UNet starter for Steel defect detection challenge

Nama Kelompok :
1. Ari Cahya Saputra (1103190093)
2. Alvin Anandra Brilliyandy (1103190111)
Kelas : TK-42-PIL

Pengertian Resnet

ResNet atau Residual Network adalah jenis arsitektur Convolutional Neural Network (CNN) dengan menggunakan model yang sudah dilatih sebelumnya

Import Library

```
import os
import cv2
import pdb
import time
import warnings
import random
import numpy as np
import pandas as pd
from tqdm import tqdm_notebook as tqdm
from torch.optim.lr_scheduler import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
from torch.nn import functional as F
import torch.optim as optim
import torch.backends.cudnn as cudnn
from torch.utils.data import DataLoader, Dataset, sampler
from matplotlib import pyplot as plt
from albumentations import (HorizontalFlip, ShiftScaleRotate, Normalize, Resize, Compose, GaussNoise)
from albumentations.pytorch import ToTensor
warnings.filterwarnings("ignore")
seed = 69
random.seed(seed)
os.environ["PYTHONHASHSEED"] = str(seed)
np.random.seed(seed)
torch.cuda.manual_seed(seed)
torch.backends.cudnn.deterministic = True
```

- Library yang kami gunakan adalah :
- Pandas
- CV2
- Numpy
- Pytorch
- SKLearn
- Matplotlib

Training The Data

```python
class Trainer(object):
    '''This class takes care of training and validation of our model'''
    def __init__(self, model):
        self.num_workers = 6
        self.batch_size = {"train": 4, "val": 4}
        self.accumulation_steps = 32 // self.batch_size['train']
        self.lr = 5e-4
        self.num_epochs = 20
        self.best_loss = float("inf")
        self.phases = ["train", "val"]
        self.device = torch.device("cuda:0")
        torch.set_default_tensor_type("torch.cuda.FloatTensor")
        self.net = model
        self.criterion = torch.nn.BCEWithLogitsLoss()
        self.optimizer = optim.Adam(self.net.parameters(), lr=self.lr)
        self.scheduler = ReduceLROnPlateau(self.optimizer, mode="min", patience=3, verbose=True)
        self.net = self.net.to(self.device)
        cudnn.benchmark = True
        self.dataloaders = {
            phase: provider(
                data_folder=data_folder,
                df_path=train_df_path,
                phase=phase,
                mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225),
                batch_size=self.batch_size[phase],
                num_workers=self.num_workers,
            )
            for phase in self.phases
        }
        self.losses = {phase: [] for phase in self.phases}
        self.iou_scores = {phase: [] for phase in self.phases}
        self.dice_scores = {phase: [] for phase in self.phases}

    def forward(self, images, targets):
        images = images.to(self.device)
        masks = targets.to(self.device)
        outputs = self.net(images)
        loss = self.criterion(outputs, masks)
        return loss, outputs
```

```python
def iterate(self, epoch, phase):
    meter = Meter(phase, epoch)
    start = time.strftime("%H:%M:%S")
    print(f"Starting epoch: {epoch} | phase: {phase} | ⏰: {start}")
    batch_size = self.batch_size[phase]
    self.net.train(phase == "train")
    dataloader = self.dataloaders[phase]
    running_loss = 0.0
    total_batches = len(dataloader)
      tk0 = tqdm(dataloader, total=total_batches)
    self.optimizer.zero_grad()
    for itr, batch in enumerate(dataloader): # replace `dataloader` with `tk0` for tqdm
        images, targets = batch
        loss, outputs = self.forward(images, targets)
        loss = loss / self.accumulation_steps
        if phase == "train":
            loss.backward()
            if (itr + 1 ) % self.accumulation_steps == 0:
                self.optimizer.step()
                self.optimizer.zero_grad()
        running_loss += loss.item()
        outputs = outputs.detach().cpu()
        meter.update(targets, outputs)
          tk0.set_postfix(loss=(running_loss / ((itr + 1))))
    epoch_loss = (running_loss * self.accumulation_steps) / total_batches
    dice, iou = epoch_log(phase, epoch, epoch_loss, meter, start)
    self.losses[phase].append(epoch_loss)
    self.dice_scores[phase].append(dice)
    self.iou_scores[phase].append(iou)
    torch.cuda.empty_cache()
    return epoch_loss

def start(self):
    for epoch in range(self.num_epochs):
        self.iterate(epoch, "train")
        state = {
            "epoch": epoch,
            "best_loss": self.best_loss,
            "state_dict": self.net.state_dict(),
            "optimizer": self.optimizer.state_dict(),
        }
    with torch.no_grad():
        val_loss = self.iterate(epoch, "val")
        self.scheduler.step(val_loss)
    if val_loss < self.best_loss:
        print("******** New optimal found, saving state ********")
        state["best_loss"] = self.best_loss = val_loss
        torch.save(state, "./model.pth")
    print()
```

- Dataset sudah kami masukkan ke dalam model dan kemudian siap untuk proses training model.

