마스크 착용 여부 분류를 위한 ResNet-18 모델 구축

2024.04.05

빅데이터 6기 이신영

목차

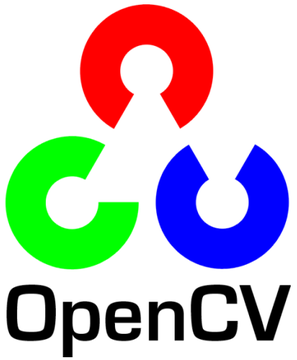
01 서론
1.1 연구 배경 및 목적

02 데이터셋
2.1 데이터셋 소개
2.2 데이터 전처리

03 모델 설계
3.1 모델 선정 과정
3.2 ResNet-18 모델 소개 및 구현
3.3 ResNet-18 모델 성능 평가 및 하이퍼파라미터 튜닝

04 결론
4.1 결과 요약
4.2 느낀 점 및 향후 방향

부록 및 참고문헌



OpenCV

- 오픈 소스 컴퓨터 비전 라이브러리
- 컴퓨터가 사람의 시각처럼 이미지나 비디오를 이해할 수 있도록 도와주는 프로그래밍 함수의 집합

PyTorch

- 인공지능 연구 및 개발을 위한 오픈소스 머신러닝 프레임워크
- 이해와 디버깅이 쉬운 직관적이고 간결한 코드로 구성되어 있음
- 파이토치 허브를 사용하여 사전 학습된 모델을 사용할 수 있음

- **OpenCV를 활용한 이미지 전처리**
- **CNN 모델**

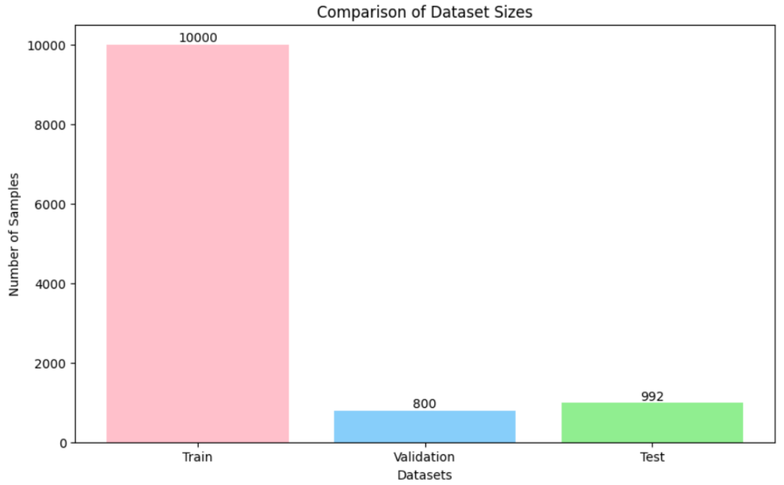
- **이미지 분류**
- **하이퍼파라미터 튜닝을 통한 모델 성능 최적화**

