|  |  |  |  |
| --- | --- | --- | --- |
|  | Politechnika Bydgoska im. J. J. Śniadeckich  Wydział Telekomunikacji,  Informatyki i Elektrotechniki  **Zakład Systemów Teleinformatycznych** | |  |
| **Przedmiot** | Sztuczne sieci neuronowe | | |
| **Prowadzący** | prof. dr hab. inż. prof. PBŚ Piotr Cofta | | |
| **Temat** | *Project* | | |
| **Student** | Cezary Tytko | | |
| **Ocena** |  | **Data oddania spr.** |  |

Etap 1. – kontrola danych.

Weryfikacji możemy dokonać przez wczytanie danych jako pliku csv i sprawdzić czy wymiary otrzymanej tablicy zgadzają się z przewidywanymi, to znaczy czy mamy odpowiednia liczbę rekordów i czy każdy rekord zawiera odpowiednią liczbę danych, to jest liczba oczek na kostce i 28 X 28 pikseli ułożonych w jednym wymiarze. Należy również sprawdzić czy wartości etykiet jak o danych są zgodne z założeniami, np. czy nie wychodzą poza przewidziany zakres, albo czy nie ma wartości brakujących, można to sprawdzić wyświetlając wartości unikatowe (Select distinct w konwencji sql), jeżeli dane będą zawierały błędy na poziome typów wartości tzn. string nie konwertowany na int, to dostaniemy błąd na etapie odczytu pliku csv (przynajmniej dla implementacji z pandas).

Przykładowy kod weryfikacji:

1. dice\_y, dice\_x = ReadDiceCSV(dice\_dir\_csv)

2. dice\_x = dice\_x.reshape((60000, 28, -1))

3. print("dice:")

4. print(f"Label Shape: {dice\_y.shape} has NAN: {np.isnan(dice\_y).any()}")

5. print(f"Values : {np.unique(dice\_y)}")

6. print(f"Data Shape: {dice\_x.shape} has NAN: {np.isnan(dice\_x).any()}")

7. print(f"Values : {np.unique(dice\_x)}")

8.

9. mnist\_y, mnist\_x = ReadDiceCSV(mnist\_train\_dir\_csv)

10. mnist\_x = mnist\_x.reshape((60000, 28, -1))

11. print("minst:")

12. print(f"Label Shape: {mnist\_y.shape} has NAN: {np.isnan(mnist\_y).any()}")

13. print(f"Values : {np.unique(mnist\_y)}")

14. print(f"Data Shape: {mnist\_x.shape} has NAN: {np.isnan(mnist\_x).any()}")

15. print(f"Values : {np.unique(mnist\_x)}")

16.

17. cifar\_y, cifar\_x = ReadDiceCSV(cifar\_all\_dir\_csv)

18. cifar\_x = cifar\_x.reshape((50000, 28, -1))

19. print("cifar:")

20. print(f"Label Shape: {cifar\_y.shape} has NAN: {np.isnan(cifar\_y).any()}")

21. print(f"Values : {np.unique(cifar\_y)}")

22. print(f"Data Shape: {cifar\_x.shape} has NAN: {np.isnan(cifar\_x).any()}")

23. print(f"Values : {np.unique(cifar\_x)}")

24.

Powyższa walidacja wskazuje na poprawność zbiorów danych wraz z etykietami. Można by również dokonać weryfikacji ręcznej sprawdzając treść obrazków (czy na zdjęciach kostek, faktycznie znajdują się kostki), ale na tym etapie zakładem weryfikacje na poziomie wartości i typów, wyświetlając tylko po jednym przykładowym elemencie zgodnie z poleceniem zadania.

Etap 2 – klasyfikacji według liczby oczek

Zastosowałem podejście, które określiłbym jako kombinacja 2 i 3 z opisu do zadania, tj. optymalizacja hperparametrów modelu, na modelu który został wybrany w oparciu o moją dotychczasową wiedzę i doświadczenie, nie mam na tyle dużej wiedzy by wybrać optymalny model tylko na tej podstawie, ale nie musiałem również sprawdzać wielu zestawów hiperparametrów, więc zostawiłem sobie pole do manewrów w tej dziedzinie nawykając moje podejście kombinacją 2 i 3.

