지능화 캡스톤 프로젝트

프로젝트 #1 결과 발표

2024. 04. 29

충북대학교 산업인공지능학과

[7조] 김봉균, 김혜영, 이준혁

프로젝트 수행체계

수행방법

-7조는 3인으로 구성
-자료조사/학습/발표/증량 등으로 업무를 분할하여 수행함
-개개인의 지역이 멀어 현재 줌 회의 또는 1주일 2회씩 비대면 회의 등으로 수행
-대면 회의 수행
-CNN-WDI, VGG-16, MobileNetV2 모델 등 구현하여 비교
-Pycharm, Jupiter notebook, Colab 등 사용

업무분장

이름	수행내용	비고
김봉균	• 자료조사 • 비교모델 학습 (VGG-16) • 모델 정의	
김혜영	• 중간발표 • 데이터 증량 • CNN모델 학습 (WDI)	
이준혁	• 비교모델 학습 (MobileNetV2) • 모델 예측 • 발표자료 정리	

데이터셋

데이터 확인

- 불량 갯수 계산
- 불량패턴 데이터 및 NONE 라벨 데이터 추출
- 전체 웨이퍼 개수 확인

```
[ ]  df['failureNum']=df.failureType      # failureType을 failureNum 변수에 입력
     df['trainTestNum']=df.trianTestLabel # trianTestLabel을 trainTestNum 변수에 입력 (trainTestNum 데이터 자체 오타)

     mapping_type={
         'Center':0,
         'Donut':1,
         'Edge-Loc':2,
         'Edge-Ring':3,
         'Loc':4,
         'Random':5,
         'Scratch':6,
         'Near-full':7,
         'none':8}

     mapping_traintest={'Training':0,'Test':1}

     df=df.replace({'failureNum':mapping_type, 'trainTestNum':mapping_traintest}) #문자를 숫자로 치환

     df_label = df[(df['failureNum']>=0) & (df['failureNum']<=8)]
     df_label = df_label.reset_index()
     df_pattern = df[(df['failureNum']>=0) & (df['failureNum']<=7)]
     df_pattern = df_pattern.reset_index()
     df_none = df[(df['failureNum']==8)]

     # 1) 불량 총 갯수, 2) 불량 패턴, 3) none
     df_label.shape[0], df_pattern.shape[0], df_none.shape[0]

     (172950, 25519, 147431)
```

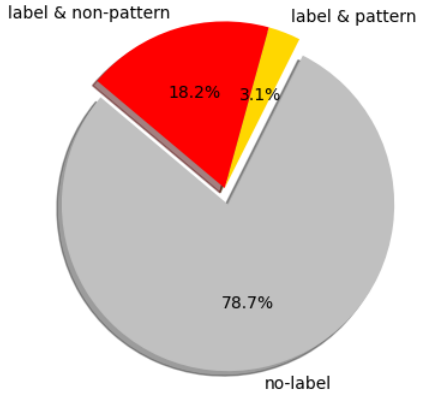
불량 총 갯수: 172,950
불량 패턴에 해당하는 데이터 수: 25,519
'none'으로 라벨링된 데이터 수: 147,431
전체 웨이퍼 개수: 811457

데이터셋★

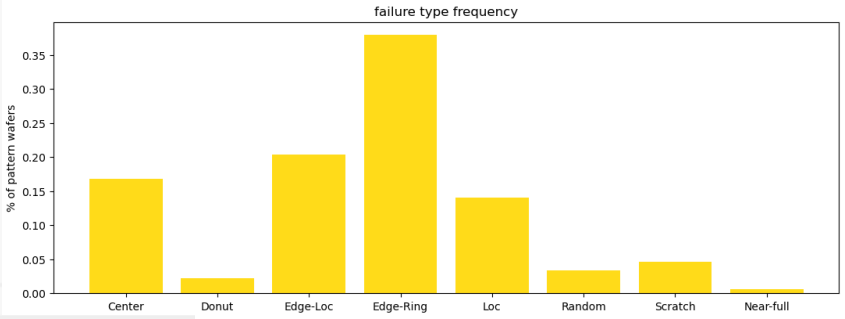
데이터 시각화

- 데이터 시각화 그리드
- 데이터 유형에 따른 차트
- 불량패턴 별 막대차트

```python
from PIL import Image

def save_image(image_data, save_dir, image_name):
    # 객체 생성
    width, height = len(image_data[0]), len(image_data)
    img = Image.new('RGB', (width, height))

    # 픽셀 색상 지정
    for y in range(height):
        for x in range(width):
            if image_data[y][x] == 0:
                img.putpixel((x,y), (255,255,255)) # 흰색
            elif image_data[y][x] == 1:
                img.putpixel((x,y), (255,200,0)) # 노랑
            elif image_data[y][x] == 2:
                img.putpixel((x,y), (0,50,200)) # 파랑

    # 저장 디렉토리
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    # 이미지 저장
    save_path = os.path.join(save_dir, image_name)
    img.save(save_path)
```