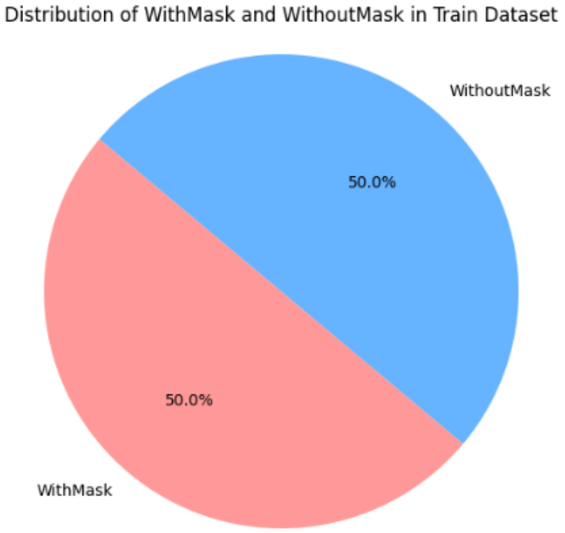
다양한 실험 시도와 최적의 결과 도출

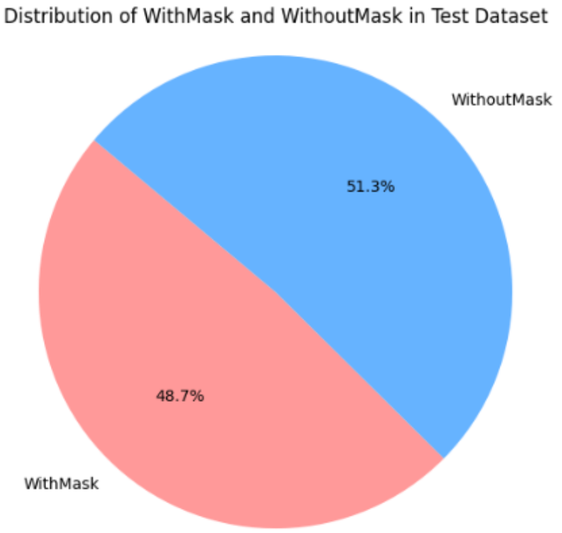
Face Mask Detection ~12K Images Dataset

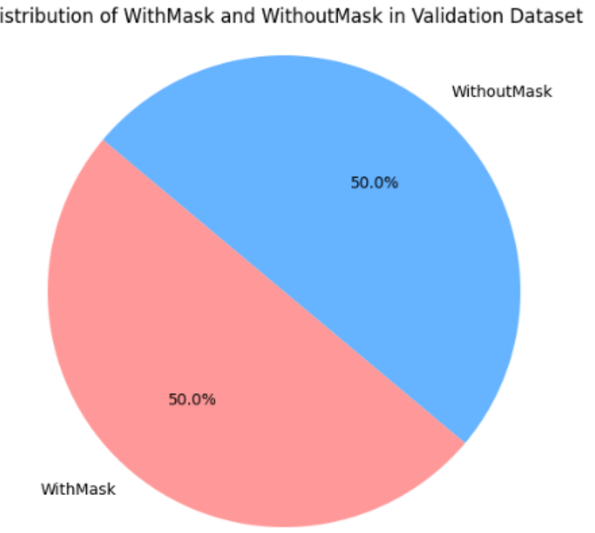
- 약 328.92MB
- 데이터 구조

전체 데이터 (11,792개)	Train 데이터 (10,000개)	WithMask(5,000개)
		WithoutMask(5,000개)
	Validation 데이터 (800개)	WithMask(400개)
		WithoutMask(400개)
	Test 데이터 (992개)	WithMask(483개)
		WithoutMask(509개)



