Politechnika Wrocławska

Autor: Filip Strózik 260377

Sieci Neuronowe

Sprawozdanie z ćwiczenia 6

Technologia

Python 3.11, Jupyter Notebook, Torchvision, Google Colab środowisko T4

Realizacja ćwiczenia

Cel ćwiczenia

W ćwiczeniu 6 należy zbudować sieć wykorzystjącą wartswy CNN i operator max pooling do klasyfikacji obrazów. Zbiorem danych jest Fashion MNIST zawierający 70,000 obrazów 28x28 posiadających jeden kanał barwy. Eksperymenty będą wykonywane na 10% danych z racji tego, że danych jest bardzo dużo.

Architektura modelu sieci konwolucyjnej

```
class CustomCNN(nn.Module):
    def __init__(self, num_channels, kernel_size, pool_size):
        super(CustomCNN, self).__init__()

        self.conv_block = nn.Sequential(
            nn.Conv2d(1, num_channels, kernel_size=kernel_size),
            nn.LeakyReLU(),
            nn.MaxPool2d(kernel_size=pool_size),
        )

        self.fc_block = nn.Sequential(
            nn.Flatten(),
            LazyLinear(120),
            nn.LeakyReLU(),
            LazyLinear(10),
            nn.LogSoftmax(dim=1)
        )

    def forward(self, x):
        x = self.conv_block(x)
        x = self.fc_block(x)
        return x
```

Ta klasa reprezentuje prostą architekturę konwolucyjną (CNN) w PyTorch:

1. **Warstwa konwolucyjna:**
    - Wejście: Obrazy o jednym kanale (grayscale), ponieważ **nn.Conv2d(1, num_channels, kernel_size=kernel_size)** używa 1 jako liczby kanałów wejściowych.
    - Liczba kanałów wyjściowych: Określona przez parametr **num_channels**.
    - Rozmiar filtra konwolucyjnego: Określony przez parametr **kernel_size**.
    - Funkcja aktywacji: Leaky ReLU (nn.LeakyReLU()) - funkcja ta pozwala na przekazywanie pewnej ilości ujemnych wartości, co może pomóc w uczeniu bardziej rozbudowanych reprezentacji.
    - Warstwa Max Pooling: Redukuje wymiar przestrzenny danych poprzez wybieranie maksymalnej wartości z okna o określonym rozmiarze (określonym przez parametr **pool_size**).

2. **Warstwy w pełni połączone:**
    - **nn.Flatten()**: Spłaszcza dane do jednowymiarowego tensora, aby mogły być przekazane do warstw fully connected.
    - **LazyLinear(120)**: Warstwa fully connected z 120 neuronami. Parametr **120** został użyty jako rozmiar wyjścia tej warstwy.
    - Funkcja aktywacji: Leaky ReLU.
    - **LazyLinear(10)**: Warstwa fully connected z 10 neuronami, co sugeruje, że sieć jest przeznaczona do klasyfikacji na 10 klas.
    - Warstwa softmax (nn.LogSoftmax(dim=1)): Funkcja aktywacji, która przekształca wyniki na prawdopodobieństwa, szczególnie używana w problemach klasyfikacji.

Inicjalizacja wag:

```
def init_normal(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
        torch.manual_seed(seed)
        if hasattr(m, 'weight'):
            nn.init.xavier_normal_(m.weight)
        if hasattr(m, 'bias'):
            nn.init.constant_(m.bias, 0)
```

Transformacja danych:

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])
```

Dodawanie szumu gaussowskiego:

