**NIPA AI 문제해결 경진대회 코드 설명자료**

**- 과제명 : 타일 이미지 내 미세 Crack 검출 모델**

**팀명: 머닝러신머닝, 대회 참가자 (김경현 진현동 이재욱 신재현)**

**2020년 12월 16일**

1. **라이브러리 및 데이터 (Library & Data)**
2. DataLoader 관련: Pytorch의 Dataset의 Subclass를 이용하여 DataLoader 클래스 NewDataset 생성
3. Data Augmentation 관련: Pytorch의 Dataset 클래스의 모듈인 transform으로 동적으로 Augmentation을 수행하였다.
4. train-test Data Split 관련: scikit-learn의 model\_selection 모듈을 이용하여 동적으로 나누어 사용하였다.
5. Model 관련: ImageNet으로 Pretrained된 EfficientNet과 Xception을 사용.

|  |  |
| --- | --- |
| |  | | --- | | **import** pretrainedmodels  **class** NewDataset(Dataset):  **def** \_\_init\_\_(self, csv, root='./data', phase='train', transform=**None**):  self.csv = csv  self.root = f'{root}/{phase}'  self.transform = transform    **def** \_\_len\_\_(self):  **return** len(self.csv)    **def** \_\_getitem\_\_(self, idx):  **if** torch.is\_tensor(idx):  idx = idx.tolist()    file\_name = self.csv.iloc[idx][0]  title\_name = self.csv.iloc[idx][1]  label = self.csv.iloc[idx][2]    image = Image.open(f'{self.root}/{file\_name}') | |

1. **데이터 전처리 (Data Cleansing & Pre-Processing)**
2. train data에 Data augmentation을 적용하였다.

균열의 크기를 더 크게 만들어 정확도를 높여보려고 image resize를 수행했으나 큰 효과가 없었고 오히려 성능이 하락하였다.

train data에서 결함이 있다고 판정된 것이 11%에 불과하기에, 결함이 있는 데이터에 가중치를 부여하는 Sampling 기법을 이용하였지만 큰 효과는 없었고, cutmix, focal loss 등의 방법을 적용했을 때도 모델의 정확도에 큰 효과가 없었다.

1. 비교적 간단한 Augmenation 기법만 사용하였다.

- HorizontalFlip (Vertical flip의 경우 효과가 없었음)

- Normalize: ToTensor와 함께 활용하여 pixel의 RGB-3 value를 표준편차와 평균을 0.5로 하여 -1~1 구간으로 Scaling

|  |  |
| --- | --- |
| |  | | --- | | **from** torchvision **import** transforms  # Data Augmentation  transform\_train = transforms.Compose([  transforms.RandomHorizontalFlip(),  transforms.ToTensor(), //ToTensor로  transforms.Normalize(mean=0.5, std=0.5),  ]) | |

1. **탐색적 자료 분석 (Exploratory Data Analysis)**
2. 이미지를 PIL library를 활용하여 출력해봄으로써 EDA를 진행하였다.
3. 결함이 있다고 labeling된 데이터와 결함 없음(정상)으로 labeling된 데이터의 비균일한 분포를 조정하기 위해 Sampling 기법을 활용해보았으나 Sampling을 사용하지 않은 모델이 더 좋은 결과를 보였다.
4. **변수 선택 및 모델 구축 (Feature Engineering & Initial Modeling)**
5. Training 과정에서 validation set은 미리 나누어 놓지 않고, scikit-learn의 train\_test\_split을 이용하여 전체 train set에서 train data와 validation data를 트레이닝 시작 시 나누어 사용했다.
6. abalation study 진행 결과, EfficientNet-B5와 Xception을 앙상블한 모델이 가장 적합한 모델로 선정됐다.

|  |  |
| --- | --- |
| find\_model 메서드, **class** EffNet(nn.Module): ,**class** Xception(nn.Module):   |  | | --- | | **from** efficientnet\_pytorch **import** EfficientNet  **import** pretrainedmodels  # 로드할 model 찾는 함수  **def** find\_model(model\_type):  **if** model\_type **in** ['b0', 'b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7']:  model = EffNet(type=model\_type, pretrained=**True**, num\_classes=2)  **elif** model\_type == 'xception':  model = Xception(num\_classes=num\_classes)  **return** model  **from** sklearn.metrics **import** f1\_score, confusion\_matrix  **for** idx **in** range(len(models[0])):  print(f"[Ensemble model {idx+1} load]")  model\_type = models[0][idx]  model\_name = models[1][idx]  **class** EffNet(nn.Module):  **def** \_\_init\_\_(self, type, pretrained=**True**, num\_classes=2):  super(EffNet, self).\_\_init\_\_()  **assert** type **in** ['b0', 'b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7'], 'efficientNet은 [b0, b1, b2, b3, b4, b5, b6, b7] 중에 하나이어야 합니다.'  **if** pretrained:  self.effnet = EfficientNet.from\_pretrained(f'efficientnet-{type}')  **else**:  self.effnet = EfficientNet.from\_name(f'efficientnet-{type}')  self.fc1 = nn.Linear(in\_features=1000,out\_features=num\_classes)  # self.fc2 = nn.Linear(in\_features=500out\_features=num\_classes)  self.relu = nn.ReLU()  **def** forward(self, x):  x = self.effnet(x)  x = self.relu(x)  x = self.fc1(x)  # x = self.relu(x)  # x = self.fc2(x)  **return** x  **class** Xception(nn.Module):  **def** \_\_init\_\_(self, num\_classes=2):  super(Xception, self).\_\_init\_\_()  self.net=pretrainedmodels.\_\_dict\_\_['xception'](num\_classes=1000 , pretrained='imagenet')  self.fc1 = nn.Linear(in\_features=1000,out\_features=num\_classes)  self.relu = nn.ReLU()  **def** forward(self, x):  x = self.net(x)  x = self.relu(x)  x = self.fc1(x)  **return** x | |

1. **모델 학습 및 검증 (Model Tuning & Evaluation)**

**모델 학습**

1) ImageNet에서 pretrain된 EficientNetB5 모델과 Xception 모델을 활용하여 전이학습(Transfer-Learning)을 수행하였다.

