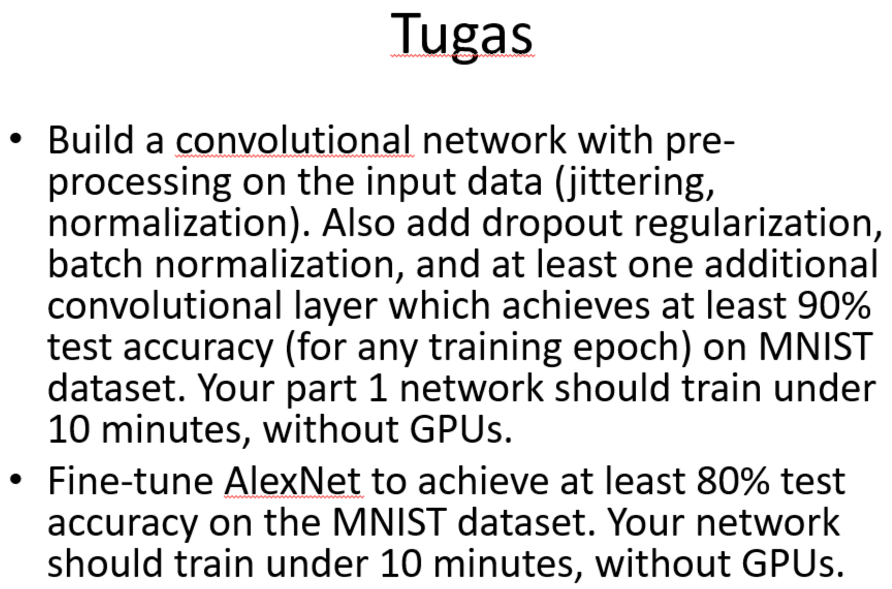
Tugas 1 Individu *Machine Learning*

Nama : Alfirsa Damasyifa Fauzulhaq
NIM : 226150100111019
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/AFauzulh/MK-Machine-Learning

Jawab :

1) Model CNN

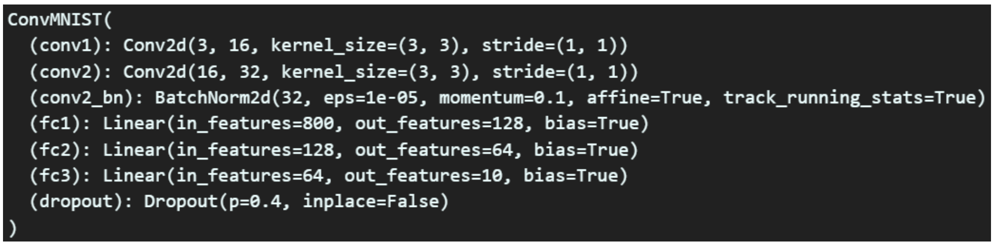
- *Source Code*

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

transform  = transforms.Compose([
                                 transforms.Lambda(lambda      image:
image.convert('RGB')),
                                 transforms.ToTensor(),
                               #  transforms.Resize((64,64)),

transforms.ColorJitter(brightness=(0.5,1.5),          contrast=(1),
saturation=(0.5,1.5), hue=(-0.1,0.1)),
                                 transforms.Normalize(mean=[0.485,
0.456, 0.406], std=[0.229, 0.224, 0.225])

])

train_data = datasets.MNIST(root='./Data', train=True, download=True,
transform=transform)
test_data = datasets.MNIST(root='./Data', train=False, download=True,
transform=transform)
```

```
torch.manual_seed(42)
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=True)

class ConvMNIST(nn.Module):

