LAMPIRAN

```python
# Lampiran 1: Kelas Dataset
class Human36Dataset(Dataset):
    def __init__(self, actions,
        data_path, is_train=True):
        self.actions, self.data_path
            , self.is_train =
            actions, data_path,
            is_train
        self.inp_list, self.out_list
            , self.key_list = [],
            [], []

        if self.is_train:
            self.data_2d = torch.
                load(data_path/'
                train_2d.pt')
            self.data_3d = torch.
                load(data_path/'
                train_3d.pt')
        else:
            self.data_2d = torch.
                load(data_path/'
                test_2d.pt')
            self.data_3d = torch.
                load(data_path/'
                test_3d.pt')

        for key in self.data_2d.keys
            ():
            assert self.data_2d[key
                ].shape[0] == self.
                data_3d[key].shape
                [0]
            num_file = self.data_2d[
                key].shape[0]
            for i in range(num_file)
                :
                self.inp_list.append
                    (self.data_2d[
                    key][i])
                self.out_list.append
                    (self.data_3d[
                    key][i])
                self.key_list.append
                    (key)

    def __getitem__(self, idx):
        inp = torch.from_numpy(self.
            inp_list[idx]).float()
        out = torch.from_numpy(self.
            out_list[idx]).float()
        return inp, out

    def get_key(self, idx):
        return self.key_list[idx]

    def __len__(self):
        return len(self.inp_list)

# Lampiran 2: Kelas ResLinear
class ResLinear(nn.Module):
    def __init__(self, size, pd=0.5)
        :
        super().__init__()
        self.size = size
        self.relu = nn.ReLU(inplace=
            True)
        self.drop = nn.Dropout(pd)
        # learnable
        self.ln1 = nn.Linear(self.
            size, self.size)
        self.bn2 = nn.BatchNorm1d(
            self.size)
        self.ln3 = nn.Linear(self.
            size, self.size)
        self.bn4 = nn.BatchNorm1d(
            self.size)
    def forward(self, x):
        y = self.drop(self.relu(self
            .bn2(self.ln1(x))))
        y = self.drop(self.relu(self
            .bn4(self.ln3(y))))
        return x + y

# Lampiran 3: Arsitektur Model
class Model(nn.Module):
    def __init__(self, size=1024,
        num_res_lyr=2, pd=0.5):
        super().__init__()
        self.size, self.num_res_lyr,
             self.pd = size,
            num_res_lyr, pd
        self.input_size, self.
            output_size = 32, 48
        self.relu = nn.ReLU(inplace=
            True)
        self.drop = nn.Dropout(self.
            pd)

        # input size
        self.ln_in = nn.Linear(self.
            input_size, self.size)
        self.bn_in = nn.BatchNorm1d(
            self.size)

        # res layers
        self.lins = []
        for i in range(num_res_lyr):
            self.lins.append(
                ResLinear(self.size,
                 self.pd))
        self.lins = nn.ModuleList(
            self.lins)

        # output size
        self.ln_out = nn.Linear(self
            .size, self.output_size)
    def forward(self, x):
        y = self.drop(self.relu(self
            .bn_in(self.ln_in(x))))
        for i in range(self.
            num_res_lyr):
            y = self.lins[i](y)
        y = self.ln_out(y)
```

```
        return y

# Lampiran 4: Options
class Options():
    def __init__(self):
        # paths
        self.data_path = Path('data'
            )
        self.model_path = Path('
            model')

        # train options
        self.actions = 'All'
        self.attempt_id = '01'
        self.attempt_path = Path('
            model')/self.attempt_id

        self.load_ckpt = False

        # train hyper-params
        self.bs = 128
        self.epochs = 10
        self.lr = 1e-3

        # model hyper-params
        self.size = 1024
        self.stages = 2
        self.dropout = 0.5

# Lampiran 5: Pemuatan Data
stat_3d = torch.load(data_path/'
    stat_3d.pt')
stat_2d = torch.load(data_path/'
    stat_2d.pt')
rcams = torch.load(data_path/'rcams.
    pt')

mean_2d = stat_2d['mean']
std_2d = stat_2d['std']
dim_use_2d = stat_2d['dim_use']
dim_ignore_2d = stat_2d['dim_ignore'
    ]

mean_3d = stat_3d['mean']
std_3d = stat_3d['std']
dim_use_3d = stat_3d['dim_use']
dim_ignore_3d = stat_3d['dim_ignore'
    ]

# Lampiran 6: Instansiasi Dataset
    dan DataLoader
train_ds = Human36Dataset(
    get_actions(options.actions),
    options.data_path, is_train=True
    )
train_dl = DataLoader(train_ds,
    batch_size=options.bs, shuffle=
    True)
test_ds = Human36Dataset(get_actions
    (options.actions), options.