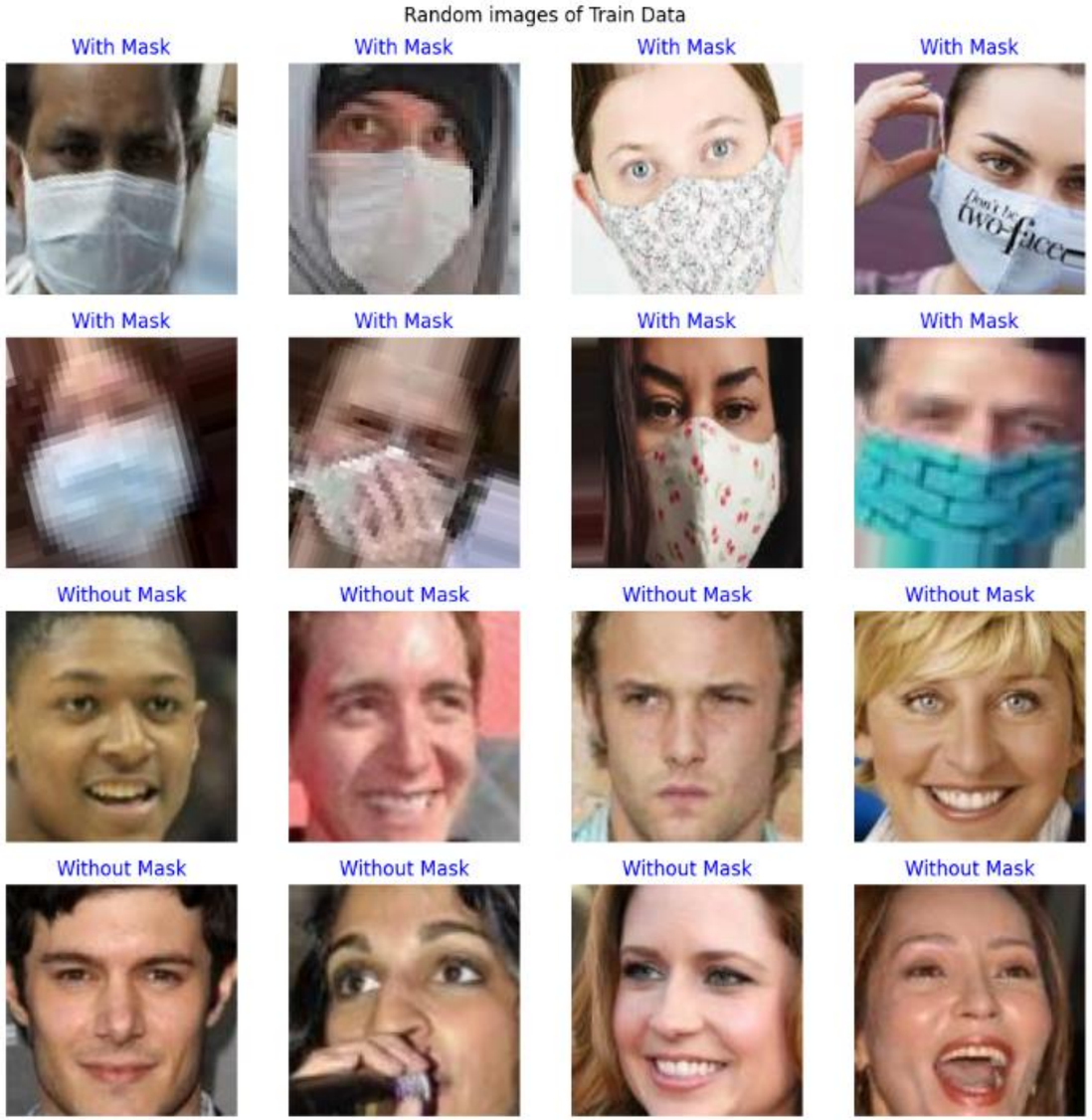






2.1 데이터셋 소개

- 데이터 예시



Resize

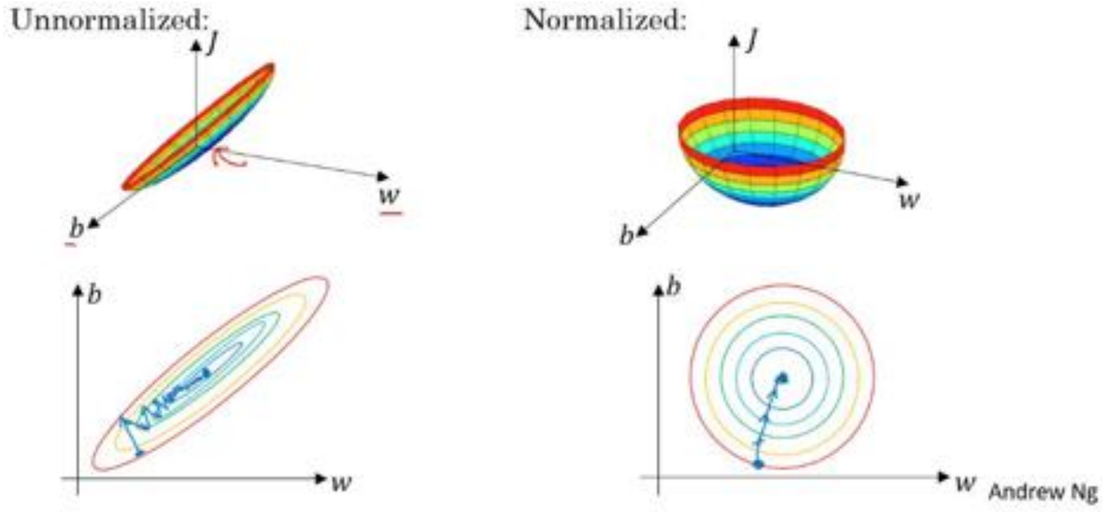
- CNN 모델은 특징 추출을 위해 Input data 형태에 맞추어 설계됨
- 모델마다 그 모델에 적합한 Input size로 전체 이미지 사이즈를 resize 진행

