Süni İntellektin əsasları

Sərbəst iş

Mövzu : CNN istifadə edərək əl yazısı ilə yazılmış rəqəmlərin tanınması.

Tələbə: Vəliyev Fəqan

Qrup: M665a4

Giriş

Bu layihənin məqsədi konvolyusion neyron şəbəkə (CNN — Convolutional Neural Network) istifadə edərək əl yazısı ilə yazılmış rəqəmləri (0–9) tanımaqdır. Dataset kimi MNIST istifadə olunub — 60,000 təlim və 10,000 test nümunəsi.

Məlumat toplusu və preprocessing

Bu layihədə məlumat toplusu kimi MNIST datasetindən istifadə olunur. Dataset keras.datasets.mnist kitabxanası vasitəsilə yüklənir və mnist.load_data() funksiyası ilə əldə edilir. Hər bir şəkil 28×28 ölçülü grayscale formatındadır, yəni bir kanallı boz rəngli şəkillərdən ibarətdir.

Məlumatların emalı mərhələsində (preprocessing) şəkillər əvvəlcə NumPy massivlərindən PyTorch tensorlarına çevrilir. Daha sonra unsqueeze(1) funksiyası vasitəsilə kanalı ifadə edən əlavə ölçü (channel dimension) artırılır. Şəkillərin piksel dəyərləri 255-ə bölünərək 0–1 diapazonuna gətirilir, bu da neyron şəbəkənin öyrənmə prosesini sabitləşdirir.

Label-lər (yəni rəqəmləri göstərən etiketlər) long tipinə çevrilir ki, bu da PyTorch modellərinin itki funksiyaları ilə uyğunluq yaratsın.

Sonda verilənlər TensorDataset obyektinə yığılır və DataLoader vasitəsilə modellə işləməyə hazır vəziyyətə gətirilir. Təlim prosesi üçün batch_size = 64 təyin edilir və shuffle=True parametri ilə verilənlər hər epochda qarışdırılır.

Kod parçası :

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = torch.from_numpy(train_images).float().unsqueeze(1) / 255.0
train_labels = torch.from_numpy(train_labels).long()

test_images = torch.from_numpy(test_images).float().unsqueeze(1) / 255.0
test_labels = torch.from_numpy(test_labels).long()

train_dataset = TensorDataset(train_images, train_labels)
test_dataset = TensorDataset(test_images, test_labels)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