데이터 유형 차트



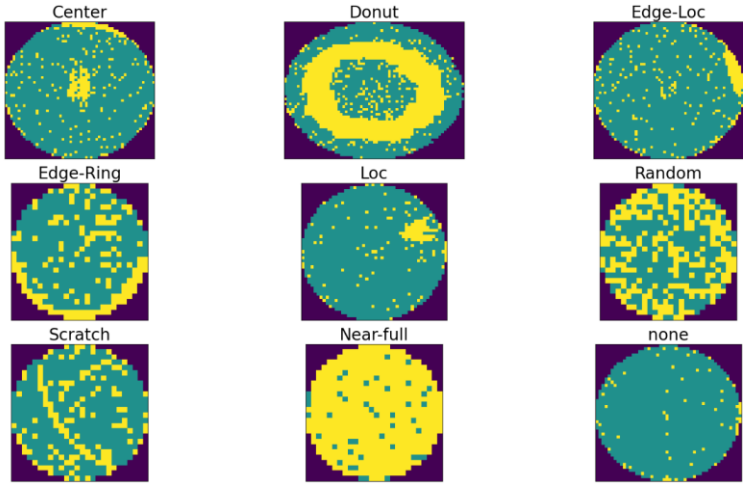
불량패턴 차트

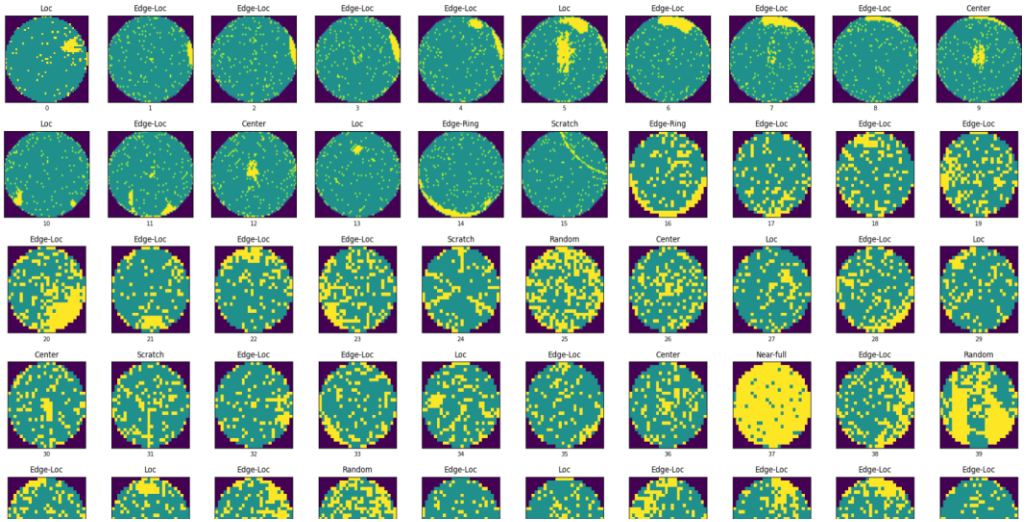
데이터셋★

데이터 시각화

- 불량 패턴 및 미검출 데이터에 대한 이미지 시각화

```
# 불량 패턴 8가지 원본 이미지 추출
for j in x:
    img = df_pattern.waferMap[df_pattern.failureType==labels[j]]
    for i in range(img.shape[0] - 1):
        image_data = img[img.index[i]]
        save_dir = './wm_images/' + str(df_pattern.failureType[img.index[i]][0][0])
        save_image(image_data, save_dir, 'wm_' + str(i) + '.bmp')

# none 추출 (1만개만 추출하여 그대로 사용)
img = df_none.waferMap[df_none.failureType=='none']
for i in range(10000):
    image_data = img[img.index[i]]
    save_dir = './wm_images/none'
    save_image(image_data, save_dir, 'wm_' + str(i) + '.bmp')
```





데이터셋

데이터 구성

- 데이터 증량 - 불량패턴별 10,000건 생성 **총 90,000** 건 생성
- 데이터 분할 - train : val : test = **60 : 25 : 15**

데이터 구분		증량 전	증량 후	비고
전체 811,457	With Lable 172,950 21.3%	None 147,431 (18.17%)	None 10,000	- 이미지 회전 30° - 이미지 크롭 - 이미지 수평 flip 50% - 수평 · 수직 이동 10% - 변환 시 공간 채움 흰색
		Center 4,294 (0.53%)	Center 10,000	
		Donut 555 (0.07%)	Donut 10,000	
		Edge-Loc 5189 (0.64%)	Edge-Loc 10,000	
		Edge-Ring 9680 (1.19%)	Edge-Ring 10,000	
		Local 3593 (0.44%)	Local 10,000	
		Random 866 (0.11%)	Random 10,000	
		Scratch 1193 (0.15%)	Scratch 10,000	
		Near-full 149 (0.02%)	Near-full 10,000	

