이미지 분류 문제 해결 학습

(주)인피닉스 – 강호용 연구원

Infinyx

01

이미지 분류 문제 해결 파트 -4

Infinyx
THINK LIFE SYNC AI

이미지 분류 문제 해결 학습 – 인퍼런스 해보기

ex_01 학습 데이터 : 음식 20가지 데이터 세트

Ex_02 학습 데이터 : **철강 데이터 (오염 및 데미지 손상)**

제공되는 두개 .PT 파일 학습 모델 : resnet50

기존 학습 코드를 Class 형태로 변경하기

```python
class Classifier:
    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = None
        self.train_losses = []
        self.val_losses = []
        self.train_accs = []
        self.val_accs = []

    def train(self, train_loader, val_loader, epochs, optimizer, criterion):
        best_val_acc = 0.0
        print("Train....")

        for epoch in range(epochs):
            train_loss = 0.0
            val_loss = 0.0
            train_acc = 0.0
            val_acc = 0.0

            self.model.train()
            train_loader_iter = tqdm(train_loader, desc=(f"Epoch: {epoch + 1}/{epochs}"), leave=False)

            for i, (data, target) in enumerate(train_loader_iter):
                data, target = data.float().to(self.device), target.to(self.device)

                optimizer.zero_grad()
                outputs = self.model(data)
                loss = criterion(outputs, target)
                loss.backward()
                optimizer.step()

                train_loss += loss.item()

                _, pred = torch.max(outputs, 1)
                train_acc += (pred == target).sum().item()

                train_loader_iter.set_postfix({"Loss": loss.item()})

            train_loss /= len(train_loader)
            train_acc = train_acc / len(train_loader.dataset)

            self.model.eval()
            with torch.no_grad():
                for data, target in val_loader:
                    data, target = data.float().to(self.device), target.to(self.device)
                    output = self.model(data)
                    pred = output.argmax(dim=1, keepdim=True)
                    val_acc += pred.eq(target.view_as(pred)).sum().item()
                    val_loss += criterion(output, target).item()

            val_loss /= len(val_loader)
            val_acc = val_acc / len(val_loader.dataset)

            self.train_losses.append(train_loss)
            self.train_accs.append(train_acc)
            self.val_losses.append(val_loss)
            self.val_accs.append(val_acc)

            if val_acc > best_val_acc:
                torch.save(self.model.state_dict(), "./ex01_0717_resnet50_best.pt")
                best_val_acc = val_acc

            print(f"Epoch [{epoch + 1}/{epochs}], Train loss: {train_loss:.4f}, "
                  f"Val loss: {val_loss:.4f}, Train ACC: {train_acc:.4f}, Val ACC: {val_acc:.4f}")

        torch.save(self.model.state_dict(), "./ex01_0717_resnet50_last.pt")
        # After training is complete:
        self.save_results_to_csv()
        self.plot_loss()
        self.plot_accuracy()
```

기존 학습 코드를 Class 형태로 변경하기

```python
def save_results_to_csv(self):
    df = pd.DataFrame({
        'Train Loss': self.train_losses,
        'Train Accuracy': self.train_accs,
        'Validation Loss': self.val_losses,
        'Validation Accuracy': self.val_accs
    })
    df.to_csv('train_val_results_ex01.csv', index=False)

def plot_loss(self):
    plt.figure()
    plt.plot(self.train_losses, label='Train Loss')
    plt.plot(self.val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('ex01_loss_plot.png')

def plot_accuracy(self):
    plt.figure()
    plt.plot(self.train_accs, label='Train Accuracy')
    plt.plot(self.val_accs, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig('ex01_accuracy_plot.png')
```

```python
def run(self):
    self.model = resnet50(pretrained=True)
    self.model.fc = nn.Linear(2048, 20)
    self.model.to(self.device)

    train_transforms = A.Compose([
        A.SmallestMaxSize(max_size=250),
        A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.6),
        A.RandomShadow(),
        A.RGBShift(r_shift_limit=15, g_shift_limit=15, b_shift_limit=15, p=0.4),
        A.RandomBrightnessContrast(p=0.5),
        A.Resize(height=224, width=224),
        ToTensorV2()
    ])

    val_transforms = A.Compose([
        A.SmallestMaxSize(max_size=250),
        A.Resize(height=224, width=224),
        ToTensorV2()
    ])

    train_dataset = MyFoodDataset("./food_dataset/train/", transform=train_transforms)
    val_dataset = MyFoodDataset("./food_dataset/validation/", transform=val_transforms)

    train_loader = DataLoader(train_dataset, batch_size=164, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=164, shuffle=False)

    epochs = 100
    criterion = CrossEntropyLoss().to(self.device)
    optimizer = AdamW(self.model.parameters(), lr=0.001, weight_decay=1e-2)

    self.train(self.model, train_loader, val_loader, epochs, optimizer, criterion)


if __name__ == "__main__":
    classifier = Classifier()
    classifier.run()
```