Modelin qurulması

Model nn.Sequential ilə qurulub. Əsas bloklar belədir

Convolutional Blok 1

- Conv2d (1 → 32)
- ReLU
- BatchNorm2d

Convolutional Blok 2

- Conv2d (32 → 32)
- ReLU
- BatchNorm2d

Convolutional Block 3

- Conv2d (32 → 32)
- ReLU
- MaxPool2d
- BatchNorm2d
- Dropout

Convolutional Block 4

- Conv2d (32 → 64)
- ReLU
- MaxPool2d
- BatchNorm2d

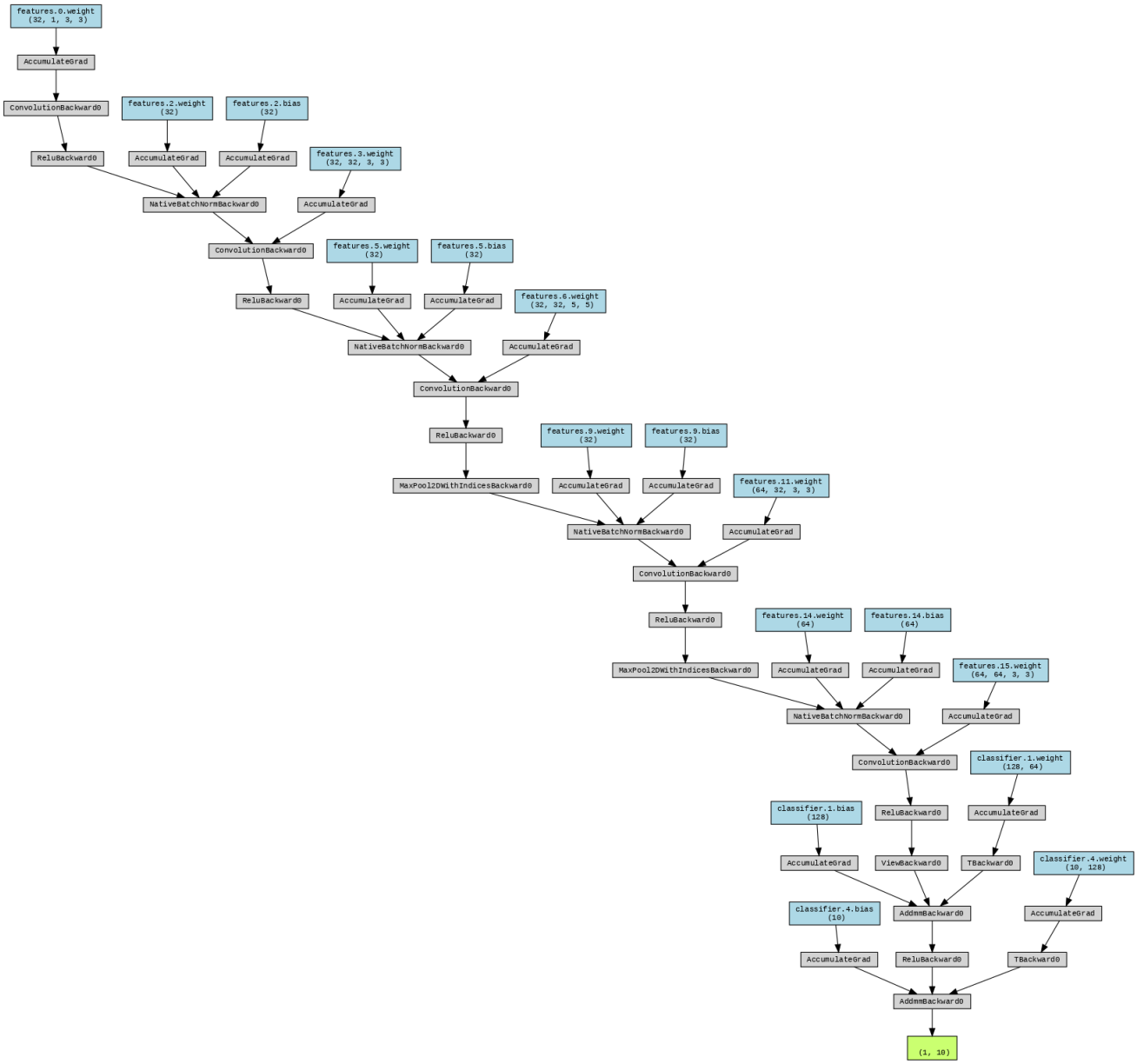
Convolutional Block 5

- Conv2d (64 → 64)
- ReLU
- Dropout

Fully Connected (Classifier) Block

- Flatten
- Linear (64 → 128)
- ReLU
- Dropout
- Linear (128 → 10, MNIST datasetində olan siniflərin sayı)

Modelin qrafik təsviri :



Təlim parametrləri

Optimizer: optim.Adam(model.parameters(), lr=0.001)

Criterion / Loss: nn.CrossEntropyLoss()