  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3,16,3,1)
    self.conv2 = nn.Conv2d(16,32,3,1)
    self.conv2_bn = nn.BatchNorm2d(32)
    self.fc1 = nn.Linear(5*5*32, 128)
    self.fc2 = nn.Linear(128, 64)
    self.fc3 = nn.Linear(64, 10)

    self.dropout = nn.Dropout(.4)

  def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    x = F.max_pool2d(x, 2, 2)
    x = self.conv2(x)
    x = self.conv2_bn(x)
    x = F.relu(x)
    x = F.max_pool2d(x, 2, 2)

    x = x.view(-1, 32*5*5)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    x = self.fc3(x)

    return F.log_softmax(x, dim=1)

device = torch.device("cpu")
model = ConvMNIST()
model.to(device)

criterion = nn. NLLLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

import time

start_time = time.time()

epochs = 3
train_losses = []
test_losses = []
train_corrects = []
test_corrects = []

for i in range(epochs):

  train_correct = 0
  for batch, (X_train, y_train) in enumerate(train_loader):
    batch+=1

    X_train = X_train.to(device)
    y_train = y_train.to(device)
    y_pred = model(X_train)
    loss = criterion(y_pred, y_train)

    predicted = torch.max(y_pred.data, 1)[1]
    batch_correct = (predicted==y_train).sum()
    train_correct+=batch_correct

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if batch%256 == 0:
      print(f"Epoch {i+1} Batch {batch} loss: {loss.item()}")

  train_losses.append(loss)
  train_corrects.append(train_correct)

  test_correct = 0
  with torch.no_grad():
    for batch, (X_test, y_test) in enumerate(test_loader):

      X_test = X_test.to(device)
      y_test = y_test.to(device)

      y_val = model(X_test)

      predicted = torch.max(y_val.data,1)[1]

      test_correct+=(predicted==y_test).sum()

  loss = criterion(y_val, y_test)
  test_losses.append(loss)
  test_corrects.append(test_correct)

current_time = time.time()
total = current_time - start_time
print(f"Training finished in {total/60} minutes")

for i in range(len(train_losses)):
  train_losses[i] = train_losses[i].detach().numpy()

for i in range(len(test_losses)):
  test_losses[i] = test_losses[i].detach().numpy()

plt.plot(train_losses, label='train_loss')
plt.plot(test_losses, label='test_loss')
plt.ylabel('Losses')
plt.xlabel('epoch')
plt.xticks(range(0,5))
plt.legend()

test_loader_all = DataLoader(test_data, batch_size=10000,
shuffle=False)

with torch.no_grad():
  correct = 0
  for X_test, y_test in test_loader_all:
    y_val = model(X_test)
    predicted = torch.max(y_val, 1)[1]
    correct+=(predicted==y_test).sum()
```

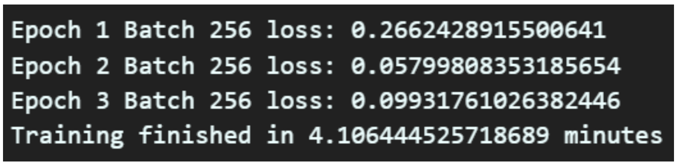
```
print(correct.item()/len(test_data))
print(classification_report(y_test, predicted))
print(confusion_matrix(y_test, predicted))
```

- Arsitektur Model :