```
transform_blur = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)),
    GaussianBlur(kernel_size=9, sigma=(1.5, 5.0))
]
```

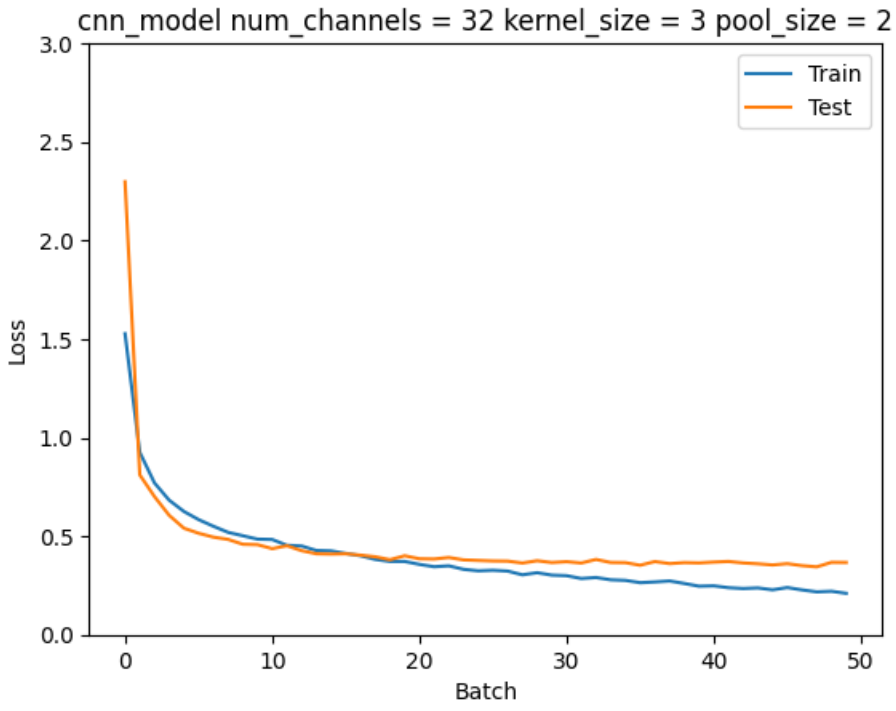
Politechnika Wrocławska

Autor: Filip Strózik 260377

Wyniki domyślnej architektury sieci

epok	50
batch	600
zbiór	10%

```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=5408, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
))
Training Time (in seconds) = 303.36
Accuracy: 0.8790
Precision: 0.8783
F1 Score: 0.8774
```



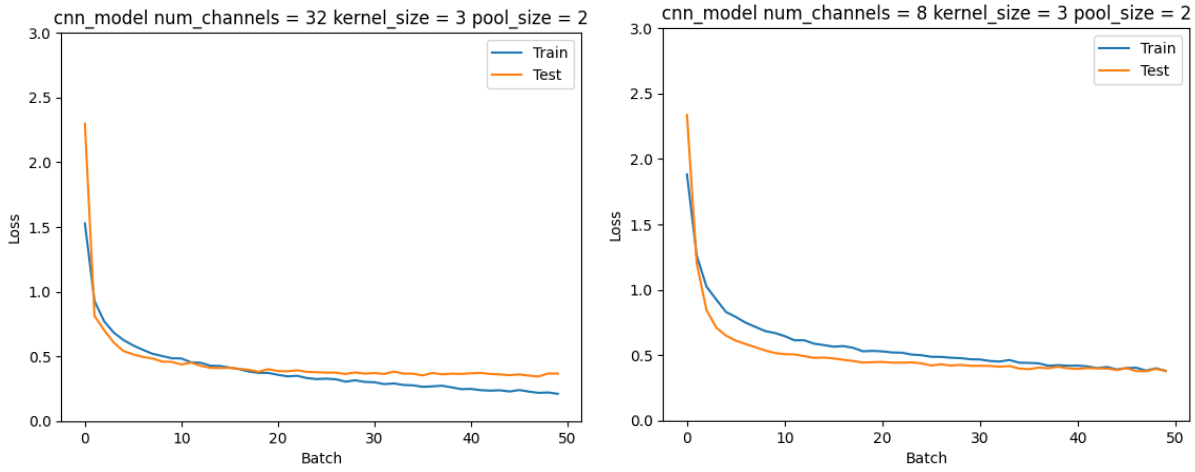
Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie liczby kanałów wyjściowych warstwy konwolucyjnej