ToTensor

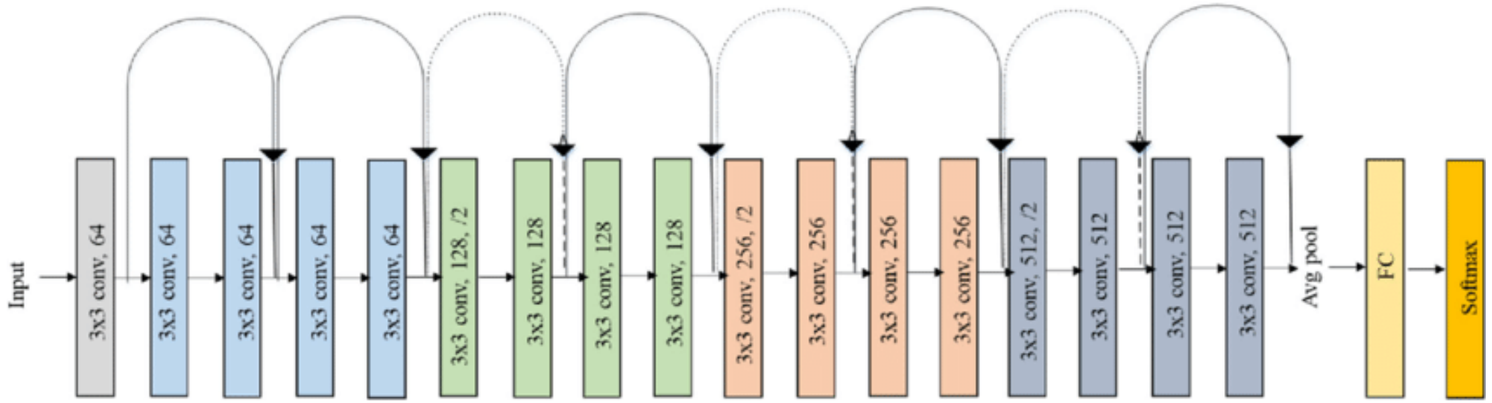
- Numpy 형태의 이미지를 torch의 (C, H, W) tensor 형태로 바꿔줌
- 픽셀 range 0~1 값으로 스케일링

Normalize

- 쉽게 최적값에 도달하고 빠르게 훈련할 수 있도록 정규화 진행
- Resize한 이미지를 학습할 것이기 때문에 resize한 데이터의 평균과 표준편차를 이용



ResNet-18 Architecture



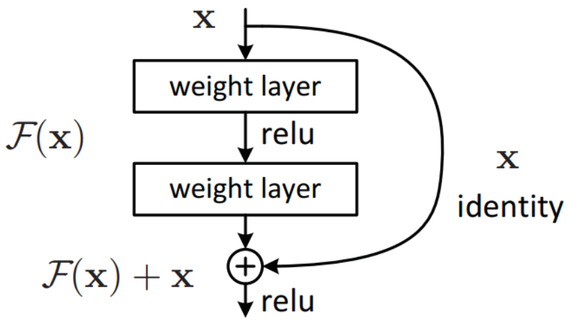


Figure 2. Residual learning: a building block.