```
ConvMNIST(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2_bn): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=800, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=10, bias=True)
  (dropout): Dropout(p=0.4, inplace=False)
)
```

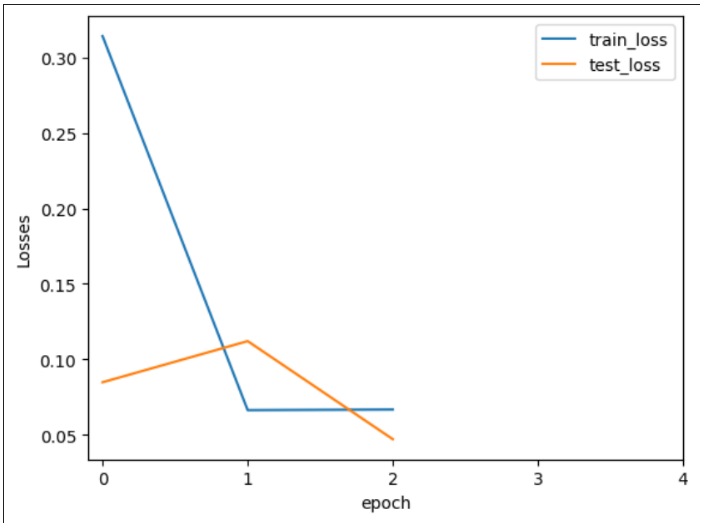
- Proses Pelatihan



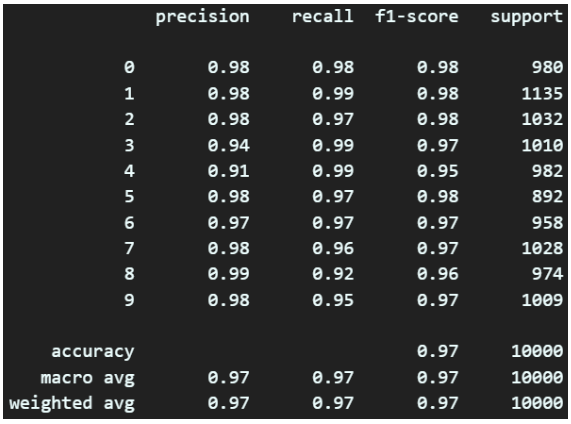
```
Epoch 1 Batch 256 loss: 0.2662428915500641
Epoch 2 Batch 256 loss: 0.05799808353185654
Epoch 3 Batch 256 loss: 0.09931761026382446
Training finished in 4.106444525718689 minutes
```

Waktu Pelatihan : ± 4.1 Menit

- Hasil *Loss* Pelatihan



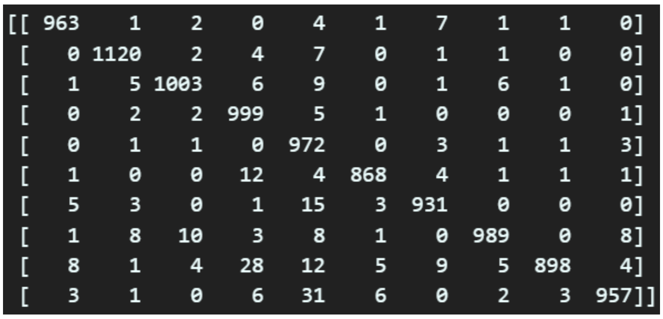
- Hasil Akurasi Terhadap Data *Test*



```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       980
           1       0.98      0.99      0.98      1135
           2       0.98      0.97      0.98      1032
           3       0.94      0.99      0.97      1010
           4       0.91      0.99      0.95       982
           5       0.98      0.97      0.98       892
           6       0.97      0.97      0.97       958
           7       0.98      0.96      0.97      1028
           8       0.99      0.92      0.96       974
           9       0.98      0.95      0.97      1009

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

Akurasi terhadap data uji : 97%

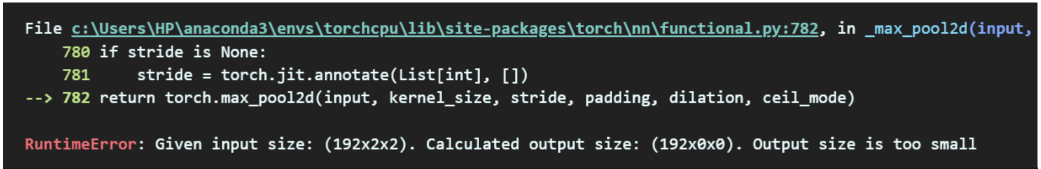
- Hasil *Confusion Matrix*



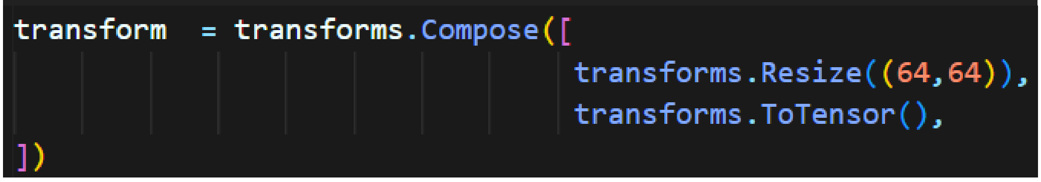
```
[[ 963    1    2    0    4    1    7    1    1    0]
 [   0 1120    2    4    7    0    1    1    0    0]
 [   1    5 1003    6    9    0    1    6    1    0]
 [   0    2    2  999    5    1    0    0    0    1]
 [   0    1    1    0  972    0    3    1    1    3]
 [   1    0    0   12    4  868    4    1    1    1]
 [   5    3    0    1   15    3  931    0    0    0]
 [   1    8   10    3    8    1    0  989    0    8]
 [   8    1    4   28   12    5    9    5  898    4]
 [   3    1    0    6   31    6    0    2    3  957]]
```