```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=1352, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
Training Time (in seconds) = 114.16
Accuracy: 0.8600
Precision: 0.8579
F1 Score: 0.8566
```

Wyniki minimalnie gorsze. Na pewno trzeba zauważyć o połowę krótszy czas.

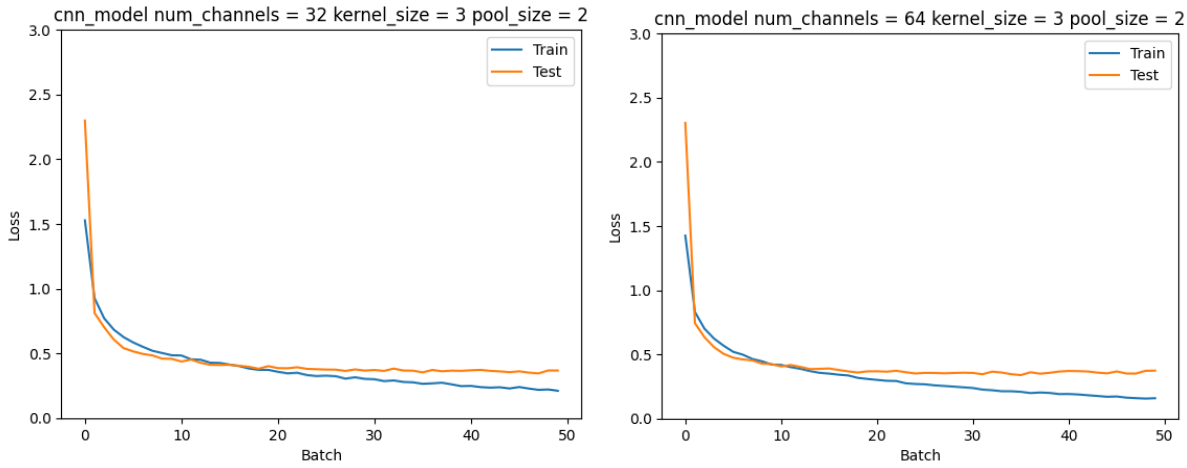


Można zauważyć, że zmniejszenie liczby kanałów warstwy konwolucyjnej sprawiło spowolnienie uczenia na danych treningowych. Również ciekawym jest fakt, iż koszt dla danych testowych czy walidacyjnych jest niższy niż dla treningowych. Zmniejszenie liczby kanałów może oznaczać, że model ma mniej możliwości do wykrywania złożonych cech w danych treningowych. To może prowadzić do spowolnienia procesu uczenia, ponieważ model musi bardziej subtelnie dostosować się do dostępnych informacji.

Politechnika Wrocławska

Autor: Filip Strózik 260377