- ResNet 시리즈 중 18개의 레이어로 구성되어 있는 모델
- **Skip Connection** 도입

  네트워크에서 발생하는 기울기 소실 문제를 완화하고, 더 깊은 네트워크를 효과적으로 학습할 수 있도록 함

3.2 ResNet-18 모델 구현

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5685, 0.4648, 0.4162], [0.2430, 0.2229, 0.2187])
])

train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32, shuffle=False)
val_loader = torch.utils.data.DataLoader(val_set, batch_size=32, shuffle=False)

next(iter(train_loader))[0].shape
# torch.Size([32, 3, 224, 224])
```

데이터 전처리

- 224X224로 resize
- Numpy 형태의 이미지를 tensor 형태로 변환
- Resize한 데이터의 평균과 표준편차를 구하여 정규화

DataLoad

- 데이터셋의 배치 크기를 32로 설정하여 load
- ‘shuffle’

  train_loader의 데이터를 무작위로 섞어

  학습 과정에서 다양한 데이터에 모델이 노출되도록 함

Train_loader shape 확인

- 배치 크기, 채널 수-RGB, 이미지 크기)

3.2 ResNet-18 모델 구현

```
class myResNet(nn.Module):
    def __init__(self):
        super(myResNet, self).__init__()
        self.resnet = models.resnet18(weights=None)
        num_ftrs = self.resnet.fc.in_features
        self.resnet.fc = nn.Linear(num_ftrs, 2)

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

model = myResNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

myResNet 클래스 정의

- nn.Module 상속
- 클래스의 초기화 메서드 정의
- 상위 클래스(nn.Module)의 초기화 메서드를 호출

ResNet-18 모델 load

- ‘weights=None’ : 모델이 미리 학습된 가중치를 사용하지 않음을 의미
- Fully Connected 레이어를 새로운 레이어로 대체
- ResNet-18 모델의 Fully Connected 레이어의 입력 특성의 수를 load
- 출력 노드 수를 2로 설정하여 모델이 이진 분류 작업에 사용될 수 있도록 함