기존 학습 코드를 Class 형태로 변경 후 사용자 입력 값 추가하기

```python
import argparse

class Classifier:
    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = None
        self.train_losses = []
        self.val_losses = []
        self.train_accs = []
        self.val_accs = []

    def train(self, train_loader, val_loader, epochs, optimizer, criterion):
        best_val_acc = 0.0
        print("Train....")

        for epoch in range(epochs):
            train_loss = 0.0
            val_loss = 0.0
            train_acc = 0.0
            val_acc = 0.0

            self.model.train()
            train_loader_iter = tqdm(train_loader, desc=(f"Epoch: {epoch + 1}/{epochs}"), leave=False)

            for i, (data, target) in enumerate(train_loader_iter):
                data, target = data.float().to(self.device), target.to(self.device)

                optimizer.zero_grad()
                outputs = self.model(data)
                loss = criterion(outputs, target)
                loss.backward()
                optimizer.step()

                train_loss += loss.item()

                _, pred = torch.max(outputs, 1)
                train_acc += (pred == target).sum().item()

                train_loader_iter.set_postfix({"Loss": loss.item()})

            train_loss /= len(train_loader)
            train_acc = train_acc / len(train_loader.dataset)
```

```python
            self.model.eval()
            with torch.no_grad():
                for data, target in val_loader:
                    data, target = data.float().to(self.device), target.to(self.device)
                    output = self.model(data)
                    pred = output.argmax(dim=1, keepdim=True)
                    val_acc += pred.eq(target.view_as(pred)).sum().item()
                    val_loss += criterion(output, target).item()

            val_loss /= len(val_loader)
            val_acc = val_acc / len(val_loader.dataset)

            self.train_losses.append(train_loss)
            self.train_accs.append(train_acc)
            self.val_losses.append(val_loss)
            self.val_accs.append(val_acc)

            if val_acc > best_val_acc:
                torch.save(self.model.state_dict(), "./ex01_0717_resnet50_best.pt")
                best_val_acc = val_acc

            print(f"Epoch [{epoch + 1}/{epochs}], Train loss: {train_loss:.4f}, "
                  f"Val loss: {val_loss:.4f}, Train ACC: {train_acc:.4f}, Val ACC: {val_acc:.4f}")

        torch.save(self.model.state_dict(), "./ex01_0717_resnet50_last.pt")
        # After training is complete:
        self.save_results_to_csv()
        self.plot_loss()
        self.plot_accuracy()
```