```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=10816, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
Training Time (in seconds) = 578.93
Accuracy: 0.8850
Precision: 0.8841
F1 Score: 0.8836
```



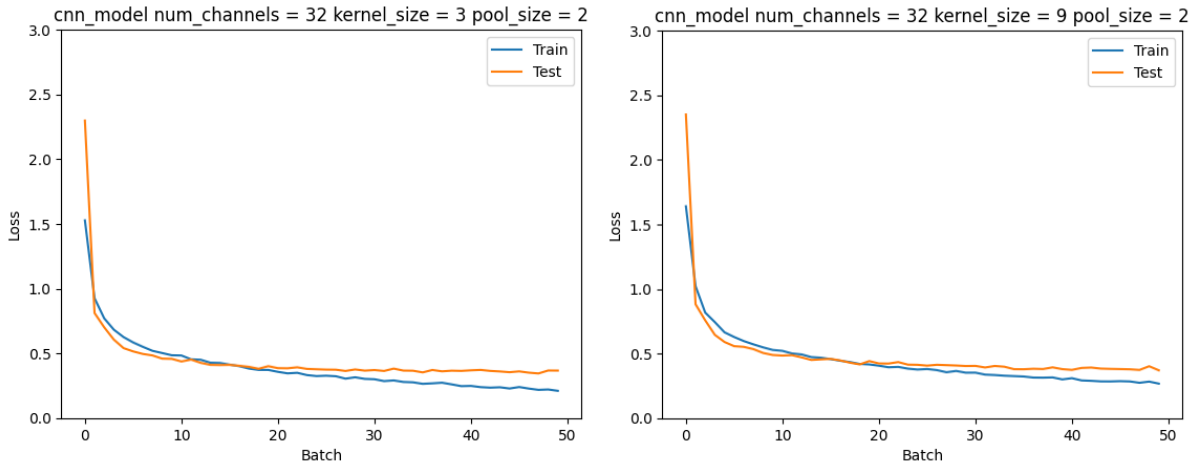
W przypadku zwiększenia liczby kanałów wyjściowych warstwy konwolucyjnej z 32 na 64, można zauważyć poprawę wyników. Uzyskaliśmy 88,5% dokładności prawie dwa razy dłuższym czasem wykonywania uczenia. Liczba wejść która wchodzi pierwszej warstwy liniowej jest dwa razy większa niż w przypadku domyślnego modelu. Zwiększona złożoność modelu sprawiła szybsze dostosowanie się do danych treningowych, co widać na wykresie.

Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie rozmiaru filtra warstwy konwolucyjnej

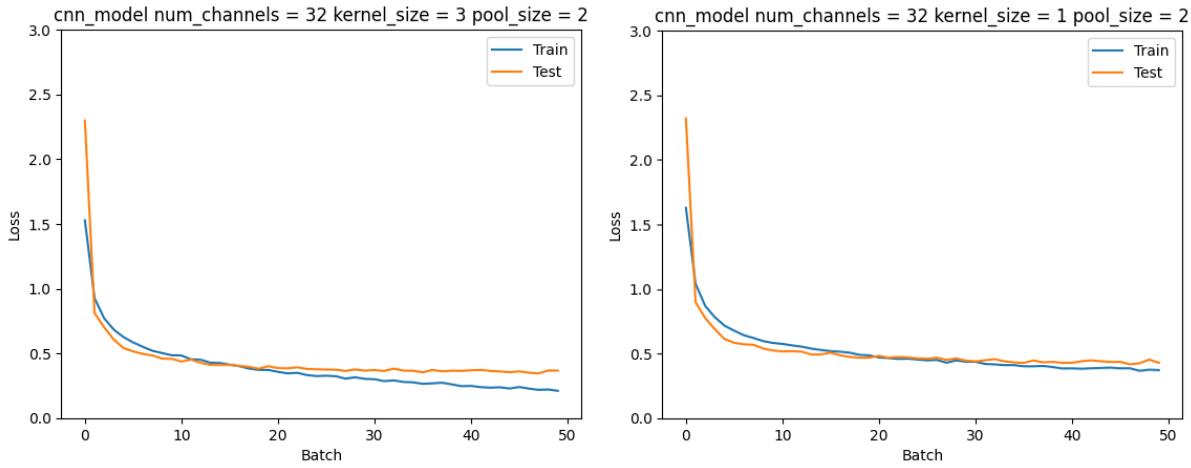
```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 32, kernel_size=(9, 9), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=3200, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
Training Time (in seconds) = 259.36
Accuracy: 0.8700
Precision: 0.8698
F1 Score: 0.8685
```



Lepszy czas, natomiast lekko gorsze wyniki. Biorąc pod stopień zwiększenia perceptywnego pola, można było spodziewać się większej różnicy w wynikach. Zwiększenie mogło spowodować reakcję na większe wzorce w danych lub do bardziej globalnych danych. Natomiast obrazy 28x28 nie posiadają takich globalnych wzorców. Spowodowało to ogólnie spowolnienie tempa uczenia się.

Autor: Filip Strózik 260377