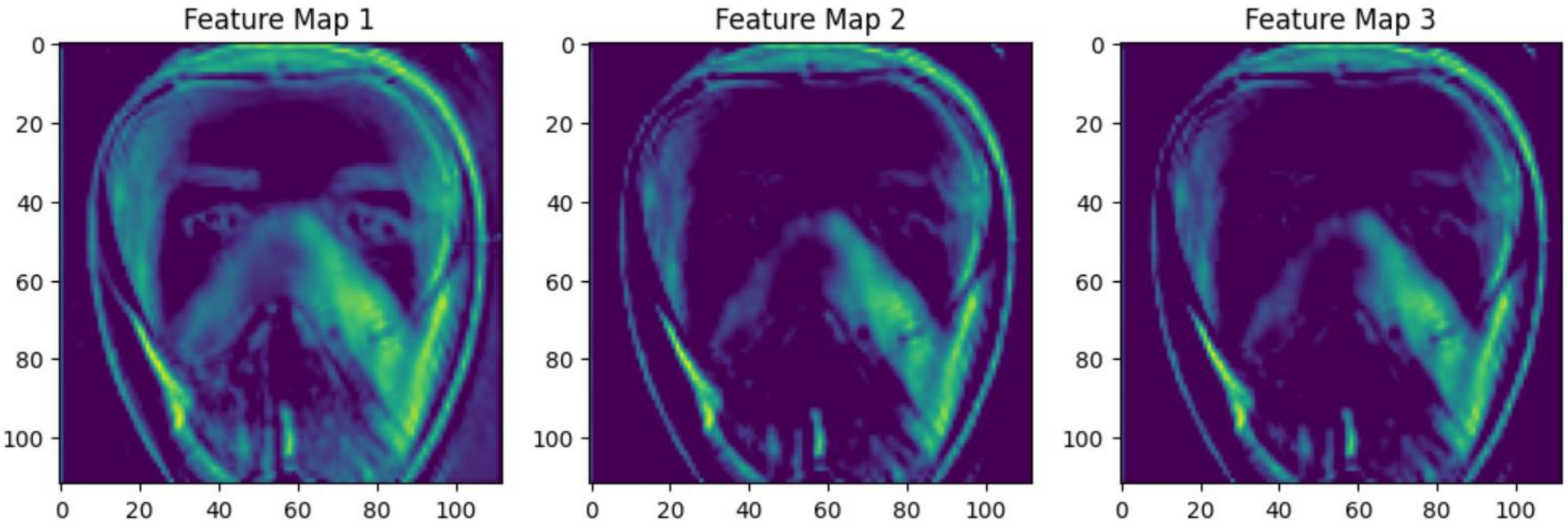
순전파 메서드 정의

- 입력 데이터 ‘x’를 ResNet-18 모델에 전달하여 출력을 반환

모델, 손실 함수, 최적화 알고리즘 설정

- 손실 함수 : 이진 분류에 일반적으로 사용되는 크로스 엔트로피 손실 함수를 사용
- 최적화 알고리즘 : 일반적으로 성능이 좋다고 알려진 Adam 사용

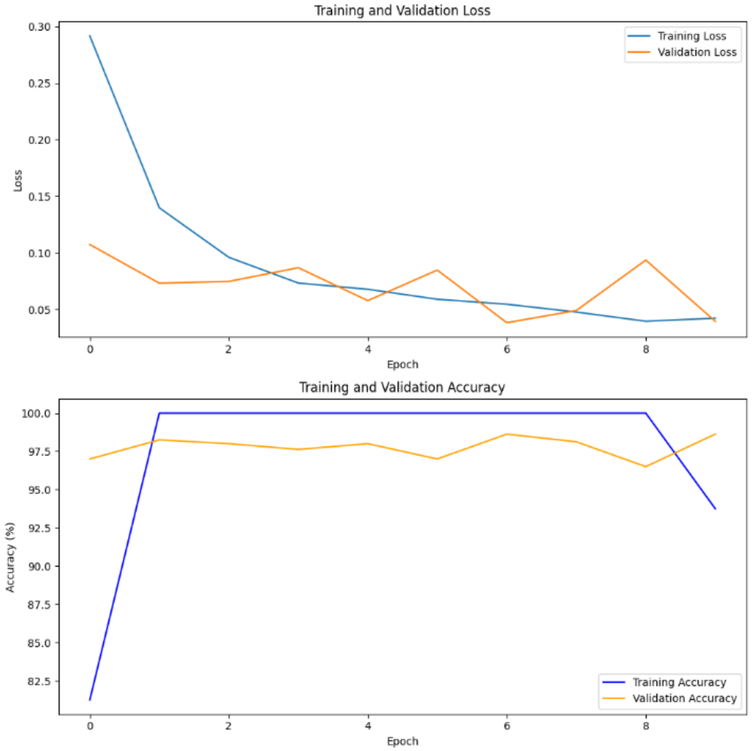
3.2 ResNet-18 모델 구현



3.3 ResNet-18 모델 성능 평가 및 하이퍼파라미터 튜닝

optimizer = Adam

```
optimizer = optim.Adam(model.parameters(), lr=0.01)
```



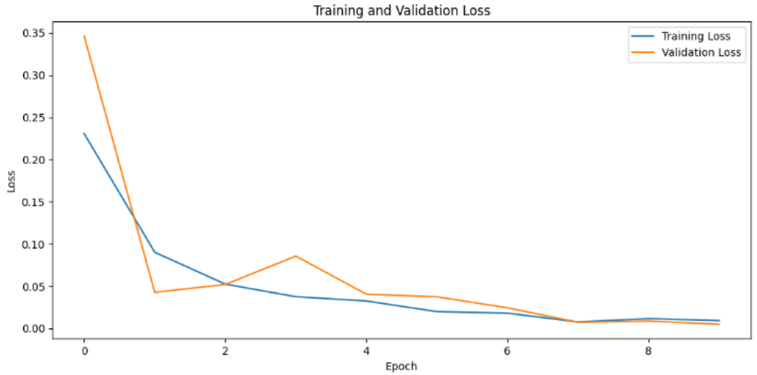

```
Epoch 1, Training loss: 0.2917
val loss: 0.1073, Accuracy: 97.00%
Epoch 2, Training loss: 0.1398
val loss: 0.0731, Accuracy: 98.25%
Epoch 3, Training loss: 0.0960
val loss: 0.0747, Accuracy: 98.00%
Epoch 4, Training loss: 0.0732
val loss: 0.0868, Accuracy: 97.62%
Epoch 5, Training loss: 0.0677
val loss: 0.0577, Accuracy: 98.00%
Epoch 6, Training loss: 0.0589
val loss: 0.0847, Accuracy: 97.00%
Epoch 7, Training loss: 0.0545
val loss: 0.0382, Accuracy: 98.62%
Epoch 8, Training loss: 0.0476
val loss: 0.0489, Accuracy: 98.12%
Epoch 9, Training loss: 0.0395
val loss: 0.0935, Accuracy: 96.50%
Epoch 10, Training loss: 0.0422
val loss: 0.0394, Accuracy: 98.62%
```

```
Test Loss: 0.050935367470130236
Test Accuracy: 0.9828629032258065
```

