Laporan Praktikum AI

CONVOLUTIONAL NEURAL NETWORK (CNN)

Komang Niko Romano Prodi | 222011356 | 3SD1

Pendahuluan CNN

CNN adalah jenis arsitektur jaringan saraf tiruan yang digunakan untuk memproses data grid seperti gambar. Dengan operasi konvolusi dan lapisan-lapisan tambahan, CNN mampu mengenali pola spasial dalam data dan melakukan ekstraksi fitur secara hierarkis. Ini membuatnya sangat efektif dalam tugas-tugas visi komputer seperti pengenalan objek dan klasifikasi gambar.

Arsitektur Convolutional Neural Network (CNN) terdiri dari beberapa jenis lapisan yang dikombinasikan secara hierarkis. Berikut adalah komponen utama dalam arsitektur NN:

a. **Lapisan Konvolusi (Convolutional Layer):** Lapisan ini terdiri dari filter atau kernel yang bergerak melintasi input secara konvolusi. Setiap kernel menghasilkan fitur-fitur khusus dari input. Hasil konvolusi dari setiap kernel menghasilkan peta fitur yang lebih kaya.
b. **Lapisan Aktivasi (Activation Layer):** Setelah lapisan konvolusi, lapisan aktivasi diterapkan untuk menerapkan fungsi aktivasi non-linear pada setiap elemen peta fitur. Fungsi aktivasi yang umum digunakan adalah ReLU (Rectified Linear Unit).
c. **Lapisan Pooling (Pooling Layer):** Lapisan pooling mengurangi dimensi spasial dari peta fitur dengan melakukan operasi seperti MaxPooling, AveragePooling, atau lainnya. Lapisan ini membantu mengurangi ukuran data, menjaga invariansi translasi, dan memperkuat fitur-fitur yang signifikan.
d. **Lapisan Fully Connected (Fully Connected Layer):** Lapisan ini menghubungkan setiap neuron dalam lapisan sebelumnya dengan setiap neuron di lapisan berikutnya. Ini mirip dengan jaringan saraf tiruan biasa dan digunakan untuk klasifikasi akhir atau tugas lainnya.
e. **Lapisan Output (Output Layer):** Lapisan output terdiri dari neuron-neuron yang menghasilkan prediksi akhir berdasarkan masukan dari lapisan sebelumnya. Jumlah neuron output sesuai dengan jumlah kelas dalam tugas klasifikasi.

Praktikum CNN (Image Classification Cifar10)

1. **Load dan Normalize Cifar10**

Load and normalize CIFAR10

```
import torch
import torchvision
import torchvision.transforms as transforms
```

2. Split Data

```
[ ]  # jumlah trainset sebelum split data
     print(len(trainset))
```

```
50000
```

```
[ ]  # setting seed
     import numpy
     import random

     def seed_worker(worker_id):
         worker_seed = torch.initial_seed() % 2**32
         numpy.random.seed(worker_seed)
         random.seed(worker_seed)

     g = torch.Generator()
     g.manual_seed(43)
     num_workers = 2
```

```
[ ]  # split data trainset sebelumnya menjadi trainset (90%) dan valset (10%)
     trainset, valset = torch.utils.data.random_split(trainset, [0.9, 0.1])
     print(len(trainset))
     print(len(valset))
```

```
45000
5000
```

3. Buat Dataloader

```
# Dataloader trainset, testset, dan valset

# trainloader
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g,
)

# testloader
testloader = torch.utils.data.DataLoader(
    testset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g,
)

#validationloader
valloader = torch.utils.data.DataLoader(
    valset,
    batch_size=batch_size,
    num_workers=num_workers,
    worker_init_fn=seed_worker,
    generator=g,
)
```

4. Buat Model CNN

a. Model Original (out_channel = 6)

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5) # model 7b
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

b. Model 7a (out_channel = 12)

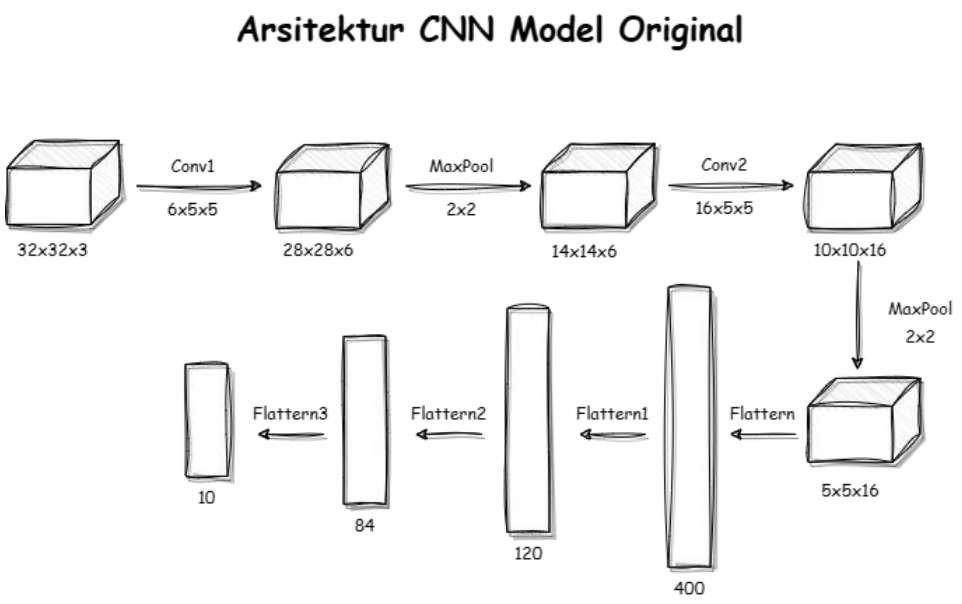
```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 5) # model
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

c. Model 7b (out_channel = 12, stride = 2, padding = 1)

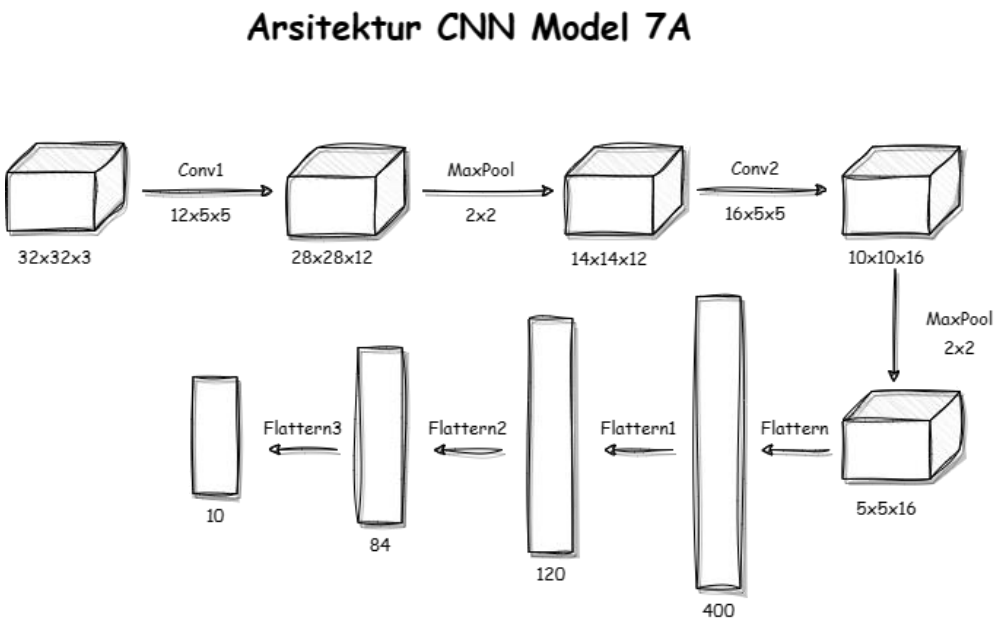
```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 5, stride = 2, padding = 1) #
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 16, 5, stride = 2, padding = 1)
        self.fc1 = nn.Linear(16 * 1 * 1, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

5. Arsitektur Model

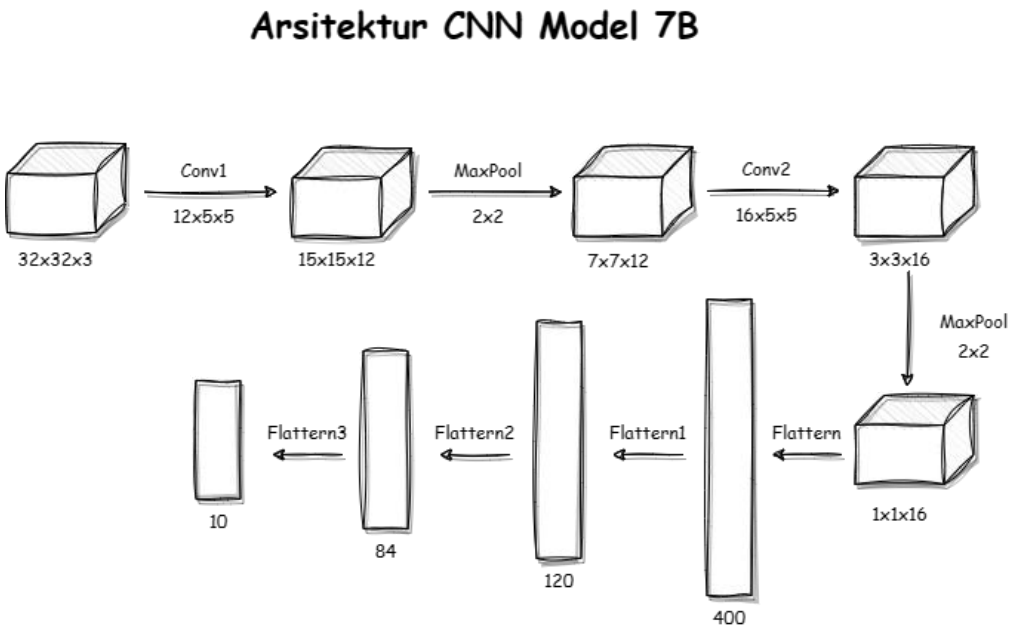
a. Model Original (out_channel = 6)



b. Model 7a (out_channel = 12)



c. Model 7b (out_channel = 12, stride = 2, padding = 1)



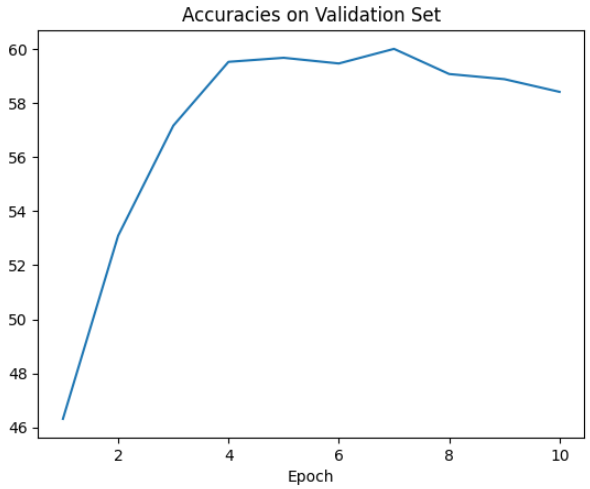
6. Proses Training

Proses training dilakukan dengan menggunakan epoch sebesar 10. Model diambil dari tingkat epoch yang memiliki akurasi pada set validasi tertinggi

a. Model Original (out_channel = 6)

```
[1,  2000] loss: 2.189
[1,  4000] loss: 1.830
[1,  6000] loss: 1.662
[1,  8000] loss: 1.573
[1, 10000] loss: 1.523
Epoch : 1: Accuracy of the network on the 5000 val images:46.32
[2,  2000] loss: 1.429
[2,  4000] loss: 1.421
[2,  6000] loss: 1.383
[2,  8000] loss: 1.344
[2, 10000] loss: 1.315
Epoch : 2: Accuracy of the network on the 5000 val images:53.09
[3,  2000] loss: 1.257
[3,  4000] loss: 1.265
[3,  6000] loss: 1.234
[3,  8000] loss: 1.223
[3, 10000] loss: 1.195
Epoch : 3: Accuracy of the network on the 5000 val images:57.16
[4,  2000] loss: 1.151
[4,  4000] loss: 1.156
[4,  6000] loss: 1.137
[4,  8000] loss: 1.135
[4, 10000] loss: 1.107
Epoch : 4: Accuracy of the network on the 5000 val images:59.52
[5,  2000] loss: 1.071
[5,  4000] loss: 1.078
[5,  6000] loss: 1.056
[5,  8000] loss: 1.053
[5, 10000] loss: 1.036
Epoch : 5: Accuracy of the network on the 5000 val images:59.67
[6,  2000] loss: 1.006
[6,  4000] loss: 1.018
[6,  6000] loss: 1.004
[6,  8000] loss: 0.987
[6, 10000] loss: 0.979
Epoch : 6: Accuracy of the network on the 5000 val images:59.46
[7,  2000] loss: 0.952
[7,  4000] loss: 0.964
[7,  6000] loss: 0.954
[7,  8000] loss: 0.929
[7, 10000] loss: 0.939
Epoch : 7: Accuracy of the network on the 5000 val images:60.0
[8,  2000] loss: 0.903
```

```
[8,  4000] loss: 0.918
[8,  6000] loss: 0.914
[8,  8000] loss: 0.895
[8, 10000] loss: 0.897
Epoch : 8: Accuracy of the network on the 5000 val images:59.07
[9,  2000] loss: 0.877
[9,  4000] loss: 0.869
[9,  6000] loss: 0.877
[9,  8000] loss: 0.857
[9, 10000] loss: 0.861
Epoch : 9: Accuracy of the network on the 5000 val images:58.88
[10,  2000] loss: 0.843
[10,  4000] loss: 0.840
[10,  6000] loss: 0.845
[10,  8000] loss: 0.814
[10, 10000] loss: 0.827
Epoch : 10: Accuracy of the network on the 5000 val images:58.41
Finished Training
```




```python
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1) # menggunakan detach() dan cpu()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

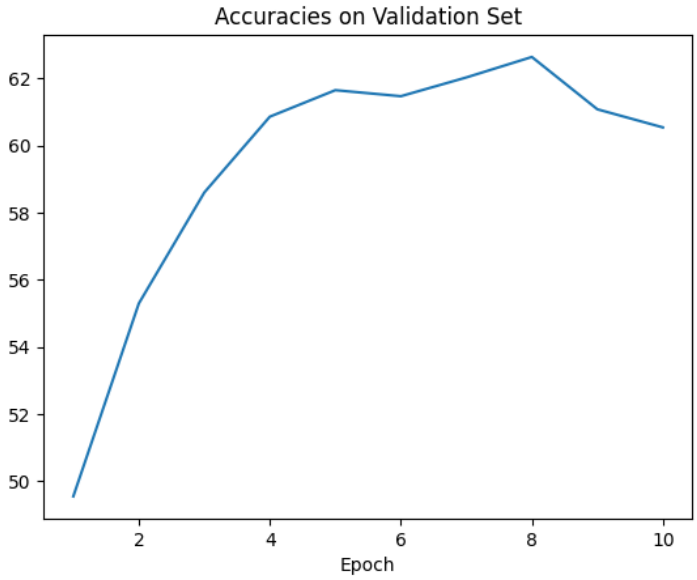
```
Accuracy of the network on the 10000 test images: 58 %
```

```
Accuracy for class: plane is 68.3 %
Accuracy for class: car   is 69.5 %
Accuracy for class: bird  is 42.2 %
Accuracy for class: cat   is 34.7 %
Accuracy for class: deer  is 50.0 %
Accuracy for class: dog   is 47.0 %
Accuracy for class: frog  is 78.5 %
Accuracy for class: horse is 63.2 %
Accuracy for class: ship  is 65.9 %
Accuracy for class: truck is 64.8 %
```

b. Model 7a (out_channel = 12)

```
[1,  2000] loss: 2.201
[1,  4000] loss: 1.814
[1,  6000] loss: 1.643
[1,  8000] loss: 1.536
[1, 10000] loss: 1.467
Epoch : 1: Accuracy of the network on the 5000 val images:49.54
[2,  2000] loss: 1.363
[2,  4000] loss: 1.366
[2,  6000] loss: 1.324
[2,  8000] loss: 1.274
[2, 10000] loss: 1.254
Epoch : 2: Accuracy of the network on the 5000 val images:55.29
[3,  2000] loss: 1.180
[3,  4000] loss: 1.200
[3,  6000] loss: 1.174
[3,  8000] loss: 1.142
[3, 10000] loss: 1.127
Epoch : 3: Accuracy of the network on the 5000 val images:58.6
[4,  2000] loss: 1.064
[4,  4000] loss: 1.087
[4,  6000] loss: 1.074
[4,  8000] loss: 1.038
[4, 10000] loss: 1.045
Epoch : 4: Accuracy of the network on the 5000 val images:60.86
[5,  2000] loss: 0.978
[5,  4000] loss: 1.003
[5,  6000] loss: 0.988
[5,  8000] loss: 0.963
[5, 10000] loss: 0.966
Epoch : 5: Accuracy of the network on the 5000 val images:61.65
[6,  2000] loss: 0.922
[6,  4000] loss: 0.938
```

```
[6,  6000] loss: 0.934
[6,  8000] loss: 0.891
[6, 10000] loss: 0.898
Epoch : 6: Accuracy of the network on the 5000 val images:61.47
[7,  2000] loss: 0.867
[7,  4000] loss: 0.886
[7,  6000] loss: 0.888
[7,  8000] loss: 0.838
[7, 10000] loss: 0.849
Epoch : 7: Accuracy of the network on the 5000 val images:62.03
[8,  2000] loss: 0.812
[8,  4000] loss: 0.843
[8,  6000] loss: 0.843
[8,  8000] loss: 0.797
[8, 10000] loss: 0.819
Epoch : 8: Accuracy of the network on the 5000 val images:62.64
[9,  2000] loss: 0.791
[9,  4000] loss: 0.805
[9,  6000] loss: 0.797
[9,  8000] loss: 0.766
[9, 10000] loss: 0.785
Epoch : 9: Accuracy of the network on the 5000 val images:61.08
[10,  2000] loss: 0.751
[10,  4000] loss: 0.762
[10,  6000] loss: 0.756
[10,  8000] loss: 0.726
[10, 10000] loss: 0.752
Epoch : 10: Accuracy of the network on the 5000 val images:60.54
Finished Training
```



```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1) # menggunakan detach() dan cpu()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

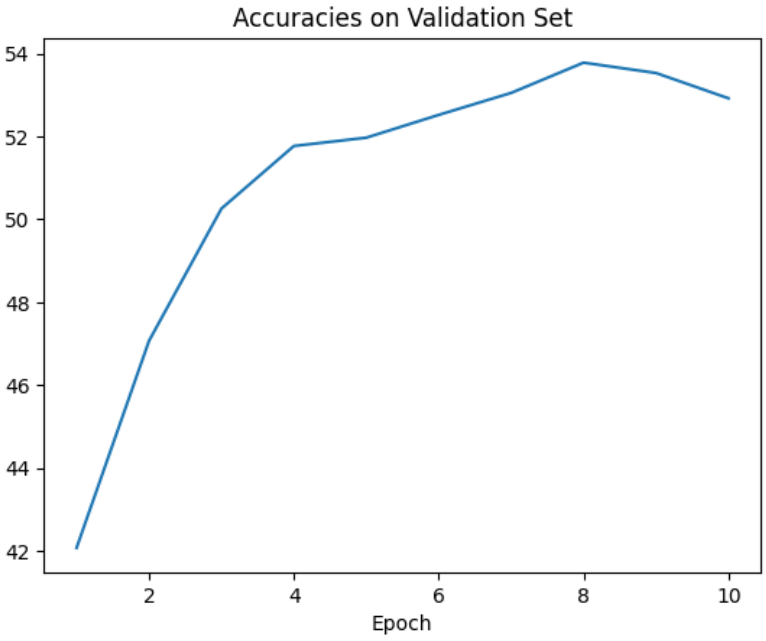
```
Accuracy of the network on the 10000 test images: 60 %
```

```
Accuracy for class: plane is 81.5 %
Accuracy for class: car   is 73.2 %
Accuracy for class: bird  is 60.3 %
Accuracy for class: cat   is 39.7 %
Accuracy for class: deer  is 49.5 %
Accuracy for class: dog   is 37.0 %
Accuracy for class: frog  is 74.0 %
Accuracy for class: horse is 63.7 %
Accuracy for class: ship  is 57.6 %
Accuracy for class: truck is 68.9 %
```

c. Model 7b (out_channel = 12, stride = 2, padding = 1)

```
[1,  2000] loss: 2.268
[1,  4000] loss: 2.023
[1,  6000] loss: 1.843
[1,  8000] loss: 1.719
[1, 10000] loss: 1.650
Epoch : 1: Accuracy of the network on the 5000 val images:42.08
[2,  2000] loss: 1.544
[2,  4000] loss: 1.550
[2,  6000] loss: 1.534
[2,  8000] loss: 1.477
[2, 10000] loss: 1.468
Epoch : 2: Accuracy of the network on the 5000 val images:47.07
[3,  2000] loss: 1.414
[3,  4000] loss: 1.425
[3,  6000] loss: 1.422
[3,  8000] loss: 1.385
[3, 10000] loss: 1.385
```

```
Epoch : 3: Accuracy of the network on the 5000 val images:50.26
[4,  2000] loss: 1.341
[4,  4000] loss: 1.350
[4,  6000] loss: 1.356
[4,  8000] loss: 1.324
[4, 10000] loss: 1.327
Epoch : 4: Accuracy of the network on the 5000 val images:51.77
[5,  2000] loss: 1.292
[5,  4000] loss: 1.310
[5,  6000] loss: 1.311
[5,  8000] loss: 1.288
[5, 10000] loss: 1.286
Epoch : 5: Accuracy of the network on the 5000 val images:51.97
[6,  2000] loss: 1.260
[6,  4000] loss: 1.275
[6,  6000] loss: 1.277
[6,  8000] loss: 1.257
[6, 10000] loss: 1.257
Epoch : 6: Accuracy of the network on the 5000 val images:52.52
[7,  2000] loss: 1.233
[7,  4000] loss: 1.252
[7,  6000] loss: 1.253
[7,  8000] loss: 1.235
[7, 10000] loss: 1.238
Epoch : 7: Accuracy of the network on the 5000 val images:53.05
[8,  2000] loss: 1.215
[8,  4000] loss: 1.234
[8,  6000] loss: 1.234
[8,  8000] loss: 1.218
[8, 10000] loss: 1.219
Epoch : 8: Accuracy of the network on the 5000 val images:53.78
[9,  2000] loss: 1.206
[9,  4000] loss: 1.217
[9,  6000] loss: 1.220
[9,  8000] loss: 1.204
[9, 10000] loss: 1.208
Epoch : 9: Accuracy of the network on the 5000 val images:53.53
[10,  2000] loss: 1.192
[10,  4000] loss: 1.207
[10,  6000] loss: 1.207
[10,  8000] loss: 1.192
[10, 10000] loss: 1.192
Epoch : 10: Accuracy of the network on the 5000 val images:52.92
Finished Training
```




```python
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1) # menggunakan detach() dan cpu()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

```
Accuracy of the network on the 10000 test images: 52 %
```

```
Accuracy for class: plane is 65.0 %
Accuracy for class: car   is 59.7 %
Accuracy for class: bird  is 44.3 %
Accuracy for class: cat   is 33.7 %
Accuracy for class: deer  is 25.8 %
Accuracy for class: dog   is 33.6 %
Accuracy for class: frog  is 82.1 %
Accuracy for class: horse is 68.5 %
Accuracy for class: ship  is 53.6 %
Accuracy for class: truck is 62.9 %
```

7. Perbandingan Model

Model	Akurasi Testing (%)
Model Original	58
Model 7a	60
Model 7b	52

Dari output yang telah ditampilkan, dapat diambil kesimpulan bahwa:

a. Model original mencapai akurasi validasi tertinggi pada epoch ke-7 yaitu sebesar 60%
b. Model 7a mencapai akurasi validasi tertinggi pada epoch ke-8 yaitu sebesar 62.64%
c. Model 7b mencapai akurasi validasi tertinggi pada epoch ke-8 yaitu sebesar 53.78%
d. Model 7a memiliki akurasi testing terbaik dibandingkan model lainnya yaitu sebesar 60%, sedangkan model 7b merupakan model yang terburuk karena memiliki akurasi testing terendah yaitu sebesar 52%.