```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 32, kernel_size=(1, 1), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=6272, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
Training Time (in seconds) = 328.56
Accuracy: 0.8480
Precision: 0.8473
F1 Score: 0.8455
```



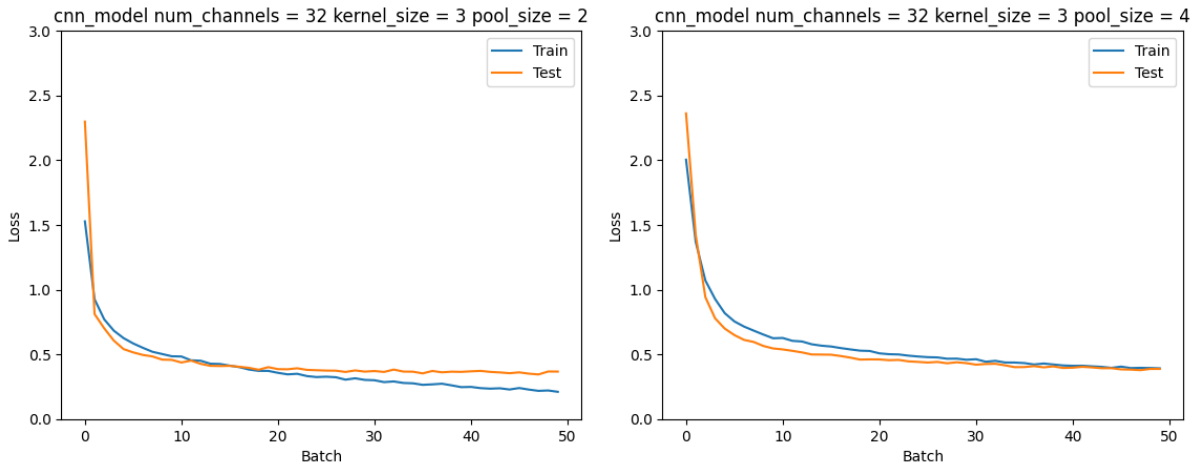
W tym porównaniu ograniczyliśmy perceptywne pole do jednego piksela w obrazie. Powoduje to, że nie zwieramy na wyjściu z warstwy informacji o otoczeniu piksela. Spowodowało to spowolnienie uczenia, stosunkowo dłuższy czas iteracji oraz już znacznie gorsze wyniki od domyślnej konfiguracji.

Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie rozmiaru okna poolingu

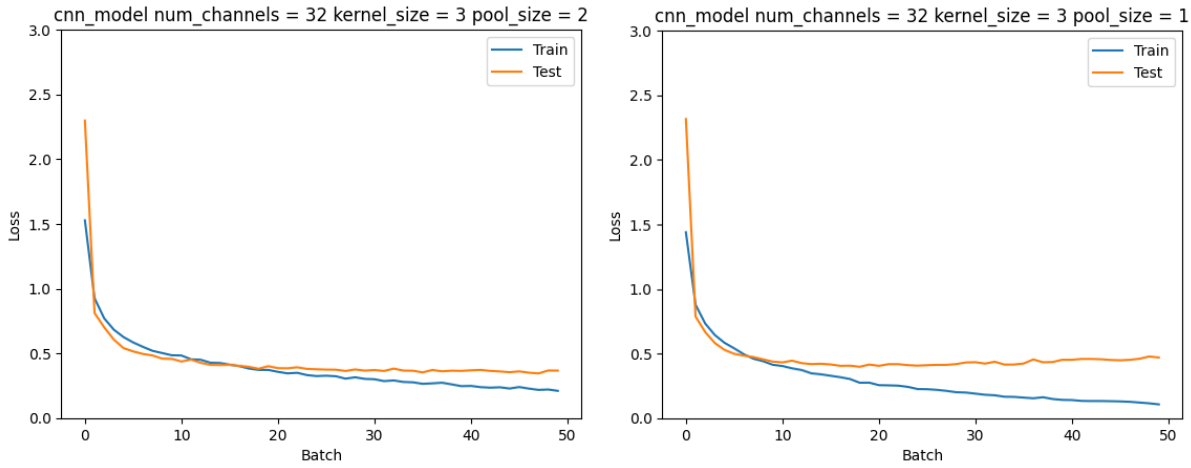
```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=1152, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
Training Time (in seconds) = 264.26
Accuracy: 0.8680
Precision: 0.8644
F1 Score: 0.8651
```



Zmiana rozmiaru okna poolingu z 2 na 4 sprawiła, że otrzymaliśmy lekko gorszy wynik. Zwiększenie okna powoduje większą redukcję przestrzennej rozdzielczości w warstwie konwolucyjnej, co prowadzi do utraty szczegółów. Zauważyć, trzeba ogólne spowolnienie uczenia jak i poziom kosztu dla danych treningowych.

Autor: Filip Strózik 260377