3.3 ResNet-18 모델 성능 평가 및 하이퍼파라미터 튜닝

optimizer = SGD

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)



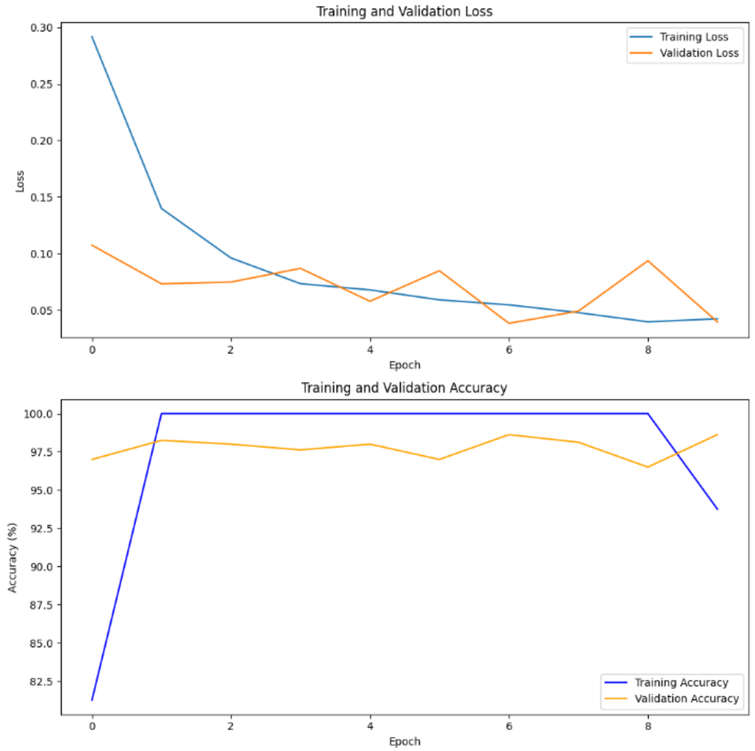

```
Epoch 1, Training loss: 0.2307
val loss: 0.3463, Accuracy: 90.00%
Epoch 2, Training loss: 0.0902
val loss: 0.0427, Accuracy: 98.38%
Epoch 3, Training loss: 0.0524
val loss: 0.0521, Accuracy: 98.00%
Epoch 4, Training loss: 0.0375
val loss: 0.0856, Accuracy: 97.12%
Epoch 5, Training loss: 0.0325
val loss: 0.0405, Accuracy: 98.88%
Epoch 6, Training loss: 0.0199
val loss: 0.0374, Accuracy: 98.62%
Epoch 7, Training loss: 0.0180
val loss: 0.0244, Accuracy: 98.88%
Epoch 8, Training loss: 0.0076
val loss: 0.0072, Accuracy: 99.88%
Epoch 9, Training loss: 0.0116
val loss: 0.0087, Accuracy: 99.50%
Epoch 10, Training loss: 0.0093
val loss: 0.0049, Accuracy: 99.75%
```

Test Loss: 0.008722839585032253
Test Accuracy: 0.9969758064516129

최종 모델 선정!

