tục train, có lẽ là có "một chút" **overfit**. Lý do "một chút" là vì mô hình sử dụng thêm cả *validation function* và sau khi train xong thì accuracy trên tập validation và trên tập test không khác nhau là bao nhiêu.