```
CustomCNN(
  (conv_block): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): Dropout2d(p=0.5, inplace=False)
    (2): LeakyReLU(negative_slope=0.01)
    (3): MaxPool2d(kernel_size=1, stride=1, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_block): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=21632, out_features=120, bias=True)
    (2): Dropout2d(p=0.5, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=120, out_features=10, bias=True)
    (5): LogSoftmax(dim=1)
  )
)
Training Time (in seconds) = 465.87
Accuracy: 0.8680
Precision: 0.8648
F1 Score: 0.8647
```

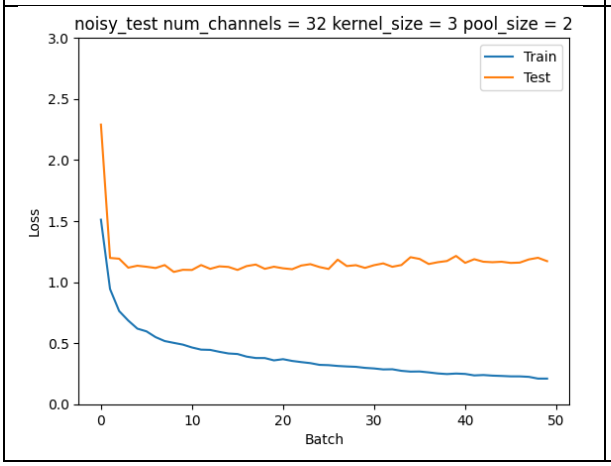
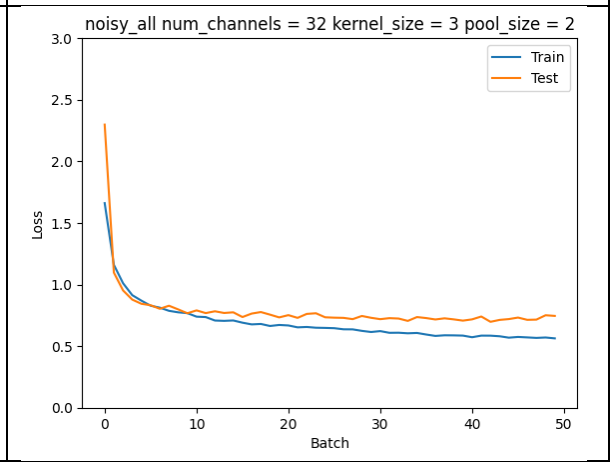


Trzeba zwrócić uwagę, że występuję tutaj brak redukcji wymiarowości, pierwsza warstwa liniowa przyjęła 21632 parametrów, co jest ogromną liczbą. Wyniki, również nie są znacznie gorsze niż w przypadku zwiększenia okna poolingu, aczkolwiek tendencja funkcji kosztu jest zupełnie inna. W przypadku zmniejszenia są bardziej rozbieżne. Można zauważyć coś co by przypominało przesunięcie funkcji kosztu w osi Loss wykresu.

Politechnika Wrocławska

Autor: Filip Strózik 260377

Porównanie zaburzenia danych

Training Time (in seconds) = 329.33 Accuracy: 0.5740 Precision: 0.6411 F1 Score: 0.5256	Training Time (in seconds) = 444.44 Accuracy: 0.7500 Precision: 0.7443 F1 Score: 0.7394
	

Podobnie jak w przypadku sieci bez warstw konwolucyjnych, szum w danych testowych sprawia, że koszt dla niespójnych danych jest wysoki. Warto zauważyć, że model, który otrzymał niezaszumione dane treningowe, uczy się jak w domyślnym modelu i po pierwszych epokach dalsze uczenie nieznacznie wpływa na wyniki kosztu dla testowych danych zaszumionych. Szum jest na tyle znaczący, że mimo lepszego dopasowania do szczegółów danych treningowych, model nie potrafi takich szczegółów rozpoznać w danych zaszumionych.

W przypadku spójności danych widać, że koszty są bardziej zbieżne. Mocny szum sprawia, że tracimy szczegóły w danych treningowych przez co trudniejsze jest lepsze dopasowanie modelu do danych. W tym przypadku uzyskujemy lepsze wyniki, aczkolwiek kosztem czasu.