
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki Zakład Systemów Teleinformatycznych		
Przedmiot	Sztuczne sieci neuronowe		
Prowadzący	prof. dr hab. inż. prof. PBS Piotr Cofta		
Temat	<i>Project</i>		
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 odpowiednią 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 poziomie 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("mnist:")
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()}")

```

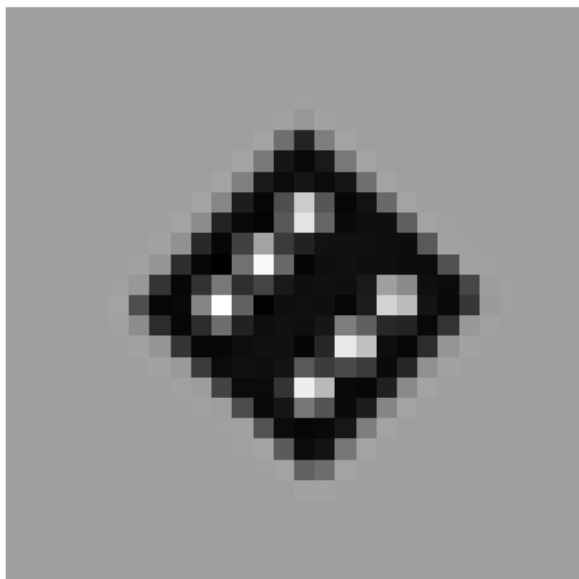
```
21. print(f"Values : {np.unique(cifar_y)}")
22. print(f"Data Shape: {cifar_x.shape} has NAN: {np.isnan(cifar_x).any()}")
23. print(f"Values : {np.unique(cifar_x)}")
24.
```

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 zakładam weryfikację na poziomie wartości i typów, wyświetlając tylko po jednym przykładowym elemencie zgodnie z poleceniem zadania.

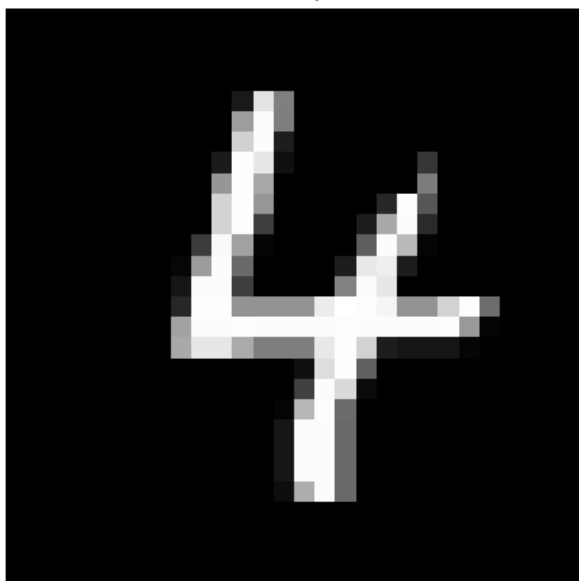
Cały kod umieściłem (w pliku main.ipynb) na:

https://github.com/Czarkowski/SSN_Project

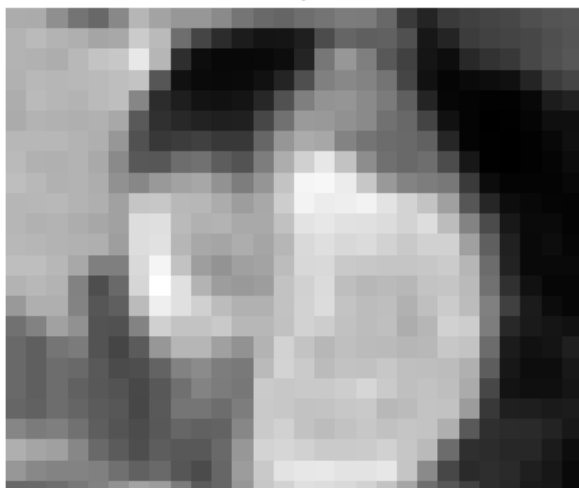
Dice 58873, value: 6



Mnist 58873, value: 4



Cifar 18873, value: 12



Etap 2 – klasyfikacji według liczby oczek

Zastosowałem podejście, które określiłbym jako kombinacja 2 i 3 z opisu do zadania, tj. optymalizacja hiperparametrów 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, przeplatane 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 kanałów 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 wpływa 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 optyimizator
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.
```

Chciałem poprawić wynik sieci, bo uważam że dla takiego problemu, możemy dojść do wyniku powyżej 99%, zwiększając liczbę epok do 20, udało mi się uzyskać wynik 99,8 % na zbiorze testowym (na zbiorze treningowym 99,7%) (przy analizie celności trzeba pamiętać, że celność na zbiorze treningowym jest brana dla predykcji przed aktualizacją wag w danej pętli treningowej i sumowana do ogólnej celności dla całej epoki, a celność na zbiorze testowym jest obliczana dla modelu po aktualizacji wag dla całej epoki). Dodatkowo wyświetliłem błędnie sklasyfikowane obrazki, aby sprawdzić czy problem nie leży np. w jakości zdjęć, albo błędnych etykietach, ale nie zauważyłem niczego co można by poprawić.

Etap 3 – klasyfikacja do domeny

Zacząłem od tego, że miałem już gotowy model sieci z poprzedniego zadania, więc połączyłem wszystkie zbiory nadając im etykiety zgodnie z pochodzeniem i wrzuciłem w model, dostałem tym sposobem skromny wynik 99,7%, nawet pomimo delikatnego nie zbalansowania klas, którego lepiej unikać dla sieci neuronowych, jak również moim zdaniem sieć była dość skomplikowana jak na problem rozpoznawania tylko 3 domen, moim zdaniem powinno to być coś lekkiego aby miało zastosowanie praktycznie

Zbalansowałem klasy wycinając fragmenty liczniejszych zbiorów, pozostawiając 50 tyś. przykładów per klas, oraz znacząco uprościłem model, do postaci:

- (1x28x28) / Conv2d(32, stride=2) / (16x14x14)
- (16x14x14) / MaxPool2d(2,2) / (16x7x7)
- (16x7x7) / Linear(16) / (16)
- (16) / Linear(3) / (3)

Po 10 epokach uzyskałam wynik 99,69%, praktycznie taki sam jak dla bardziej złożonej sieci, w tym przypadku jednak nie starałbym się próbować uzyskać wyniku 100%, ponieważ może okazać się to niemożliwe, dlatego że jestem sobie w stanie wyobrazić sytuację, że domena dla danego zdjęcia jest określona, to nie wyklucza to tego że nie może ona pasować do innej domeny, albo łudząco ją przypominać, oczywiście to wszystko będzie zależało od konkretnego przypadku już w praktyce, ale należy o tym pamiętać i zastosować odpowiednie podejście, na przykład na etapie projektowania systemu założyć możliwość wielo-klasowości i zaprojektować do tego odpowiedni klasyfikator.