**Lampiran 3 Source Code**

Berikut adalah source code dalam pembuatan model pada jupyter:

❖ **Instalasi PyTorch**

```
pip install torch torchvision

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt
import time
import os
import copy
import torchvision
```

❖ **Tentukan Device (GPU atau CPU)**

```
device = torch.device("cuda:0" if
torch.cuda.is_available() else "cpu")
```

❖ **Data Augmentation dan Normalisasi**

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485,   0.456,   0.406],
[0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485,   0.456,   0.406],
[0.229, 0.224, 0.225])
    ]),
}

data_dir = 'dataset'
image_datasets = {x:
datasets.ImageFolder(os.path.join(data_dir, x),
```

```
data_transforms[x]
                  for x in ['train', 'test']}
dataloaders    =    {x:    DataLoader(image_datasets[x],
batch_size=4, shuffle=True, num_workers=4)
                for x in ['train', 'test']}
dataset_sizes  =  {x:  len(image_datasets[x])  for  x  in
['train', 'test']}
class_names = image_datasets['train'].classes
```

❖ **Visualisasi Beberapa Gambar dari Dataset**

```
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are
updated
```

❖ **Mendapatkan batch gambar**

```
inputs, classes = next(iter(dataloaders['train']))
```

❖ **Membuat grid dari gambar**

```
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```

❖ **Load Model ResNet50 Pretrained dan Modifikasi Layer Akhir**

```
model_ft = models.resnet50(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)  # 2 kelas (on_time
dan not_on_time)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()
```

❖ **Menggunakan SGD dengan momentum**

```
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
```

❖ **Scheduler untuk Mengurangi Learning Rate**

```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

❖ **Training Model**

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Setiap epoch memiliki fase training dan fase testing
        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()  # Set model ke mode training
            else:
                model.eval()   # Set model ke mode evaluasi

            running_loss = 0.0
            running_corrects = 0

            # Iterasi melalui data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # Zero the parameter gradients
                optimizer.zero_grad()

                # Forward
                # Track history only in train
                with torch.set_grad_enabled(phase == 'train'):
```

```
outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # Backward + optimize only if in
training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # Statistics
                running_loss    +=    loss.item()    *
inputs.size(0)
                running_corrects += torch.sum(preds ==
labels.data)

            if phase == 'train':
                scheduler.step()

            epoch_loss      =      running_loss      /
dataset_sizes[phase]
            epoch_acc  =  running_corrects.double()  /
dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # Deep copy the model
            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts                          =
copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best test Acc: {:4f}'.format(best_acc))

    # Load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

❖ **Jalankan Training**

```
model_ft     =     train_model(model_ft,     criterion,
optimizer_ft, exp_lr_scheduler, num_epochs=25)
```

❖ **Evaluasi Model**

```
def evaluate_model(model, dataloaders, class_names, device):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in dataloaders['test']:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return all_preds, all_labels

preds, labels = evaluate_model(model_ft, dataloaders, class_names, device)
```

❖ **Visualisasi Hasil Evaluasi**

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(labels, preds, target_names=class_names))
print(confusion_matrix(labels, preds))

# Plot confusion matrix
import seaborn as sns
sns.heatmap(confusion_matrix(labels, preds), annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

❖ **Simpan Model**

```
torch.save(model_ft.state_dict(), 'resnet50_signature_model.pth')
print("Model telah disimpan.")
```