Zdecydowałem się wykorzystać konwolucyjne sieci neuronowe (oczywisty wybór dla obrazów) utworzone z wykorzystaniem biblioteki pyTorch, zdecydowałem się na tą bibliotekę ponieważ będę z niej korzystał w ramach laboratorium z tego przedmiotu, jednak dotychczas korzystałem tylko z keras, więc pod względem obsługi i implementacji jest to dla mnie nowość, nowością nie są natomiast działania z sieciami konwolucyjnymi i wiedza od strony teoretycznej jak taka implementacja powinna wyglądać i co zawierać.

Utworzyłem sieć z w oparciu o warstwy konwolucyjne, poprzeplatane warstwami maxpool, na końcu umieściłem warstwę gęstą która odpowiada za ostateczną klasyfikację na 6 klas. Taki dobór wynika ze wcześniejszych doświadczeń dla warstw konwolucyjnych i maxpool, końcowa warstwa gęsta (w pyTorch Linear) jest natomiast wymagana ze względu na problem klasyfikacji na zadaną liczbę klas (funkcja straty to CrossEntropyLoss, albo dla keras CategoricalCrossentropy).

Struktura sieci (wejście/ typ/ wyjście):

- (1x28x28) / Conv2d(32) / (32x28x28)

- (32x28x28) / MaxPool2d(2,2) / (32x14x14)

- (32x14x14) / Conv2d(64) / (64x14x14)

- (64x14x14) / MaxPool2d(2,2) / (64x7x7)

- (64x7x7) / Linear(64) / (64)

- (64) / Linear(6) / (6)

Taka sieć uzyskała wynik dokładności na poziomie 98-99%, tzn. że tyle danych zostało sklasyfikowanych poprawnie.

Kod (z komentarzami wewnątrz):

1. x\_normalized = torch.tensor(dice\_x / 255.0, dtype=torch.float32)

2. print(x\_normalized.size())

3. y\_tensor = torch.tensor(dice\_y, dtype=torch.int64)

4.

5. # Podziel dane na zbiory treningowy i testowy

6. x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_normalized, y\_tensor, test\_size=0.2, random\_state=42, stratify=y\_tensor)

7. # Dodanie wymiaru dla kanałow obrazu

8. x\_train = x\_train.unsqueeze(1)

9. x\_test = x\_test.unsqueeze(1)

10. # Odjęcie 1 od etykiet, aby dane były z zakresu 0-5, nie wypływa to na model,

11. # trzeba pamiętać że po tej operacji wyniki nie odpowiadają wprost na rzeczywistą liczbą oczek

12. y\_train = y\_train -1

13. y\_test = y\_test -1

14. # Twórz obiekty DataLoader dla danych treningowych i testowych

15. batch\_size = 64

16. train\_dataset = TensorDataset(x\_train, y\_train)

17. train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

18. test\_dataset = TensorDataset(x\_test, y\_test)

19. test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

20.

21. # Definiuj model sieci konwolucyjnej

22. class CNN(nn.Module):

23. def \_\_init\_\_(self):

24. super(CNN, self).\_\_init\_\_()

25. self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, stride=1, padding=1)

26. self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)

27. self.pool = nn.MaxPool2d(2, 2)

28. self.fc1 = nn.Linear(64 \* 7 \* 7, 64)

29. self.fc2 = nn.Linear(64, 6) # Warstwa wyjściowa z 6 klasami

30.

31. def forward(self, x):

32. x = self.pool(torch.relu(self.conv1(x)))

33. x = self.pool(torch.relu(self.conv2(x)))

34. x = x.view(-1, 64 \* 7 \* 7)

35. x = torch.relu(self.fc1(x))

36. x = self.fc2(x)

37. return x

38.

39. # Inicjalizuj model

40. model = CNN()

41.

42. # Definiuj funkcję straty i optymizator

43. criterion = nn.CrossEntropyLoss()

44. optimizer = optim.Adam(model.parameters(), lr=0.001)

45.

46. # Trenuj model

47. num\_epochs = 10

48. for epoch in range(num\_epochs):

49. for images, labels in train\_loader:

50. optimizer.zero\_grad()

51. outputs = model(images)

52. loss = criterion(outputs, labels)

53. loss.backward()

54. optimizer.step()

55. print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}')

56.

57. # Ocena dokładności modelu na zbiorze testowym

58. with torch.no\_grad():

59. correct = 0

60. total = 0

61. for images, labels in test\_loader:

62. outputs = model(images)

63. \_, predicted = torch.max(outputs.data, 1)

64. total += labels.size(0)

65. correct += (predicted == labels).sum().item()

66. accuracy = 100 \* correct / total

67. print(f'Accuracy on test set: {accuracy:.2f}%')

68.