2) Model AlexNet + Fine Tune

Terjadi *error* size terlalu kecil pada data MNIST (28x28) saat maxpool sehingga gambar harus diresize menjadi ukuran yang lebih besar.



```
File c:\Users\HP\anaconda3\envs\torchcpu\lib\site-packages\torch\nn\functional.py:782, in _max_pool2d(input,
    780 if stride is None:
    781     stride = torch.jit.annotate(List[int], [])
--> 782 return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)

RuntimeError: Given input size: (192x2x2). Calculated output size: (192x0x0). Output size is too small
```



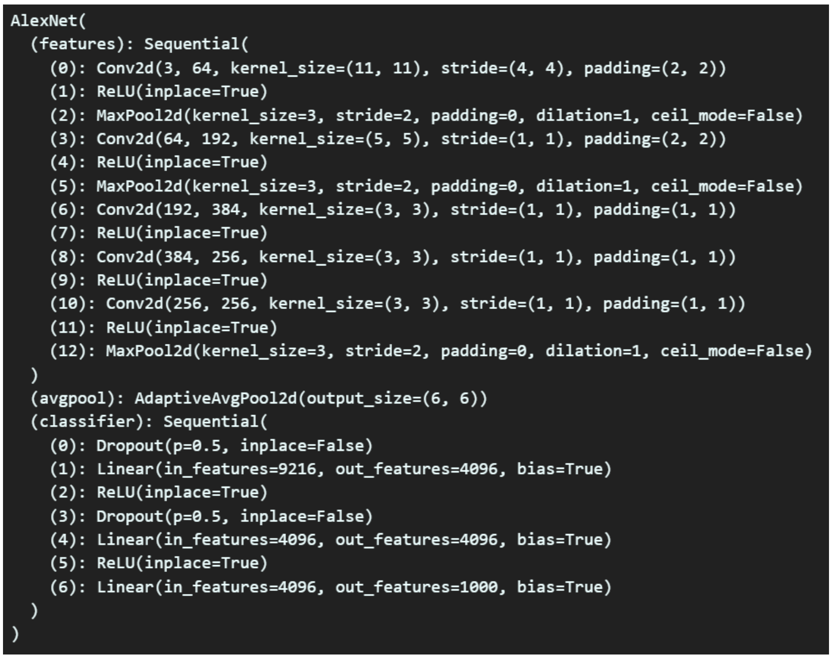
```
transform  = transforms.Compose([
                                transforms.Resize((64,64)),
                                transforms.ToTensor(),
])
```

- *Source Code*

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

transform  = transforms.Compose([
                                transforms.Resize((64,64)),
                                transforms.ToTensor(),
])

train_data = datasets.MNIST(root='./Data', train=True, download=True,
transform=transform)
test_data = datasets.MNIST(root='./Data', train=False, download=True,
transform=transform)

image, label = train_data[0]
print(plt.imshow(image.squeeze(0)))

torch.manual_seed(42)
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=True)