기존 학습 코드를 Class 형태로 변경 후 사용자 입력 값 추가하기

```python
def save_results_to_csv(self):
    df = pd.DataFrame({
        'Train Loss': self.train_losses,
        'Train Accuracy': self.train_accs,
        'Validation Loss': self.val_losses,
        'Validation Accuracy': self.val_accs
    })
    df.to_csv('train_val_results_ex01.csv', index=False)

def plot_loss(self):
    plt.figure()
    plt.plot(self.train_losses, label='Train Loss')
    plt.plot(self.val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('ex01_loss_plot.png')

def plot_accuracy(self):
    plt.figure()
    plt.plot(self.train_accs, label='Train Accuracy')
    plt.plot(self.val_accs, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig('ex01_accuracy_plot.png')
```

```python
def run(self, args):
    self.model = resnet50(pretrained=True)
    self.model.fc = nn.Linear(2048, 20)
    self.model.to(self.device)

    train_transforms = A.Compose([
        A.SmallestMaxSize(max_size=250),
        A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.6),
        A.RandomShadow(),
        A.RGBShift(r_shift_limit=15, g_shift_limit=15, b_shift_limit=15, p=0.4),
        A.RandomBrightnessContrast(p=0.5),
        A.Resize(height=224, width=224),
        ToTensorV2()
    ])

    val_transforms = A.Compose([
        A.SmallestMaxSize(max_size=250),
        A.Resize(height=224, width=224),
        ToTensorV2()
    ])

    train_dataset = MyFoodDataset(args.train_dir, transform=train_transforms)
    val_dataset = MyFoodDataset(args.val_dir, transform=val_transforms)

    train_loader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=args.batch_size, shuffle=False)

    epochs = args.epochs
    criterion = CrossEntropyLoss().to(self.device)
    optimizer = AdamW(self.model.parameters(), lr=args.learning_rate, weight_decay=args.weight_decay)

    self.train(self.model, train_loader, val_loader, epochs, optimizer, criterion)
```

기존 학습 코드를 Class 형태로 변경 후 사용자 입력 값 추가하기

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--train_dir', type=str, default="./food_dataset/train/",
                        help='directory path to the training dataset')
    parser.add_argument('--val_dir', type=str, default="./food_dataset/validation/",
                        help='directory path to the validation dataset')
    parser.add_argument('--epochs', type=int, default=100,
                        help='number of epochs for training')
    parser.add_argument('--batch_size', type=int, default=164,
                        help='batch size for training and validation')
    parser.add_argument('--learning_rate', type=float, default=0.001,
                        help='learning rate for optimizer')
    parser.add_argument('--weight_decay', type=float, default=1e-2,
                        help='weight decay for optimizer')
    args = parser.parse_args()