데이터셋

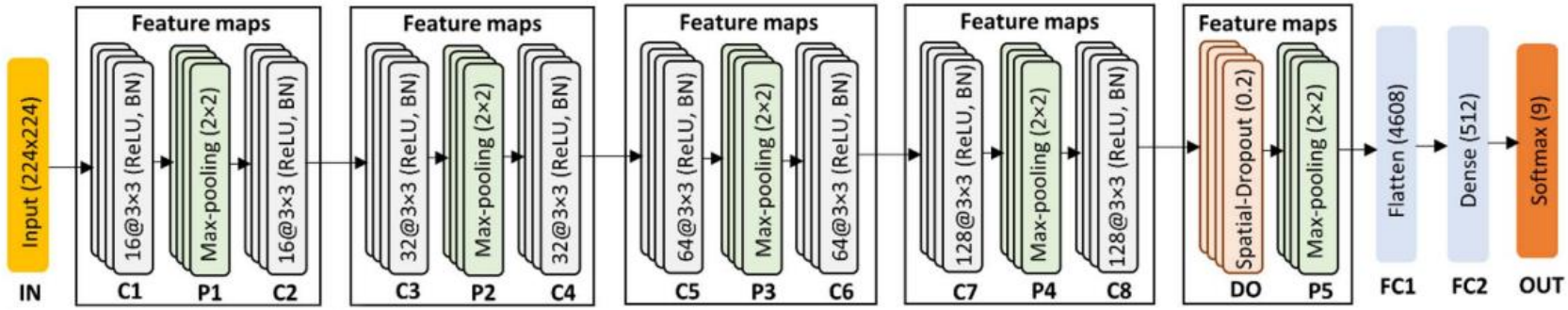
데이터 증량 코드

```python
import random
import torch
from torchvision import transforms
from torchvision.io import read_image
from torchvision.utils import save_image
import torchvision.transforms.functional as TF

def augment_images(save_dir, max_num_images=10000, seed=42):
    # 데이터 증강 파라미터 설정 파이프라인 정의
    transform = transforms.Compose([
        transforms.RandomRotation(40, fill=1), # 40도 회전
        #transforms.RandomResizedCrop((224, 224), scale=(0.8, 1.0)),
        transforms.RandomResizedCrop(224, scale=(0.9, 1.0), ratio=(0.8, 1.25)),  # 크기 조정 및 크롭
        transforms.RandomHorizontalFlip(), # 좌우 반전
        transforms.RandomAffine(degrees=0, translate=(0.2, 0.15), shear=0.1, scale=(0.9, 1.1), fill=1), # 이동, 전단, 스케일 변환
        transforms.RandomApply([transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1)], p=0.5), # 색상 변환
    ])

    random.seed(seed)
    image_file_list = os.listdir(save_dir)
    random.shuffle(image_file_list)
    max_num_augmented_images = max_num_images - len(image_file_list)
    num_augmentations = max_num_augmented_images // len(image_file_list) + 1

    i = 0
    for file_name in image_file_list:
        image_path = os.path.join(save_dir, file_name)
        image = Image.open(image_path).convert("RGB") # BMP 파일을 열고 RGB로 변환
        image = TF.to_tensor(image) # PIL 이미지를 PyTorch 텐서로 변환
        image = transforms.Resize((224, 224))(image) # 크기 조정

        for j in range(num_augmentations):
            augmented_image = transform(image)
            save_path = os.path.join(save_dir, f'{file_name}d_{j}.bmp')
            save_image(augmented_image, save_path)
            i += 1
            if i >= max_num_images:
```

모델 학습하기

모델 정의 (논문과 동일)

- CNN구조
  8개의 Conv층, 5개의 max-pooling층, 2개의 fully층, 1개의 출력층
- 파이토치 이용
- 20 epoch, 32 batch, Adam Optimizer 0.001 학습률 적용



*Note: IN denotes input layer; C convolutional layer; P pooling layer; DO dropout layer; FC fully connected layer; OUT output layer; and BN batch normalization

Fig. 3. Architecture of proposed deep CNN model for wafer defect identification.

모델 학습하기

모델 정의 코드

```
# 모델 정의

from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F

# 데이터 전처리
data_transforms = {
    'train' : transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val' : transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test' : transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

# 데이터셋 로드
data_dir = './wm_images_augmented'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir,x),data_transforms[x])
                  for x in ['train', 'val', 'test']}

# 데이터 로더 생성
dataloaders = {x: DataLoader(image_datasets[x], batch_size=32,
                             shuffle=True, num_workers=4)
               for x in ['train', 'val', 'test']}
```

```
# 모델 정의 (CNN)
class createCNN(nn.Module):
    # 신경망 레이어 초기화 함수
    def __init__(self):
        super(createCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=0)  # 변화: 필터 사이즈, 패딩
        self.conv2 = nn.Conv2d(16, 16, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
        self.conv5 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv6 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.conv7 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv8 = nn.Conv2d(128, 128, kernel_size=3, padding=1)

        self.fc1 = nn.Linear(128 * 6 * 6, 4608)
        self.dropout1 = nn.Dropout(0.5)  # Dropout 추가, 50% 확률로 뉴런 비활성화
        self.fc2 = nn.Linear(4608, 512)
        self.dropout2 = nn.Dropout(0.5)  # 또 다른 Dropout 추가
        self.fc3 = nn.Linear(512, 9)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv6(x))
        x = F.relu(self.conv7(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv8(x))
        x = F.max_pool2d(x, 2)
        x = x.view(-1, 128 * 6 * 6)  # Flatten
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


model = createCNN()
```

모델 학습하기

딥러닝 학습 코드

- 옵티마이저, 손실함수 설정

```
# 모델 학습

import torch.optim as optim
from torch.optim.lr_scheduler import StepLR

# 손실 함수 및 옵티마이저 설정
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = StepLR(optimizer, step_size=5, gamma=0.1)  # 5 에포크마다 학습률을 10%로 감소

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