model = torchvision.models.alexnet(pretrained=True)
```

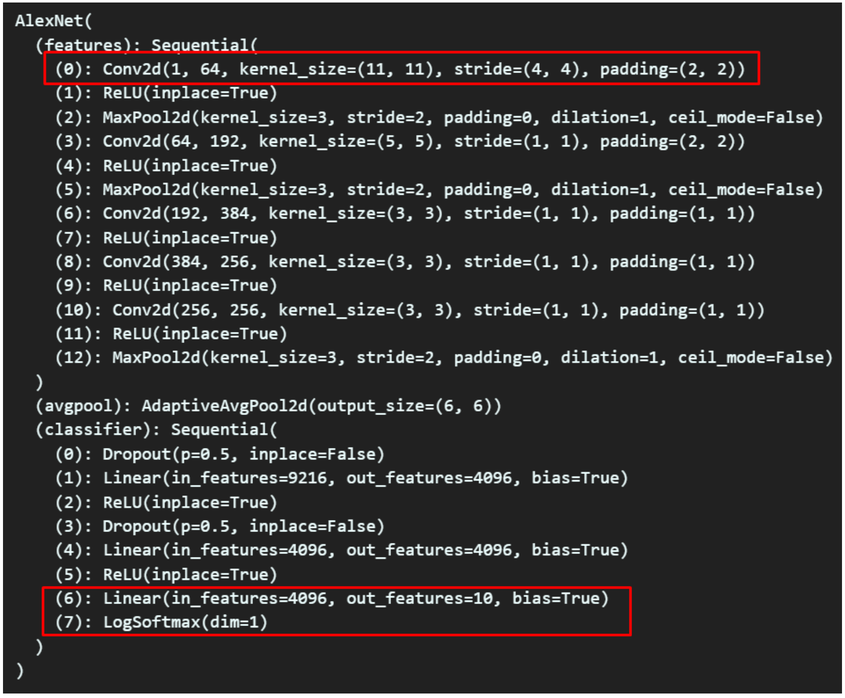
```
# Freeze Layers
for param in model.parameters():
  param.requires_grad = False

# Add tuning layer
model.features[0] = nn.Conv2d(1, 64, kernel_size=(11, 11), stride=(4, 
4), padding=(2, 2))
model.classifier[6] = nn.Linear(4096, 10)
model.classifier.add_module('7', nn.LogSoftmax(dim=1))

for name, param in model.named_parameters():
  print(f"parameter name {name} gradient \t: {param.requires_grad}")

criterion = nn.NLLLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)

device = 'cpu'
model.to(device)

import time
gc.collect()

start_time = time.time()

epochs = 3
train_losses = []
test_losses = []
train_corrects = []
test_corrects = []

for i in range(epochs):

  train_correct = 0
  for batch, (X_train, y_train) in enumerate(train_loader):
    batch+=1

    X_train = X_train.to(device)
    y_train = y_train.to(device)

    y_pred = model(X_train)
    loss = criterion(y_pred, y_train)

    predicted = torch.max(y_pred.data, 1)[1]
    batch_correct = (predicted==y_train).sum()
    train_correct+=batch_correct

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if batch%256 == 0:
      print(f"Epoch {i+1} Batch {batch} loss: {loss.item()}")

  train_losses.append(loss)
  train_corrects.append(train_correct)

  test_correct = 0
  with torch.no_grad():
    for batch, (X_test, y_test) in enumerate(test_loader):
      X_test = X_test.to(device)
      y_test = y_test.to(device)

      y_val = model(X_test)

      predicted = torch.max(y_val.data,1)[1]

      test_correct+=(predicted==y_test).sum()

  loss = criterion(y_val, y_test)
  test_losses.append(loss)
  test_corrects.append(test_correct)

current_time = time.time()
total = current_time - start_time
print(f"Training finished in {total/60} minutes")

for i in range(len(train_losses)):
  train_losses[i] = train_losses[i].cpu().detach().numpy()

for i in range(len(test_losses)):
  test_losses[i] = test_losses[i].cpu().detach().numpy()

print("train loss: ", train_losses)
print("test_loss : ", train_losses)

plt.plot(train_losses, label='train_loss')
plt.plot(test_losses, label='test_loss')
plt.ylabel('Losses')
plt.xlabel('epoch')
plt.xticks(range(0,5))
plt.legend()
print(plt.show())

torch.manual_seed(42)
test_loader_all = DataLoader(test_data, batch_size=10000,
shuffle=False)
test_loader_mini = DataLoader(test_data, batch_size=100,
shuffle=False)
preds = []