Plot Training

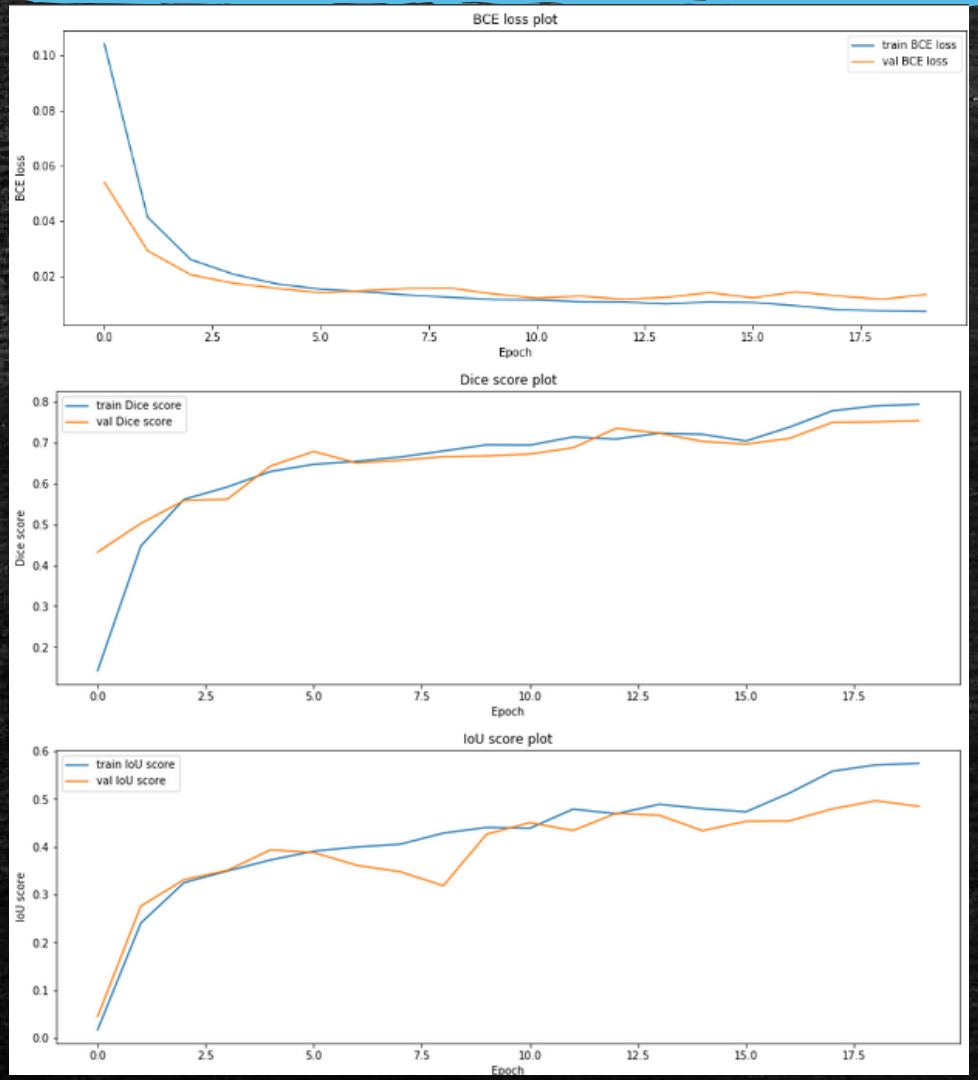
```
# PLOT TRAINING
losses = model_trainer.losses
dice_scores = model_trainer.dice_scores # overall dice
iou_scores = model_trainer.iou_scores

def plot(scores, name):
    plt.figure(figsize=(15,5))
    plt.plot(range(len(scores["train"])), scores["train"], label=f'train {name}')
    plt.plot(range(len(scores["train"])), scores["val"], label=f'val {name}')
    plt.title(f'{name} plot'); plt.xlabel('Epoch'); plt.ylabel(f'{name}');
    plt.legend();
    plt.show()

plot(losses, "BCE loss")
plot(dice_scores, "Dice score")
plot(iou_scores, "IoU score")
```

- Pada proses ini model sudah selesai di train dan dapat kami visualisasikan agar lebih mudah untuk dimengerti hasilnya.

Hasil



- Dapat dilihat bahwa BCE Score kami terus mengalami penurunan sehingga hasil yang didapatkan lebih akurat, begitu juga DICE Score dan IoU Score kami menjadi naik. Hal ini menandakan bahwa model yang telah kami train tergolong baik.

Terima Kasih !!!