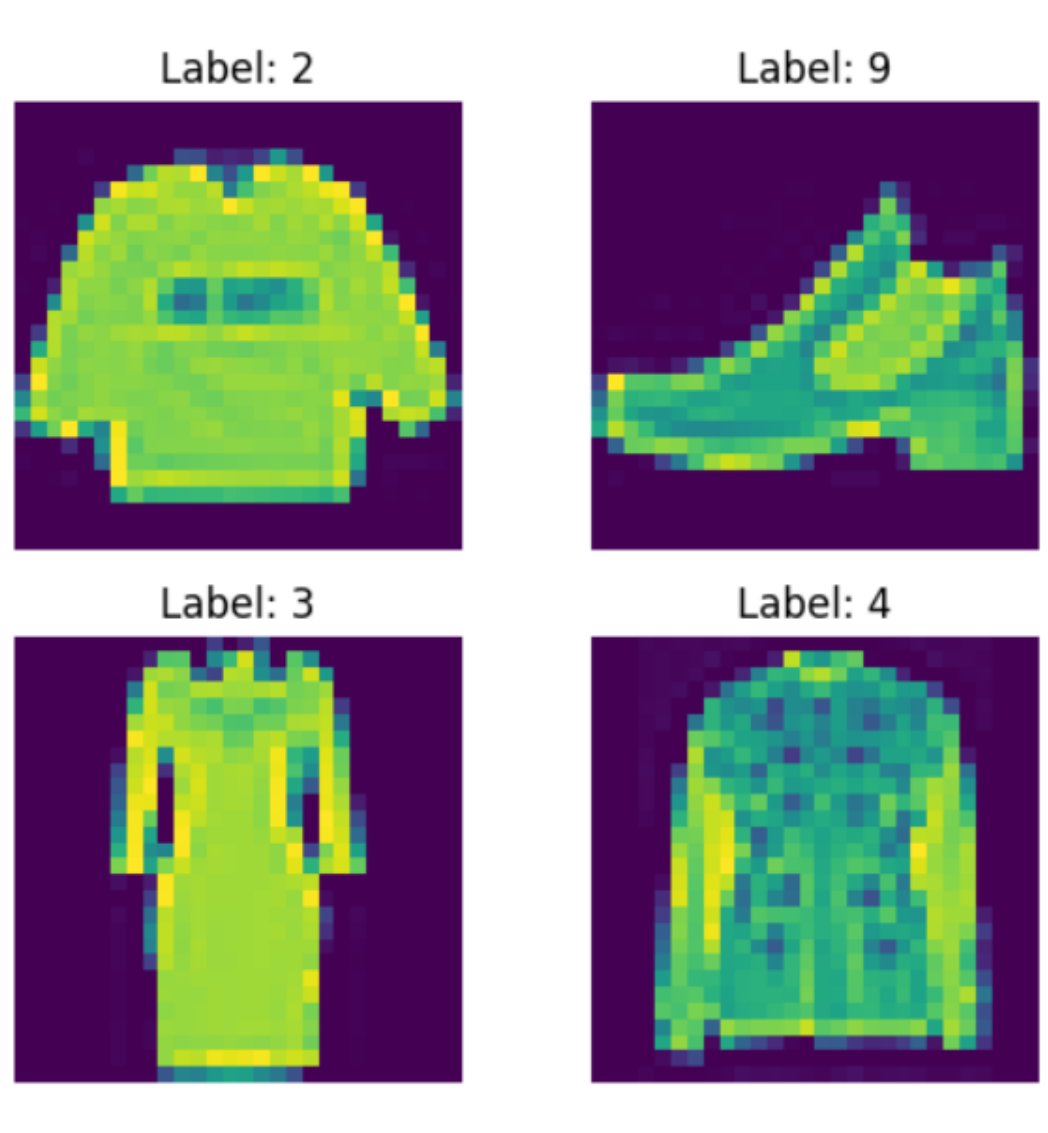
9.0 OPTIMIZING THE NEURAL NETWORK

24 August 2025 02:48 AM

Avinash Yadav



fashion MNIST Dataset Sampel

← data consists of cloths and apparels.