with torch.no_grad():
  correct = 0
  for batch, (X_test, y_test) in enumerate(test_loader_mini):
    batch+=1
    X_test = X_test.to(device)
    y_test = y_test.to(device)

    y_val = model(X_test)
    predicted = torch.max(y_val, 1)[1]
    preds.append(predicted)

    correct+=(predicted==y_test).sum()

y_test = test_data.targets
y_test = y_test.numpy()

y_pred = []

for batch_y in preds:
  for y_val in batch_y:
    y_pred.append(y_val)

for i in range(len(y_pred)):
  y_pred[i] = y_pred[i].cpu().detach().numpy()

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

- Arsitektur Model Sebelum *Fine Tune*



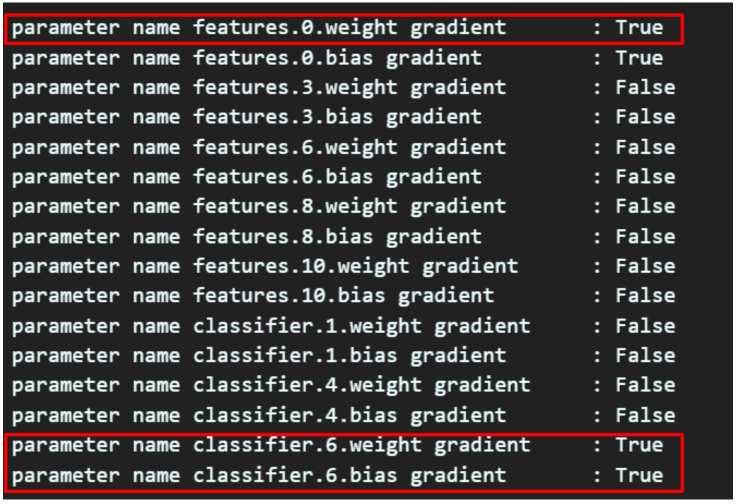
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (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=1000, bias=True)
  )
)
```

- Arsitektur Model Setelah *Fine Tune* (Mengubah jumlah channel input, kelas output dan menambah layer logsoftmax)



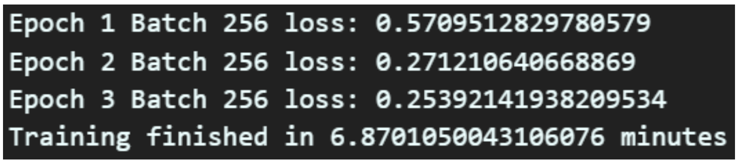
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (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): LogSoftmax(dim=1)
  )
)
```

- Freeze Layer (Hanya layer input dan output yang di update bobotnya / *require grad*)



```
parameter name features.0.weight gradient       : True
parameter name features.0.bias gradient         : True
parameter name features.3.weight gradient       : False
parameter name features.3.bias gradient         : False
parameter name features.6.weight gradient       : False
parameter name features.6.bias gradient         : False
parameter name features.8.weight gradient       : False
parameter name features.8.bias gradient         : False
parameter name features.10.weight gradient      : False
parameter name features.10.bias gradient        : False
parameter name classifier.1.weight gradient     : False
parameter name classifier.1.bias gradient       : False
parameter name classifier.4.weight gradient     : False
parameter name classifier.4.bias gradient       : False
parameter name classifier.6.weight gradient     : True
parameter name classifier.6.bias gradient       : True
```