```
createCNN(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv8): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=4608, out_features=4608, bias=True)
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=4608, out_features=512, bias=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc3): Linear(in_features=512, out_features=9, bias=True)
)
```

모델 학습하기

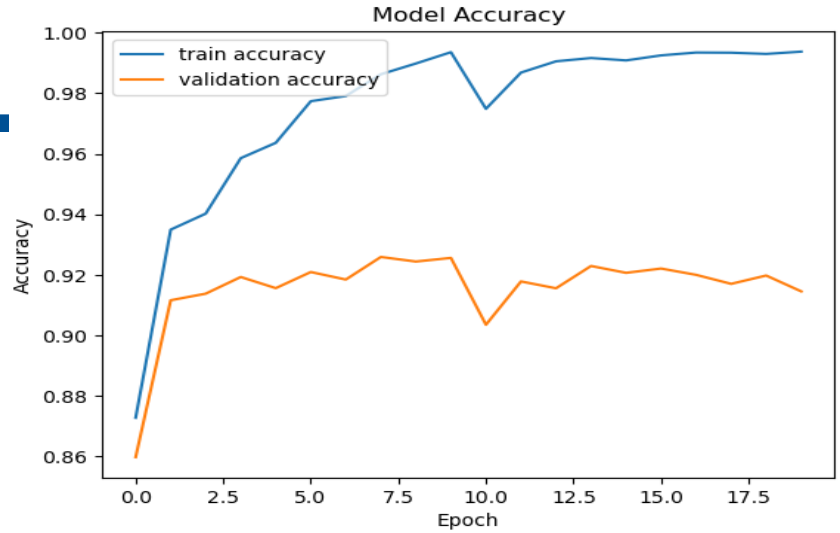
딥러닝 학습 및 결과

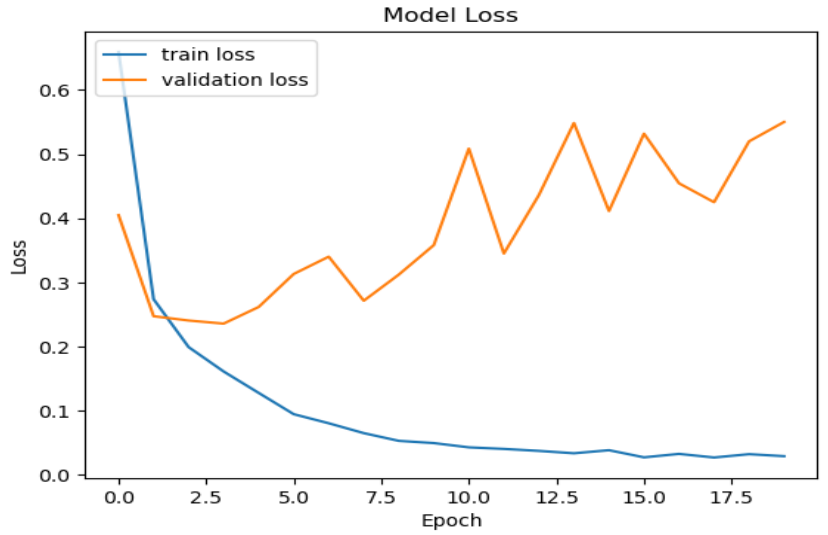
```
Epoch 1, Train Loss: 0.6594049721691344, Val Loss: 0.4048885262489319, Train Acc: 0.8727592592592592, Val Acc: 0.8597333333333333
Epoch 2, Train Loss: 0.2737144338885943, Val Loss: 0.24716902501359583, Train Acc: 0.9349444444444445, Val Acc: 0.9116000000000001
Epoch 3, Train Loss: 0.1989815527206218, Val Loss: 0.24043595568471485, Train Acc: 0.9402407407407407, Val Acc: 0.9137777777777778
Epoch 4, Train Loss: 0.160935813187725, Val Loss: 0.23563305019827352, Train Acc: 0.958574074074074, Val Acc: 0.9192888888888889
Epoch 5, Train Loss: 0.12751402568292838, Val Loss: 0.26158025779922806, Train Acc: 0.9636481481481481, Val Acc: 0.9156000000000001
Epoch 6, Train Loss: 0.09433246476761997, Val Loss: 0.31306321712654706, Train Acc: 0.9774074074074074, Val Acc: 0.9209333333333334
Epoch 7, Train Loss: 0.0801632749607045, Val Loss: 0.34006329802977076, Train Acc: 0.9791296296296296, Val Acc: 0.9184444444444445
Epoch 8, Train Loss: 0.06476278730017064, Val Loss: 0.27132042678859497, Train Acc: 0.9863148148148148, Val Acc: 0.9259111111111111
Epoch 9, Train Loss: 0.052724574359644776, Val Loss: 0.31205671766565907, Train Acc: 0.9898703703703704, Val Acc: 0.9244
Epoch 10, Train Loss: 0.04924348046605775, Val Loss: 0.3581447919471462, Train Acc: 0.9936111111111111, Val Acc: 0.9256000000000001
Epoch 11, Train Loss: 0.04258626778200999, Val Loss: 0.5086433635751856, Train Acc: 0.9748888888888889, Val Acc: 0.9035111111111112
Epoch 12, Train Loss: 0.0401481587919827, Val Loss: 0.3449835972543806, Train Acc: 0.9868703703703704, Val Acc: 0.9178222222222223
Epoch 13, Train Loss: 0.037055867442178876, Val Loss: 0.4362408522541531, Train Acc: 0.9905925925925926, Val Acc: 0.9155555555555556
Epoch 14, Train Loss: 0.03333595904932251, Val Loss: 0.5486150448103707, Train Acc: 0.9916851851851852, Val Acc: 0.9229333333333334
Epoch 15, Train Loss: 0.03812388848532525, Val Loss: 0.41129895757395335, Train Acc: 0.9908888888888889, Val Acc: 0.9206666666666667
Epoch 16, Train Loss: 0.02701124951504392, Val Loss: 0.5319765442681679, Train Acc: 0.9925555555555555, Val Acc: 0.922088888888889
Epoch 17, Train Loss: 0.032370175936621476, Val Loss: 0.45449883134787283, Train Acc: 0.9934999999999999, Val Acc: 0.92
Epoch 18, Train Loss: 0.02676630144986783, Val Loss: 0.42502809884875187, Train Acc: 0.9934444444444445, Val Acc: 0.9170222222222223
Epoch 19, Train Loss: 0.03192728947915844, Val Loss: 0.5199580303349222, Train Acc: 0.993037037037037, Val Acc: 0.9197777777777778
Epoch 20, Train Loss: 0.02881647721731, Val Loss: 0.5502585270502501, Train Acc: 0.9937962962962963, Val Acc: 0.9145333333333334
```

모델 학습하기

학습결과 시각화