→ Problem of multi-class classification.

STAGE 1:-

Trained the model on CPU DONE ✓
↓
5000 images out of 60,000 image → Accuracy → ≈ 82%

STAGE 2:-

Trained the model on GPU DONE ✓
↓
full dataset 60,000 images → Accuracy → ≈ 88%

⊕ PROBLEM NOT ACKNOWLEDGED IN STAGE -2 :

What we did:-

- Data loaded
- Images loaded
- Train Test split performed
- Making of TrainLoader and TestLoader class
- Defined Neural Network architecture
- Model created
- Shifted the model on GPU
- Trained the model for 100 epochs
- Performed Evaluation

THE PROBLEM:

The test accuracy during evaluation for test-data is ≈ 89%

```python
# EVALUATION CODE FOR TEST DATA USING `test_loader`:

total = 0
correct = 0

with torch.no_grad():

  for batch_features, batch_labels in test_loader:

    batch_features, batch_labels = batch_features.to(device), batch_labels.to(device)

    outputs = model(batch_features)

    _, predicted = torch.max(outputs, 1)

    total = total + batch_labels.shape[0]

    correct = correct + (predicted == batch_labels).sum().item()

print(correct/total)
```

Python

```
0.8869166666666667
```

The test accuracy during evaluation for train-data is ≈ 98%

```python
# EVALUATION CODE FOR TRAIN DATA USING `train_loader`:
total = 0
correct = 0

with torch.no_grad():

  for batch_features, batch_labels in train_loader:

    batch_features, batch_labels = batch_features.to(device), batch_labels.to(device)

    outputs = model(batch_features)

    _, predicted = torch.max(outputs, 1)

    total = total + batch_labels.shape[0]

    correct = correct + (predicted == batch_labels).sum().item()

print(correct/total)
```

Python

```
0.9795625
```

So the model we trained has accuracy on

test data ≈ 89% &
train data ≈ 98%

Therefore we are getting almost of 10% of accuracy difference between training data & testing data.
↓
This clearly means our model is overfitted.
↓
giving good result on training data but not on the test data.

Therefore the next move would be to reduce the overfitting because the performance won't be good in new unseen situation.

⊕ Various solution to reduce the overfitting:-

1.) Adding more data:

The more data is feeded to model, the more biases will reduces and the chances of overfitting will also decrease.

(As of now we are using all the 60,000 images, so we won't be able to use this technique ✗)

2.) Reducing the complexity of NN architecture:

sometimes overfitting also occurs because of complex model.

(This is also not applicable for our model as our architecture is simple with 1 input layer, 2 hidden layer and 1 output layer)

3.) Regularization:

Here we add a penality term in loss fxn. Now our model along with loss fxn. also tries to minimize penality. And during this process overfitting reduces.

(In DL. we prefer more of L2 regularization. so we can do it)

4.) Dropouts:

Here, during training, we randomly turn off some neurons from some layers. Since this is a random behaviour for every forward propagation, so model's overfitting is reduced.

(we are apply this technique to our model)

5.) Data Augmentation:

We perform some sort of transformation on our images like tilting, rotating, flipping, etc so for each of the image we'll get different varition. And when we'll feed this data to model, the chances of overfitting can reduce.

(The data augmentation approach works well for CNN architecture but as of now we are using ANN, so we'll not use it.)

6.) Batch Normalization:

In general it is performed to stabilize the training but one of the side effect of normalization is that we get impact of regularization as well.

(So we'll use this well)

7.) Early Stopping:

Here suppose we are training our model for 100 epochs and we see that if after 58 epochs the loss is not improving, then we can stop our training after that.

In a nutshell to optimize the model we'll use (as of now):

1. Regularization
2. Dropouts
3. Batch Normalization