2) 앙상블 활용 : Xception과 EfficientNetB5 모델을 각각 3개씩 multi-model ensemble(앙상블)을 활용하였다. ResNet 등 다른 Pretrained 모델과 직접 모델을 구축하여 사용해보았지만 성능면에서 큰 이점이 없었다.

\_infer\_ensemble메서드와 feed\_infer메서드를 활용하였다.

**모델 검증**

베이스라인 코드로 주어진 evaluation.py를 통해 예측값의 f1\_score를 검증에 활용하였다.

|  |  |
| --- | --- |
| Validate 메서드, \_infer\_ensemble메서드, feed\_infer   |  | | --- | | model\_types = ['xception', 'xception', 'xception', 'b5', 'b5', 'b5']  saved\_model = ['./model/xception\_0/best.pth', './model/xception\_1/best.pth', './model/xception\_2/best.pth', './model/15\_b5/best.pth', './model/b5\_0/best.pth', './model/b5\_1/best.pth']  models = [model\_types, saved\_model]  ensemble(prediction\_file, models, test\_dataloader, cuda)  **from** sklearn.metrics **import** f1\_score, confusion\_matrix  **def** validate(prediction\_file, model, validate\_dataloader, validate\_label\_file, cuda):  feed\_infer(prediction\_file, **lambda** : \_infer(model, cuda, data\_loader=validate\_dataloader))    pred = pd.read\_csv(prediction\_file, header=**None**)  pred = pred[0].to\_list()    metric\_result = f1\_score(pred, validate\_label\_file)  print('Eval result: {:.4f}'.format(metric\_result))  **return** metric\_result  **def** evaluate(prediction\_labels, gt\_labels):  **if** len(prediction\_labels) != len(gt\_labels):  **return** 0  tp = 0  tn = 0  fp = 0  fn = 0  precision = 0  recall = 0  **for** i **in** range(len(prediction\_labels)):  **if** prediction\_labels[i] == gt\_labels[i] **and** prediction\_labels[i] == '1':  tp += 1  **elif** prediction\_labels[i] == gt\_labels[i] **and** prediction\_labels[i] == '0':  tn += 1  **elif** prediction\_labels[i] == '0' **and** gt\_labels[i] == '1':  fn += 1  **else**:  fp += 1  **if** tp+fp>0 :  precision = tp/(tp+fp)  **if** tp+fn >0:  recall = tp / (tp+fn)  temp = f1\_score(gt\_labels, prediction\_labels, pos\_label='1')  score = 0  **if** precision+recall >0 :  score = 2\*precision\*recall/(precision + recall)    **return** score  **def** \_infer\_ensemble(models, cuda, data\_loader):  res\_fc = **None**  total\_res\_fc = **None**  **for** idx **in** range(len(models[0])):  print(f"[Ensemble model {idx+1} load]")  model\_type = models[0][idx]  model\_name = models[1][idx]    model = find\_model(model\_type)  load\_model(model\_name, model)  **def** feed\_infer(output\_file, infer\_func):  prediction\_class = infer\_func()  print('write output')  predictions\_str = [] | |

1. **결과 모델 활용 방안**
2. 균열을 검출하기 위하여필터링을 통한 사전처리, 패턴인식, 이미지 중첩 인식 및 형상 검지, 조합 등과 같은 다수의 효과적인 이미지 프로세싱 기술 및 패턴인식 기법들이 고려되었다.
3. 저희 팀이 고안해낸 모델 또한 수집된 이미지를 활용하여 손상여부를 자동으로 검출해내는 딥러닝 기반의 AI 이미지 분석 기법으로 활용되어, 기존 고전적 방법이 지니는 인력적, 접근적 한계를 극복하여 막대한 인력과 시간, 비용을 절감하여 효과적인 안전점검에 기여할 수 있으면 좋겠습니다.
4. **결과 및 결언 (Conclusion & Discussion)**
5. 모델 학습하는 과정에서 커맨드 명령을 자동화하기 위하여 load\_model 메서드로 학습한 모델을 불러올 때 --mode 옵션을 ensemble로 주었을 때, multi model을 Soft-voting으로 앙상블한 모델결과를 불러올 수 있도록 하였다.
6. Abalation study를 통해 XCeption 3개, EfficientNet-B5 3개 모델을 Soft voting 으로 Ensemble 했을 때 최고 성능인 f1-score : 0.9531428571(public 기준)를 기록하였다.
7. Abalation study 과정에서 HPO(hyper-parameter optimization)을 통해
   * 10 epoch (Transfer-learning)
   * batch-size : 64
   * learning\_rate: 1e-3
   * StepLearningRateDecay (매 3epoch 마다 learning\_rate를 1/10로 줄였음)

를 적용하였다.

|  |  |
| --- | --- |
| 커맨드 명령어   |  | | --- | | !python main.py --mode ensemble --prediction\_file./prediction/153\_ensemble\_final.tsv  //기능: 모델 기반으로 predictionfile생성 | |