Epochs: epochs = 5

Batch size: batch_size = 64

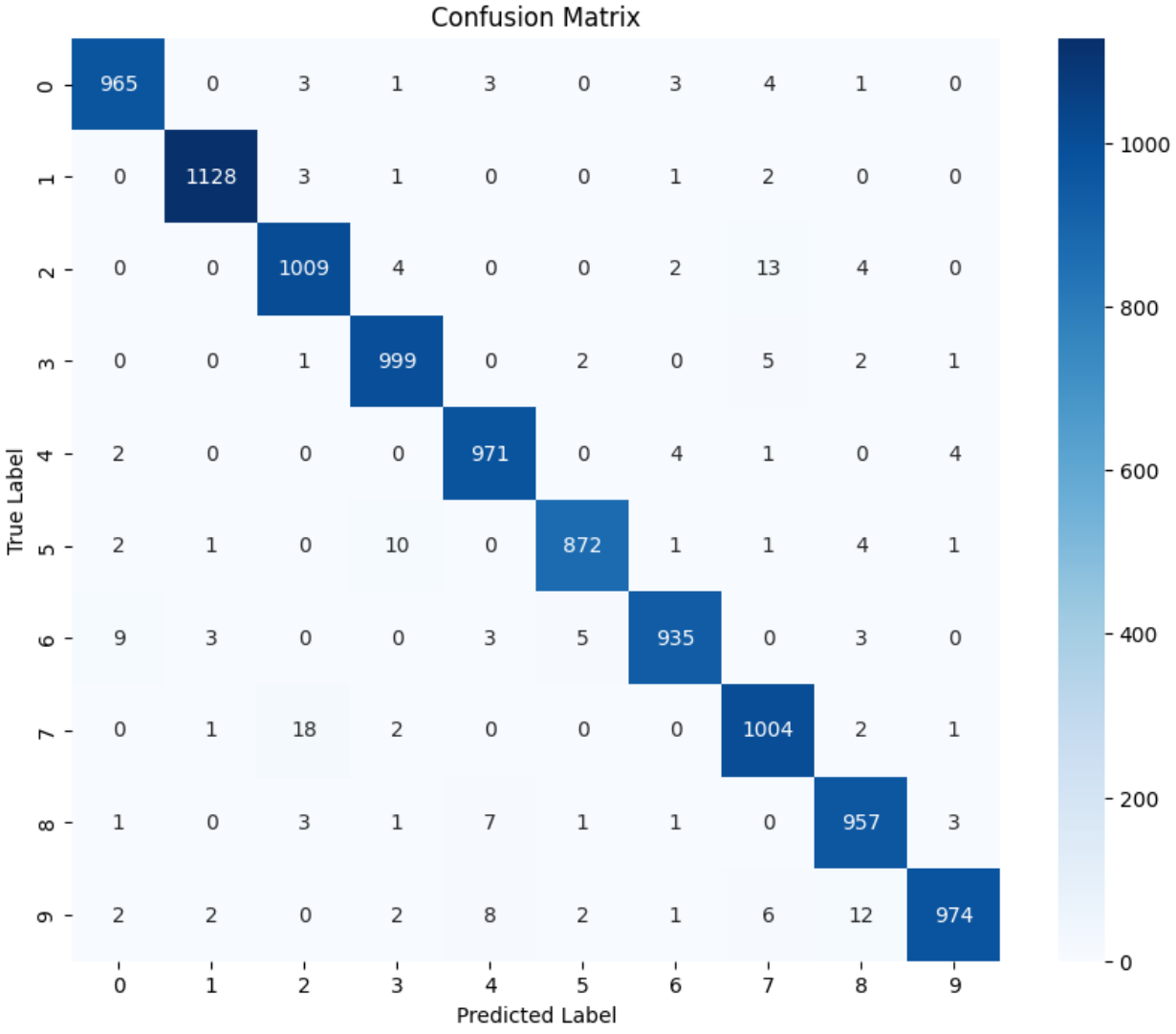
Device: avtomatik (cuda varsa GPU, yoxsa CPU)

Qiymətləndirmə və nəticələr

Təlim prosesində, hər bir epoch üçün itki (Loss) dəyəri çap olunur. Bu, istifadəçiyə modelin hər dövr ərzində necə öyrəndiyini izləmək imkanı verir və çap formatı Epoch {i}/{epochs}, Loss: ... şəklində göstərilir. Test mərhələsində isə modelin düzgün təsnifat etdiyi nümunələrin nisbəti əsasında dəqiqlik (accuracy) hesablanır; bu, **accuracy = 100 * correct / total** düsturu ilə əldə edilir və **print(f"Test Accuracy: {accuracy:.2f}%")** vasitəsilə ekrana çıxarılır. Təlim və test üçün istifadə olunan label-lər long tipinə çevrilir ki, PyTorch-un itki funksiyaları ilə uyğun işləsin. Məlumatların idarəsi üçün isə TensorDataset istifadə olunur və DataLoader vasitəsilə verilənlər batch-lərə bölünür; təlim zamanı isə batch ölçüsü 64 olaraq seçilir və verilənlər qarışdırılır (shuffle=True), beləliklə model daha sabit və ümumiləşdirici öyrənir.

Modelin test dəqiqliyi təlimdən sonra test datası üzərində **98.14 %** olmuşdur

Vizualizasiyalar



Yalnış sinifləndirilmiş şəkillərdən nümunlər: