Tugas 1 Individu Machine Learning
Nama : Syafri Wira Wicaksana
NIM : 226150100111018
Kelas : Machine Learning – A

Tugas

- Build a convolutional network with pre-processing on the input data (jittering, normalization). Also add dropout regularization, batch normalization, and at least one additional convolutional layer which achieves at least 90% test accuracy (for any training epoch) on MNIST dataset. Your part 1 network should train under 10 minutes, without GPUs.
- Fine-tune AlexNet to achieve at least 80% test accuracy on the MNIST dataset. Your network should train under 10 minutes, without GPUs.

Link Repository Github
https://github.com/wirasyafri/machine-learning/tree/master/Tugas%201

1. CNN

```
import time
import torch
import torch.nn as nn
import torchvision
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.autograd import Variable
import torch.nn.functional as F
import torch.optim as optim
from sklearn.metrics import classification_report, confusion_matrix
# transform  = transforms.ToTensor()
trans_train = transforms.Compose([transforms.Lambda(lambda image: image.convert('RGB')),
                                  transforms.ColorJitter(brightness=0.05, contrast=0.8,
saturation=0.02, hue=0.02),
                                  transforms.Resize((64,64)),
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.5,), (1.0,))])

trans = transforms.Compose([transforms.Lambda(lambda image: image.convert('RGB')),
                            transforms.ColorJitter(brightness=0.05, contrast=0.8,
saturation=0.02, hue=0.02),
                            transforms.Resize((64,64)),
                            transforms.ToTensor(),
                            transforms.Normalize((0.5,), (1.0,))])
train_data = dset.MNIST(root='../Data', train=True, download=True, transform=trans_train)
test_data = dset.MNIST(root='../Data', train=False, download=False, transform=trans)
print(train_data)
image,label=train_data[10]
print(image.shape)
```

```
print(label)
#loader data untuk training dan testing
batch_size = 64
train_loader =
torch.utils.data.DataLoader(dataset=train_data,batch_size=batch_size,shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_data,batch_size=batch_size,shuffle=False)
class SimpleNet(nn.Module):
    def __init__(self, num_classes):
        super(SimpleNet, self).__init__()
        #bagian 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu1 = nn.ReLU(inplace=True)
        self.max_poo1 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3)
        self.bn2 = nn.BatchNorm2d(32)
        self.relu2 = nn.ReLU(inplace=True)
        self.max_poo2 = nn.MaxPool2d(kernel_size = 2, stride = 2)

        # bagian 2
        self.conv_layer3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3)
        self.bn3 = nn.BatchNorm2d(64)
        self.relu3 = nn.ReLU(inplace=True)
        self.max_poo3 = nn.MaxPool2d(kernel_size = 2, stride = 2)
        self.conv_layer4 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3)
        self.bn4 = nn.BatchNorm2d(64)
        self.relu4 = nn.ReLU(inplace=True)
        self.max_poo4 = nn.MaxPool2d(kernel_size = 2, stride = 2)

        #bagian classifier
        self.fc1 = nn.Linear(256, 4096)
        # self.relu5 = nn.ReLU(inplace=True)
        self.dropout1 = nn.Dropout(p=0.1)
        self.fc4 = nn.Linear(4096, num_classes)
        self.classifier = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu1(out)
        out = self.max_poo1(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu2(out)
        out = self.max_poo2(out)

        out = self.conv_layer3(out)
        out = self.bn3(out)
        out = self.relu3(out)
        out = self.max_poo3(out)
        out = self.conv_layer4(out)
        out = self.bn4(out)
        out = self.relu4(out)
        out = self.max_poo4(out)
```

```
        out = out.view(out.size(0), -1) # mengubah dimensi tensor

        out = self.fc1(out)
        # out = self.relu5(out)
        out = self.dropout1(out)
        out = self.fc4(out)
        out = self.classifier(out)
        return out
model = SimpleNet(num_classes = 10)
model
def train(epoch):
    model.train()
    start_time = time.time()
    correct = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        # if torch.cpu.is_available():
        #     data, target = data.cpu(), target.cpu()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        train_losses.append(loss.item())
        loss.backward()
        optimizer.step()

        # Menghitung jumlah prediksi yang benar
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

        if batch_idx % 100 == 0:
            print('\rEpoch: {} {:.0f}%\t     Loss: {:.6f}'.format(
                epoch,
                100. * batch_idx / len(train_loader), loss.item()), end='')
    end_time = time.time()
    print("\nLama waktu Training pada epoch {}: {} Menit".format(epoch, ((end_time -
start_time)/60)))

    # Menghitung dan mencetak akurasi pelatihan
    train_accuracy = 100. * correct / len(train_loader.dataset)
    train_akurasi.append(train_accuracy)
    print('Akurasi pada Training pada epoch {}: {:.2f}%'.format(epoch, train_accuracy))
def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        # if torch.cpu.is_available():
        #     data, target = data.cpu(), target.cpu()
        data, target = Variable(data), Variable(target)
        output = model(data)
        test_loss += F.cross_entropy(output, target, reduction='sum').item()
        pred = output.data.max(1, keepdim=True)[1] # get the index of the max log-probability
        correct += pred.eq(target.data.view_as(pred)).long().cpu().sum()
```

```python
    test_loss /= len(test_loader.dataset)
    test_losses.append(test_loss)
    acc=100. * float(correct.to(torch.device('cpu')).numpy())
    print('\nHasil Testing: Rata-Rata loss: {:.4f}, Akurasi: {:.4f}%\n'.format(
        test_loss, acc / len(test_loader.dataset)))

    test_accuracy.append(acc / len(test_loader.dataset))
optimizer = optim.SGD(model.parameters(), lr=0.01)

train_losses = []
test_losses =[]
test_accuracy = []
train_akurasi = []
for epoch in range(1, 2):
    train(epoch)
    test()
# membuat prediksi dan ground truth labels
y_pred = []
y_true = []
for data, target in test_loader:
    data, target = Variable(data), Variable(target)
    output = model(data)
    pred = output.data.max(1, keepdim=True)[1].cpu().numpy().squeeze()
    y_pred.extend(pred)
    y_true.extend(target.data.cpu().numpy())

# membuat confusion matrix dan classification report
conf_mat = confusion_matrix(y_true, y_pred)
class_report = classification_report(y_true, y_pred)
print('Confusion Matrix:')
print(conf_mat)
print('Classification Report:')
print(class_report)
```