    data_path, is_train=False)
test_dl = DataLoader(test_ds,
    batch_size=options.bs, shuffle=
    False)

# Lampiran 7: Algoritma Pelatihan
def train(train_dl, model, criterion
    , optimizer, options, mb):
    model.train()
    loss_list = []
    skel_loss_list = []
    for xb, yb in progress_bar(
        train_dl, parent=mb):
        xb, yb = xb.cuda(), yb.cuda
            ()
        yhat = model(xb)
        optimizer.zero_grad()
        loss_skel = criterion(yhat,
            yb)
        loss = loss_skel.mean()
        loss.backward()
        nn.utils.clip_grad_norm_(
            model.parameters(),
            max_norm=1)
        optimizer.step()

        loss_list.append(loss.item()
            )
        skel_loss_list.append(
            loss_skel.data.cpu().
            numpy())

        mb.child.comment = f'train␣
            loss:␣{loss.item()}'
    return loss_list, skel_loss_list

# Lampiran 8: Algoritma Validasi
def test(test_dl, model, criterion,
    options, mb):
    model.eval()
    loss_list = []
    skel_loss_list = []
    for xb, yb in progress_bar(
        test_dl, parent=mb):
        xb, yb = xb.cuda(), yb.cuda
            ()
        with torch.no_grad():
            yhat = model(xb)
            loss_skel = criterion(
                yhat, yb)
            loss = loss_skel.mean()
        loss_list.append(loss.item()
            )
        skel_loss_list.append(
            loss_skel.data.cpu().
            numpy())
        mb.child.comment = f'test␣
            loss:␣{loss.item()}'
    return loss_list, skel_loss_list

# Lampiran 9: Instansiasi Model
model = Model()
model = model.cuda()
model.apply(init_kaiming)
print(f'total␣params:␣{sum(p.numel()
    ␣for␣p␣in␣model.parameters())}')

criterion = nn.MSELoss(reduction='
    none').cuda()
optimizer = optim.Adam(model.
    parameters(), lr=options.lr)

if options.load_ckpt:
    options = torch.load('model/01/
```

```
        options.pt')
    model_state = torch.load(options
        .attempt_path/'last_model.pt
        ')
    optimizer_state = torch.load(
        options.attempt_path/'
        last_optimizer.pt')
    model.load_state_dict(
        model_state)
    optimizer.load_state_dict(
        optimizer_state)

# Lampiran 10: Visualisasi Titik
    Kunci
key = train_key_list[184]
plt.figure(figsize=(16,6))
gs2 = GridSpec(1,2)
ax1 = plt.subplot(gs2[0])
ax2 = plt.subplot(gs2[1], projection
    ='3d')

idx = 200

ts_2d = utils.unnormalize_data(
    train_set_2d[key][idx], mean_2d,
     std_2d, dim_ignore_2d)[0]

ts_3d = utils.unnormalize_data(
    train_set_3d[key][idx], mean_3d,
     std_3d, dim_ignore_3d)[0]
ts_3d = utils.cam_to_world_centered(
    ts_3d, key, rcams)

utils.show_2d_pose(ts_2d, ax1)
ax1.invert_yaxis()

utils.show_3d_pose(ts_3d, ax2)

plt.show()

# Lampiran 11: Visualisasi Inferensi
## %matplotlib qt
fig = plt.figure(figsize=(15,15))
gs = GridSpec(2, 2)
ax0 = plt.subplot(gs[0])
ax1 = plt.subplot(gs[1])
ax2 = plt.subplot(gs[2])
ax3 = plt.subplot(gs[3], projection=
    '3d')
ax3.view_init(elev=20, azim=70)

img0 = None
img1 = None
for i in range(len(img_lists)):
# for i in range(3):
    if img0 is None:
        img0 = ax0.imshow(img_ls[i])
    else:
        img0.set_data(img_ls[i])

    if img1 is None:
        img1 = ax1.imshow(out_ls[i])
    else:
        img1.set_data(out_ls[i])

    ax2.clear()
    show_2d_pose(kp_ls[i], ax2)
    ax2.invert_yaxis()

    ax3.clear()
    show_3d_pose(kp3d_ls[i], ax3)

    plt.pause(1e-25)
    plt.draw()
    plt.savefig(f'imgs/asd/bro{i}.
        jpg')

# Lampiran 12: Plot Grafik
train_loss_lists = []
train_mean = []
train_max = []
train_min = []
for i in range(options.epochs):
    tll = torch.load(options.
        attempt_path/f'
        train_loss_list_e{i}.pt')
    train_mean.append(np.mean(tll))
    train_max.append(np.max(tll))
    train_min.append(np.min(tll))
    train_loss_lists.append(tll)
test_loss_lists = []
test_mean = []
test_max = []
test_min = []
for i in range(options.epochs):
    tll = torch.load(options.
        attempt_path/f'
        test_loss_list_e{i}.pt')
    test_mean.append(np.mean(tll))
    test_max.append(np.max(tll))
    test_min.append(np.min(tll))
    test_loss_lists.append(tll)
plt.ylabel("Rata-Rata Kesalahan")
plt.xlabel("Epoch")

plt.plot(train_mean, label="
    pelatihan", linewidth=1)
plt.plot(test_mean, label="validasi"
    , linewidth=1)
plt.xticks(range(0, 10))

plt.legend()
plt.savefig("brrr.jpg", dpi=500)
plt.show()
```