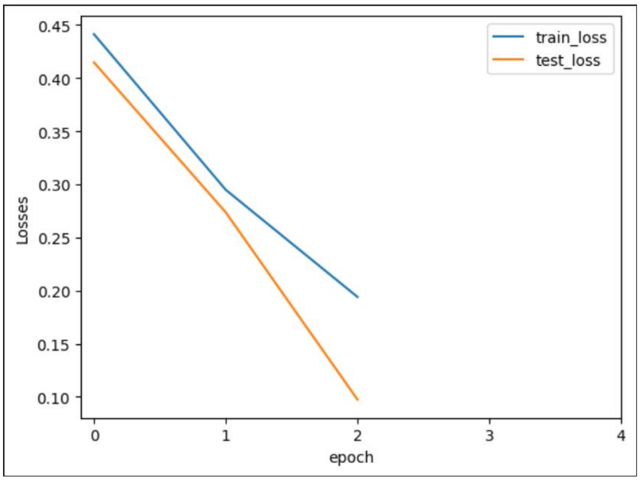
- Proses Pelatihan



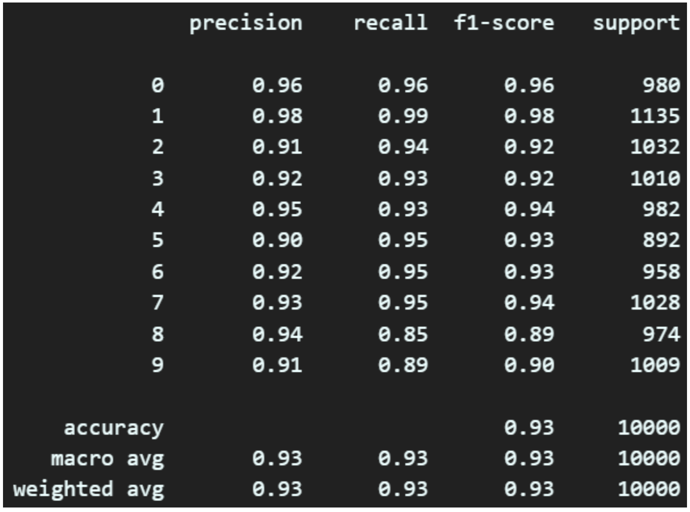
```
Epoch 1 Batch 256 loss: 0.5709512829780579
Epoch 2 Batch 256 loss: 0.271210640668869
Epoch 3 Batch 256 loss: 0.25392141938209534
Training finished in 6.8701050043106076 minutes
```

Waktu pelatihan: ± 7 Menit

- Hasil *Loss* Pelatihan



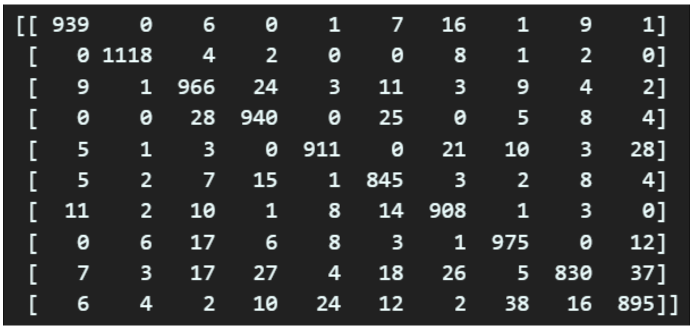
- Hasil Akurasi Terhadap Data *Test*



```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       980
           1       0.98      0.99      0.98      1135
           2       0.91      0.94      0.92      1032
           3       0.92      0.93      0.92      1010
           4       0.95      0.93      0.94       982
           5       0.90      0.95      0.93       892
           6       0.92      0.95      0.93       958
           7       0.93      0.95      0.94      1028
           8       0.94      0.85      0.89       974
           9       0.91      0.89      0.90      1009

    accuracy                           0.93     10000
   macro avg       0.93      0.93      0.93     10000
weighted avg       0.93      0.93      0.93     10000
```

Hasil akurasi terhadap data uji : 93%

- Hasil *Confusion Matrix*



```
[[ 939    0    6    0    1    7   16    1    9    1]
 [   0 1118    4    2    0    0    8    1    2    0]
 [   9    1  966   24    3   11    3    9    4    2]
 [   0    0   28  940    0   25    0    5    8    4]
 [   5    1    3    0  911    0   21   10    3   28]
 [   5    2    7   15    1  845    3    2    8    4]
 [  11    2   10    1    8   14  908    1    3    0]
 [   0    6   17    6    8    3    1  975    0   12]
 [   7    3   17   27    4   18   26    5  830   37]
 [   6    4    2   10   24   12    2   38   16  895]]
```