Arsitektur model

```
SimpleNet(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (max_poo1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (max_poo2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv_layer3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu3): ReLU(inplace=True)
  (max_poo3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv_layer4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4): ReLU(inplace=True)
  (max_poo4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=256, out_features=4096, bias=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (fc4): Linear(in_features=4096, out_features=10, bias=True)
  (classifier): Softmax(dim=1)
)
```

Hasil Pelatihan

```
Epoch: 1 96%         Loss: 1.591911
Lama waktu Training pada epoch 1: 4.317583632469177 Menit
Akurasi pada Training pada epoch 1: 81.66%

Hasil Testing: Rata-Rata loss: 1.5222, Akurasi: 96.6500%
```

Hasil *Confusion Matrix*

```
Confusion Matrix:
[[ 963    0    7    0    0    2    7    1    0    0]
 [    0 1125    5    1    0    1    2    1    0    0]
 [    6    2  987    1   10    0    7   18    1    0]
 [    1    1   18  971    0    4    1   10    1    3]
 [    1    3    1    0  964    0    6    0    0    7]
 [    5    5    1   12    1  858    6    1    3    0]
 [    9    5    0    0    3    3  938    0    0    0]
 [    0    3   29    1    0    1    0  990    0    4]
 [    8    3   17    3    5    6    6    8  912    6]
 [    9    8    5    3    8    5    0   10    2  959]]
```

Hasil Akurasi

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       980
           1       0.98      0.99      0.98      1135
           2       0.92      0.95      0.94      1032
           3       0.98      0.96      0.97      1010
           4       0.97      0.98      0.98       982
           5       0.98      0.97      0.97       892
           6       0.97      0.98      0.98       958
           7       0.95      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.98      0.95      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

97%

2. Alexnet + Finetune

Load model

```
#inisialisasi Model AlexNet
Fitur = 10

model1 = torchvision.models.alexnet(weights='AlexNet_Weights.DEFAULT')
model1.classifier.add_module('6', nn.Linear(in_features=4096, out_features=Fitur))
model1.classifier.add_module('7', nn.Softmax(dim=1))
model = model1
model
```

Fine tune pada model

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
    (7): Softmax(dim=1)
  )
)
```

Hasil Training

```
Epoch: 1 96%          Loss: 1.483884
Lama waktu Training pada epoch 1: 7.372456188996633 Menit
Akurasi pada Training pada epoch 1: 89.77%

Hasil Testing: Rata-Rata loss: 1.4860, Akurasi: 97.6300%
```

Confussion Matrix

```
Confusion Matrix:
[[ 974    0    3    0    0    0    1    1    0    1]
 [   0 1121    3    3    2    0    2    4    0    0]
 [   4    1 1018    3    1    0    0    4    1    0]
 [   0    0    9  992    0    1    0    3    5    0]
 [   0    0    0    0  971    0    0    1    2    8]
 [   2    0    1   18    1  845    4    3   16    2]
 [   7    1    2    1    5    2  927    0   13    0]
 [   0    1    9    1    2    0    0 1009    3    3]
 [   4    0    8    7    3    1    0    4  933   14]
 [   4    2    2    4    8    0    0   12    8  969]]
```

Hasil Akurasi

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       1.00      0.99      0.99      1135
           2       0.96      0.99      0.98      1032
           3       0.96      0.98      0.97      1010
           4       0.98      0.99      0.98       982
           5       1.00      0.95      0.97       892
           6       0.99      0.97      0.98       958
           7       0.97      0.98      0.98      1028
           8       0.95      0.96      0.95       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

Akurasi 98%