```python
# 학습 이력 로드
with open('train_history.pkl', 'rb') as f:
    history = pickle.load(f)

# 정확도 및 손실 그래프
plt.plot(history['train_accuracy'], label='train accuracy')
plt.plot(history['val_accuracy'], label='validation accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

plt.plot(history['train_loss'], label='train loss')
plt.plot(history['val_loss'], label='validation loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

# 테스트 데이터셋에 대한 평가
test_accuracy = calculate_accuracy(model, dataloaders['test'], device)
print(f"Test Accuracy: {test_accuracy.item() * 100:.2f}%")

# 예측 및 평가 지표
predictions, true_labels = predict(model, dataloaders['test'])
print('Predictions:', predictions)
```






```
Test Accuracy: 91.88%
```

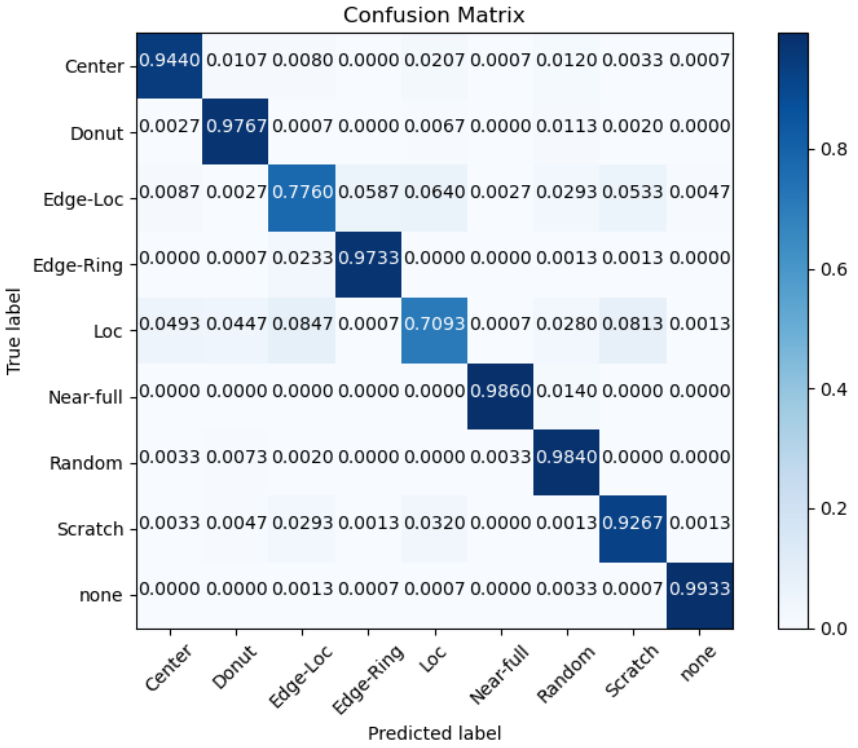
- **정확도 그래프**: 학습 과정에서 각 에폭마다 학습 정확도와 검증 정확도를 시각화
- **손실 그래프**: 학습 과정에서 각 에폭마다 학습 손실과 검증 손실을 시각화
- **테스트 정확도 출력**: **91.88%**

비교, 성능평가

모델 평가 및 혼동행렬

```python
# 예측값 도출 함수
def predict(model, dataloader):
    model.eval()  # 모델을 평가 모드로 설정

    predictions = []
    true_labels = []

    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            predictions.extend(predicted.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    return predictions, true_labels

# Confusion Matrix 계산

# scikit-learn 활용하여 평가 메트릭 계산
print('Accuracy = {:.2f}'.format(accuracy_score(true_labels, predictions)))
print('Precision = {:.2f}'.format(precision_score(true_labels, predictions, average=None)[0]))
print('Recall = {:.2f}'.format(recall_score(true_labels, predictions, average=None)[0]))
print('F1-score = {:.2f}'.format(f1_score(true_labels, predictions, average=None)[0]))

# 혼동 행렬
def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=plt.cm.Blues):

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.4f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

```
Accuracy = 0.92
Precision = 0.93
Recall = 0.94
F1-score = 0.94
```



다양한 딥러닝 아키텍처와의 비교, 성능평가

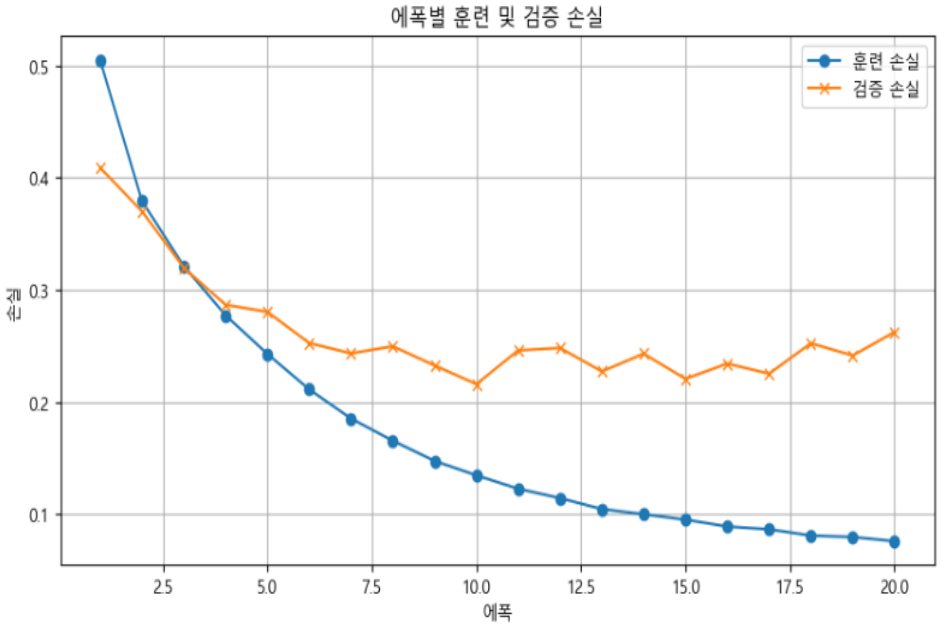
VGG-16 모델 정의 코드 및 결과

```python
# VGG-16 모델
from torchvision.models import vgg16, VGG16_Weights
vgg16 = vgg16(weights=VGG16_Weights.IMAGENET1K_V1)

num_features = vgg16.classifier[6].in_features
vgg16.classifier[6] = nn.Linear(num_features, 9)

for param in vgg16.features.parameters():
    param.requires_grad = False
# 손실 함수와 옵티마이저를 설정
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(vgg16.classifier.parameters(), lr=0.0001)  # 학습률을 더 세밀하게 조정

# 모델 학습 함수 정의
def train_model(model, dataloaders, criterion, optimizer, num_epochs=20):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for inputs, labels in dataloaders['train']:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)

        epoch_loss = running_loss / len(dataloaders['train'].dataset)
        print(f'에폭 {epoch+1}, 훈련 손실: {epoch_loss:.4f}')

        # 검증 단계
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for inputs, labels in dataloaders['val']:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item() * inputs.size(0)

        val_loss = val_loss / len(dataloaders['val'].dataset)
        print(f'검증 손실: {val_loss:.4f}')

# 모델 학습 시작
train_model(vgg16, dataloaders, criterion, optimizer, num_epochs=20)
```

```
에폭 1, 훈련 손실: 0.5048
검증 손실: 0.4091
에폭 2, 훈련 손실: 0.3794
검증 손실: 0.3702
에폭 3, 훈련 손실: 0.3214
검증 손실: 0.3199
에폭 4, 훈련 손실: 0.2775
검증 손실: 0.2869
에폭 5, 훈련 손실: 0.2431
검증 손실: 0.2806
에폭 6, 훈련 손실: 0.2115
검증 손실: 0.2528
에폭 7, 훈련 손실: 0.1855
검증 손실: 0.2436
에폭 8, 훈련 손실: 0.1657
검증 손실: 0.2499
에폭 9, 훈련 손실: 0.1477
검증 손실: 0.2328
에폭 10, 훈련 손실: 0.1349
검증 손실: 0.2162
에폭 11, 훈련 손실: 0.1229
검증 손실: 0.2463
에폭 12, 훈련 손실: 0.1146
검증 손실: 0.2484
에폭 13, 훈련 손실: 0.1048
검증 손실: 0.2275
에폭 14, 훈련 손실: 0.1002
검증 손실: 0.2433
에폭 15, 훈련 손실: 0.0954
검증 손실: 0.2209
에폭 16, 훈련 손실: 0.0893
검증 손실: 0.2345
에폭 17, 훈련 손실: 0.0868
검증 손실: 0.2255
에폭 18, 훈련 손실: 0.0812
검증 손실: 0.2526
에폭 19, 훈련 손실: 0.0799
검증 손실: 0.2417
에폭 20, 훈련 손실: 0.0762
검증 손실: 0.2626
테스트 정확도: 95.28%
```

다양한 딥러닝 아키텍처와의 비교, 성능평가

VGG-16 모델 성능 평가 및 혼동행렬



웨이퍼 탐지 모델 테스트 정확도: 95.28%

Accuracy: 0.95
Precision: 0.95
Recall: 0.95
F1-Score: 0.95

다양한 딥러닝 아키텍처와의 비교, 성능평가

MobileNetV2

- 모바일 및 임베디드 장치에서 경량화된 딥 러닝 모델을 구축용
- 이미지 분류, 객체 감지 및 전이 학습과 같은 다양한 컴퓨터 비전 작업 수행
- 계산 및 메모리 요구 사항을 줄이기 위해 설계됨
- 선형 활성화 함수 대신 선형 활성화 함수가 사용

```python
#MobileNetV2로 진행
import torchvision.models as models
import torch
import torch.nn as nn
import torch.optim as optim


# 모델 정의 (MobileNetV2)
class MobileNetV2(nn.Module):
    def __init__(self, num_classes=9):
        super(MobileNetV2, self).__init__()
        self.mobilenetv2 = models.mobilenet_v2(pretrained=True)
        self.mobilenetv2.classifier[1] = nn.Linear(self.mobilenetv2.classifier[1].in_features, num_classes)

    def forward(self, x):
        return self.mobilenetv2(x)

# 모델 생성
model = MobileNetV2()


# 손실 함수 및 옵티마이저 설정
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

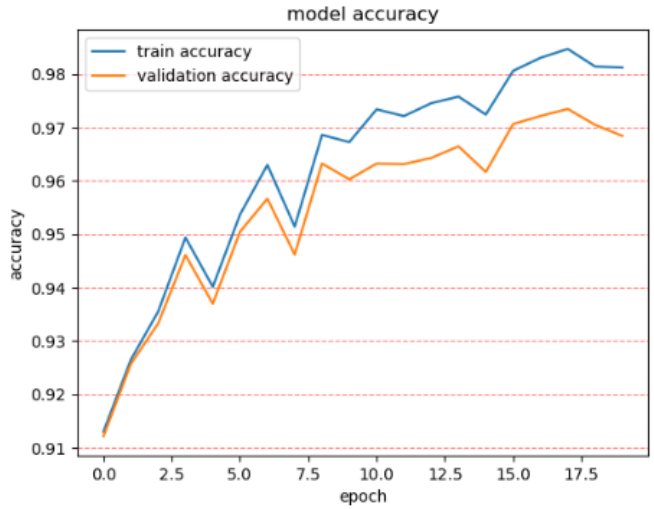
```
Epoch 1, Train Loss: 0.3837687751193353, Val Loss: 0.24369547456808133, Train Acc: 0.9130260986177017, Val Acc: 0.9121657118555024
Epoch 2, Train Loss: 0.2579365422000387, Val Loss: 0.20891162497356036, Train Acc: 0.9264622446874355, Val Acc: 0.9256748305105946
Epoch 3, Train Loss: 0.22048335327041108, Val Loss: 0.18639364918634746, Train Acc: 0.9354755518877657, Val Acc: 0.9332549470892852
Epoch 4, Train Loss: 0.19595783757830199, Val Loss: 0.16023628932377437, Train Acc: 0.9493501134722508, Val Acc: 0.9460635929252246
Epoch 5, Train Loss: 0.17609963209483767, Val Loss: 0.17845912555375607, Train Acc: 0.9401304930885084, Val Acc: 0.9369574462762364
Epoch 6, Train Loss: 0.15887461336491365, Val Loss: 0.13785968646361438, Train Acc: 0.9536697957499484, Val Acc: 0.9504415480449303
Epoch 7, Train Loss: 0.14667620593964384, Val Loss: 0.12110287044116205, Train Acc: 0.9629538890035073, Val Acc: 0.9566207189853151
Epoch 8, Train Loss: 0.13777731909497892, Val Loss: 0.1515273442181227, Train Acc: 0.9514003507324118, Val Acc: 0.9461636604708178
Epoch 9, Train Loss: 0.12771427111119132, Val Loss: 0.10326112905839709, Train Acc: 0.9685630286775325, Val Acc: 0.963200160108073
Epoch 10, Train Loss: 0.11977801022033548, Val Loss: 0.11514426822993791, Train Acc: 0.9672219929853517, Val Acc: 0.9602481675130713
Epoch 11, Train Loss: 0.11026716539826477, Val Loss: 0.10459391126902022, Train Acc: 0.9733598101918712, Val Acc: 0.963200160108073
Epoch 12, Train Loss: 0.10413184331300374, Val Loss: 0.10360637502359704, Train Acc: 0.9720961419434702, Val Acc: 0.9631000925624796
Epoch 13, Train Loss: 0.09574877827552983, Val Loss: 0.10345082682295251, Train Acc: 0.9745074272746028, Val Acc: 0.9642508693368024
Epoch 14, Train Loss: 0.09322187073140828, Val Loss: 0.0975307374154388, Train Acc: 0.9757453063750774, Val Acc: 0.9664273384534561
Epoch 15, Train Loss: 0.08751825558850575, Val Loss: 0.11321295010671782, Train Acc: 0.9723798225706622, Val Acc: 0.9616240962649789
Epoch 16, Train Loss: 0.08276854460843869, Val Loss: 0.08890123548552908, Train Acc: 0.9805678770373427, Val Acc: 0.9706051584819754
Epoch 17, Train Loss: 0.07869313093223088, Val Loss: 0.08372502162953695, Train Acc: 0.9830049515164019, Val Acc: 0.9721061716658744
Epoch 18, Train Loss: 0.07546758990197677, Val Loss: 0.08146293874636255, Train Acc: 0.9846683515576645, Val Acc: 0.9734320666449854
Epoch 19, Train Loss: 0.06947457675304443, Val Loss: 0.09429063283557897, Train Acc: 0.9813802351970291, Val Acc: 0.9704550571635854
Epoch 20, Train Loss: 0.06813407243655265, Val Loss: 0.09383416864000868, Train Acc: 0.98118681658758, Val Acc: 0.968378655592525
```

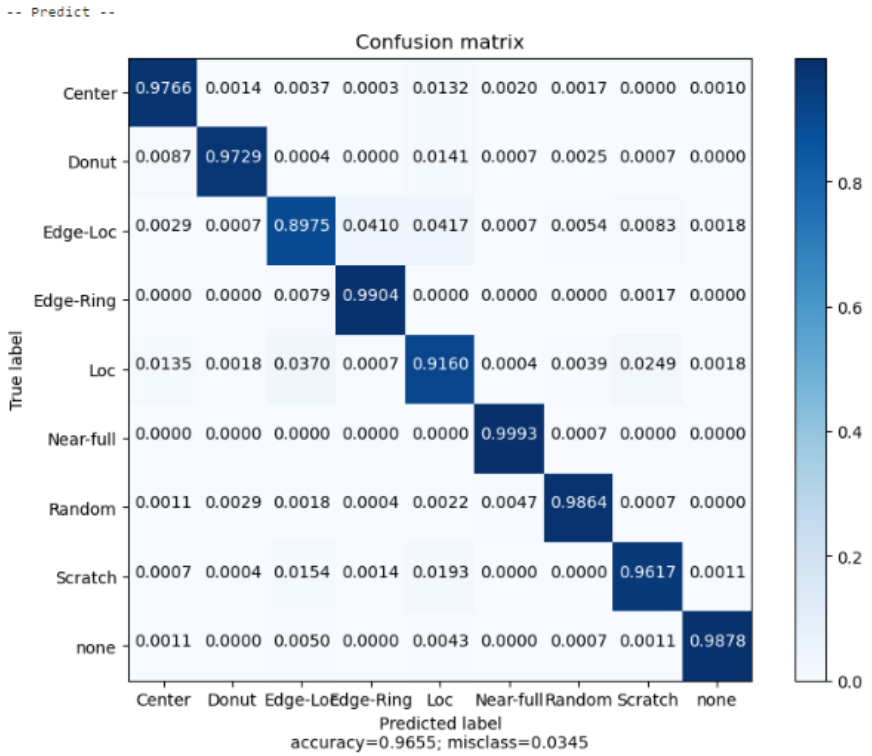
```
MobileNetV2(
  (mobilenetv2): MobileNetV2(
    (features): Sequential(
      (0): Conv2dNormActivation(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU6(inplace=True)
      )
      (1): InvertedResidual(
        (conv): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (2): InvertedResidual(
        (conv): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=96, bias=False)
            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
Test Accuracy: 0.9655199621271895
```

다양한 딥러닝 아키텍처와의 비교, 성능평가

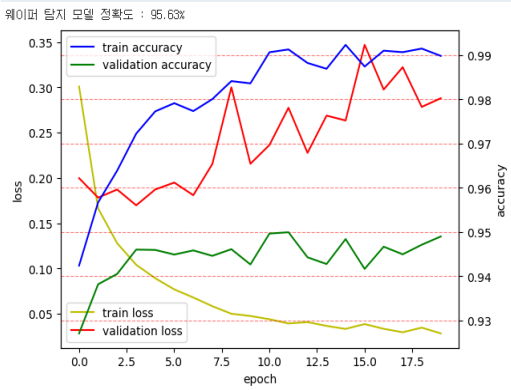
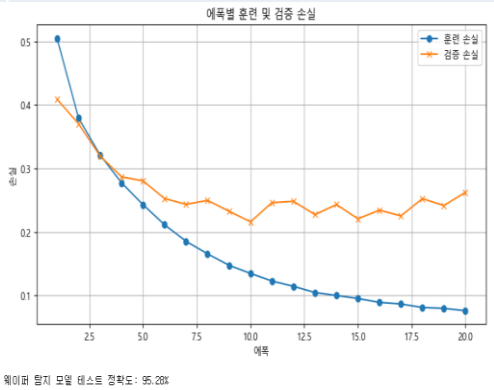
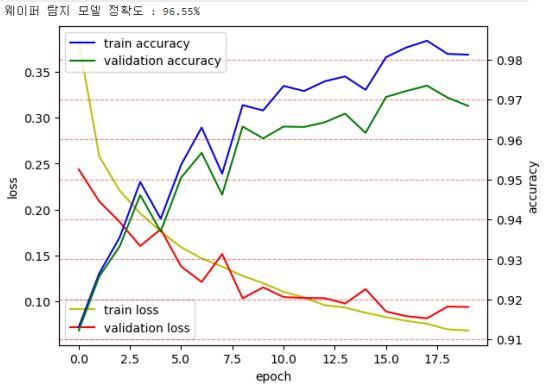
MobileNetV2 모델 성능 평가 및 혼동행렬






```
Accuracy = 0.97
Precision = 0.97
Recall = 0.98
f1-score = 0.98
```

다양한 딥러닝 아키텍처와의 비교, 성능평가

Classifier	CNN-WDI(기존)	VGG-16	CNN-MOBILEnetV2
Training Acc	**0.92**	**0.95**	**0.97**
Testing Acc	**0.92**	**0.95**	**0.96**
Precision	**0.93**	**0.95**	**0.97**
Recall	**0.94**	**0.95**	**0.98**
F1-Score	**0.94**	**0.95**	**0.98**
그래프 비교			

결론

결론

1. Training Accuracy와 Testing Accuracy 비교

CNN은 학습 및 테스트 데이터셋에서
비교 모델로 선택한 VGG-16, MobileNetV2 두 모델 모두
비교적 높은 정확도를 보였음.
이는 모델이 훈련 데이터에 잘 적합됨을 확인.

2. Precision, Recall, F1-Score 비교

두 모델 모두 CNN-WDI(기존)에 모델에 비해 더 높은 성능 평가를 보임
→ 해당 웨이퍼 프로젝트에서 상대적으로 더 정확히 예측하고 분류

결론

향후 연구 방향

1.Computational Efficiency(연산 효율성) 분석: 모바일넷V2는 작은 모델 크기와 낮은 연산량으로도 높은 성능을 보이는 경향이 있기 때문에 이를 확인할 필요가 있어 보임.

2.Generalization Ability(일반화 능력) 평가: 추가적인 테스트 데이터셋을 사용하여 모델의 일반화 능력을 평가할 필요가 있음. 이는 모델이 훈련된 데이터 이외의 데이터에서도 얼마나 잘 수행되는지를 확인하기 위함.

3.Hyperparameter Tuning(하이퍼파라미터 조정): 모델의 하이퍼파라미터를 조정하여 성능 최적화. 학습률, 배치 크기, 드롭아웃 비율 등의 하이퍼파라미터를 변경하여 모델의 성능에 미치는 영향을 확인할 필요가 있음.

4.Data Augmentation(데이터 증강) 적용: 데이터 증강 기법을 사용하여 훈련 데이터셋을 더 다양하게 만들어 모델의 성능 변화를 확인할 필요가 있음.

5.Fine-tuning(미세 조정) 실험: 사전 훈련된 모델을 가져와서 추가적인 미세 조정을 수행하여 성능을 개선할 수 있는지 확인 필요가 있음.

감사합니다