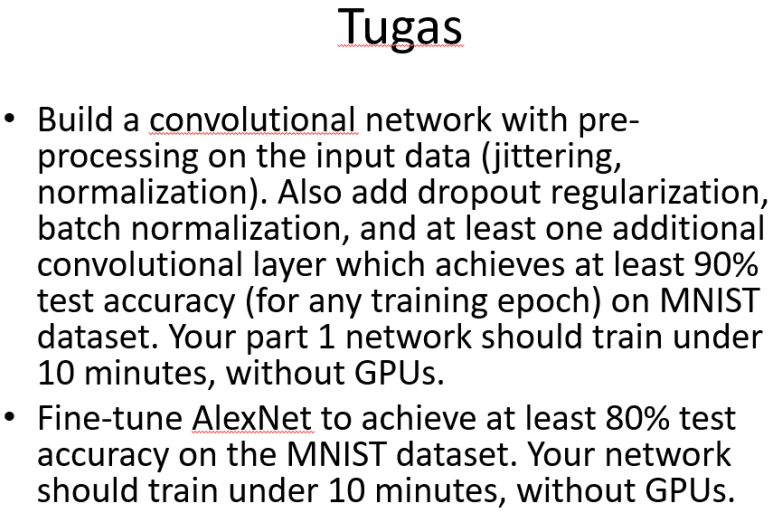
Tugas 1 Individu Machine Learning

Nama : Syafri Wira Wicaksana

NIM : 226150100111018

Kelas : Machine Learning – A



**Link Repository Github**

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()

    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)

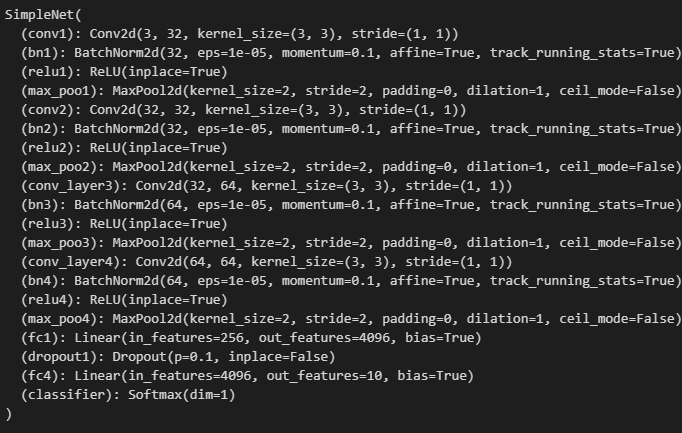
print('Confusion Matrix:')

print(conf\_mat)

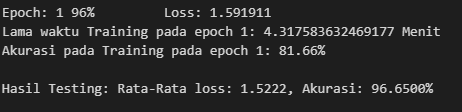
print('Classification Report:')

print(class\_report)

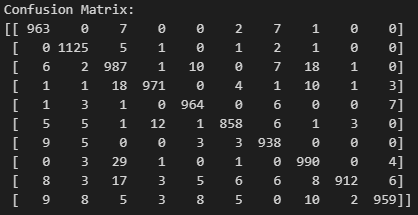
**Arsitektur model**



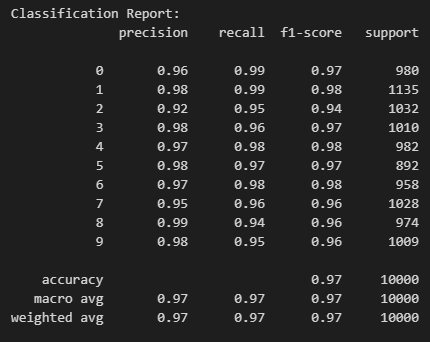
**Hasil Pelatihan**



**Hasil *Confusion Matrix***



**Hasil Akurasi**



**97%**

1. **Alexnet + Finetune**

**Load model**

#inisialisasi Model AlexNet

Fitur = 10

model1 = torchvision.models.alexnet(weights='AlexNet\_Weights.DEFAULT')

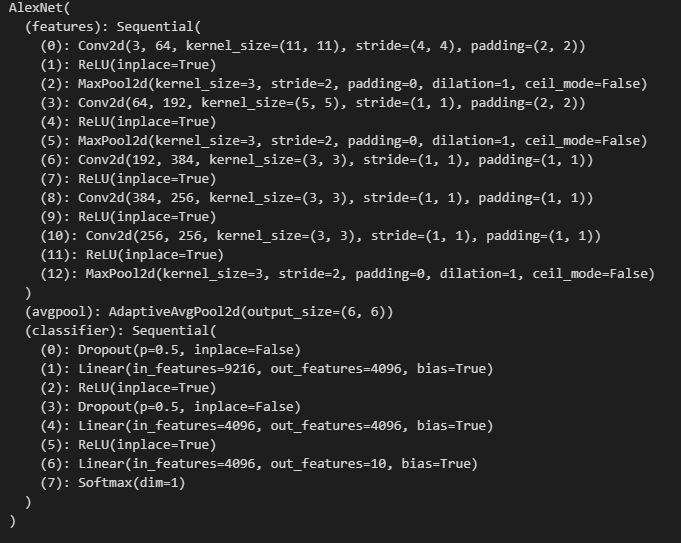
model1.classifier.add\_module('6', nn.Linear(in\_features=4096, out\_features=Fitur))

model1.classifier.add\_module('7', nn.Softmax(dim=1))

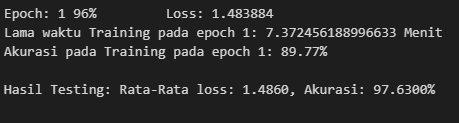
model = model1

model

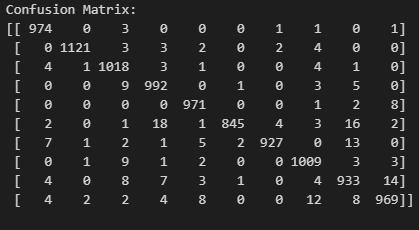
**Fine tune pada model**



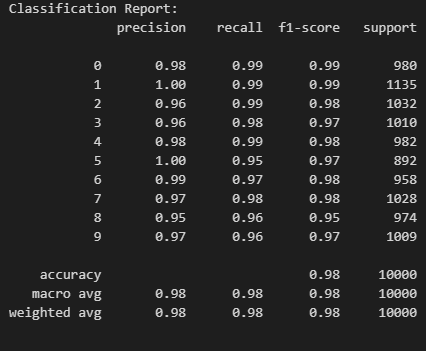
**Hasil Training**



**Confussion Matrix**



**Hasil Akurasi**



Akurasi 98%