    classifier = Classifier()
    classifier.run(args)
```

이미지 분류 문제 해결 학습
>> 데이터 소개

US license plates- Image Classification

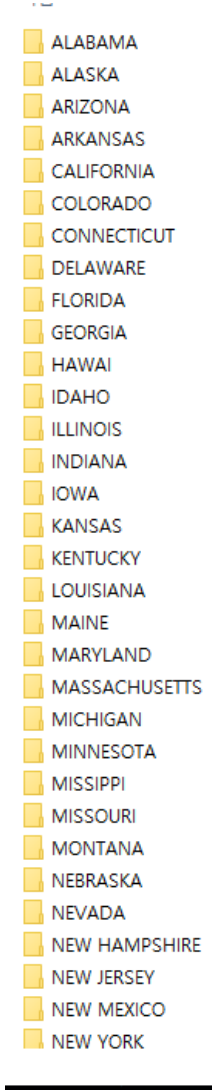
50개 주 전체의 자동차 번호판 이미지 데이터 세트입니다.

모든 이미지는 jpg 형식의 128 X 224 X 3 크기입니다.

EfficientNetB1 모델도 포함되어 있습니다.
- > 20 에포크 동안 97.9%

이미지 분류 문제 해결 학습 – 학습된 모델 (정보를 csv 에 저장하기)

- 데이터 소개 (폴더 구성)



이미지 분류 문제 해결 학습
>> 데이터 소개

Pneumonia_dataset – 흑색 데이터 학습 하기

- 정상
- 세균성 폐렴
- 바이러스성 폐렴

Pneumonia_dataset – 흑색 데이터 학습 하기

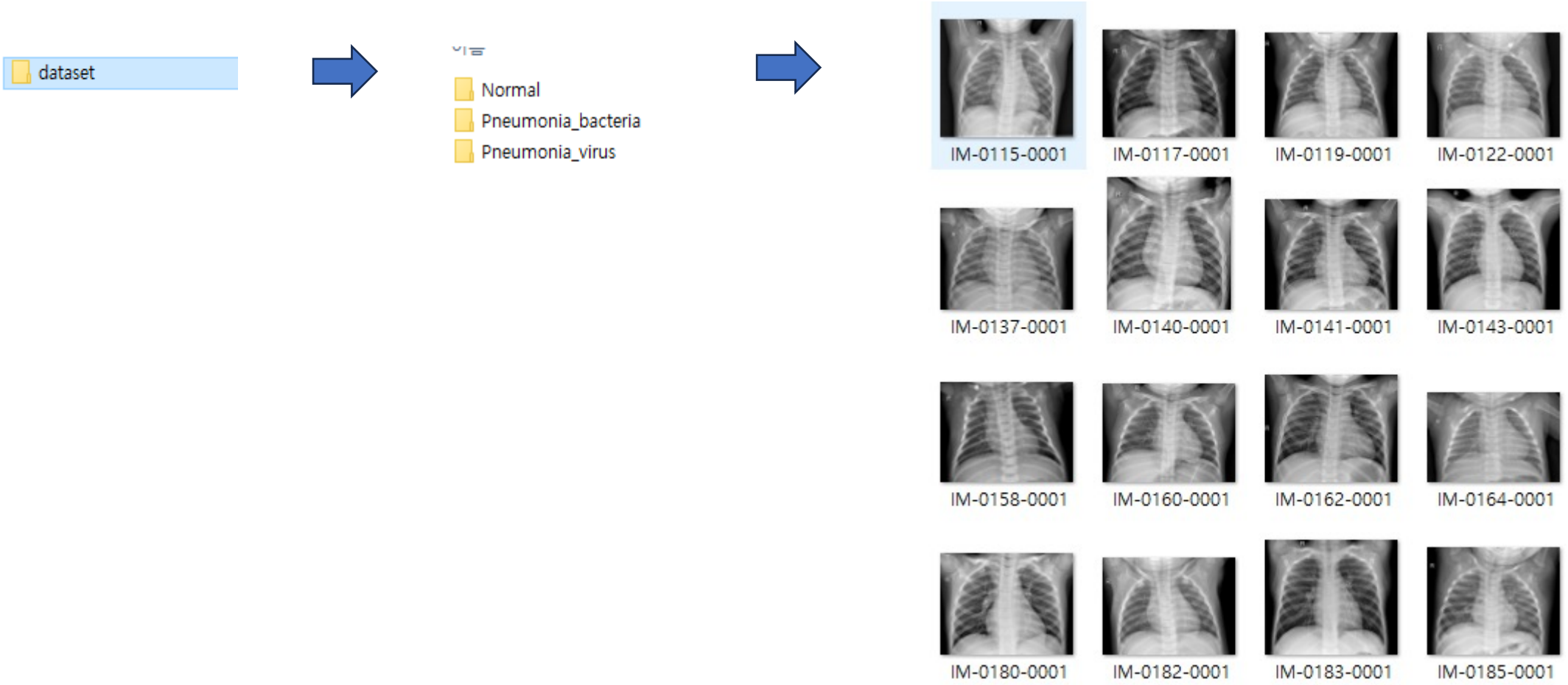
- 1341 레코드 - 정상
- 2530 기록 폐렴 박테리아
- 1345 기록 폐렴 바이러스

- 흉부 X선 이미지(전후)는 광저우의 광저우 여성 및 아동 의료 센터에서 1~5세 소아 환자의 후향적 코호트에서 선택되었습니다. 모든 흉부 X선 영상은 환자의 일상적인 임상 치료의 일부로 수행되었습니다.

- 흉부 X-레이 이미지 분석을 위해 모든 흉부 방사선 사진은 초기에 품질이 낮거나 읽을 수 없는 스캔을 모두 제거하여 품질 관리를 위해 선별되었습니다. 그런 다음 이미지에 대한 진단은 AI 시스템 교육을 위해 승인되기 전에 두 명의 전문 의사가 등급을 매겼습니다. 채점 오류를 설명하기 위해 세 번째 전문가도 평가 세트를 확인했습니다.

이미지 분류 문제 해결 학습

- 데이터 소개 (폴더 구성) – dataset train / val 데이터를 나눌 필요있음




감사합니다.