3.3 ResNet-18 모델 성능 평가 및 하이퍼파라미터 튜닝

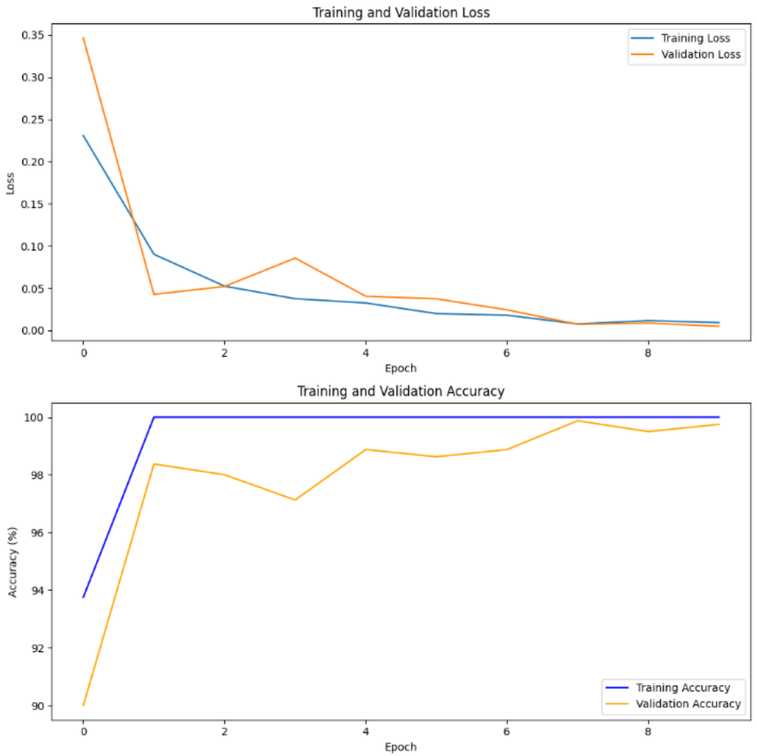
optimizer = Adam



Test Loss: 0.050935367470130236
Test Accuracy: 0.9828629032258065

최종 모델 선정!

optimizer = SGD



Test Loss: 0.008722839585032253
Test Accuracy: 0.9969758064516129

마스크 착용 여부 분류를 위한 ResNet-18 모델 구축

모델 선정

VGG-16

직접 구성한 CNN

ResNet -18

Input Size → (224, 224)

손실 함수 : CrossEntropyLoss

하이퍼파라미터 튜닝

Adam

SGD

lr=0.01, momentum=0.9

Test Loss: 0.008722839585032253
Test Accuracy: 0.9969758064516129

머신러닝은 경험이다!

SGD가 Adam보다 낮은 성능을 보인다는 실험 내용이 많았지만, 실제로 본 연구 결과에서는 SGD가 Adam보다 더 우수한 성능을 보였다. 이를 통해 하이퍼파라미터 튜닝에 정답은 없음을 경험하였고 실제 문제에 적합한 CNN 모델을 만들기 위해서는 많은 실험과 경험이 필요함을 알 수 있었다.

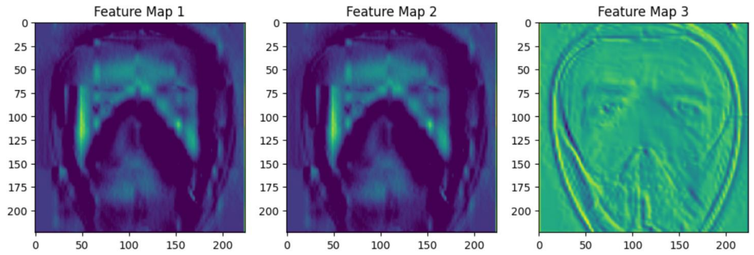
실험 환경의 아쉬움

연구 과정에서는 실험 환경으로 인해 더 많은 실험을 해보지 못한 아쉬움이 있었다. 모델의 구조를 단순화하거나 CPU 대신 GPU를 사용하는 등의 방법을 활용하였음에도 모델 학습 시간이 많이 소요되어 다양한 CNN 모델이나 하이퍼파라미터 조정을 더 시도하지 못한 것이 본 연구의 제약이 되었다. 따라서 향후 머신러닝을 진행한다면 실험 환경을 고려하여 모델의 구조에 변화를 주는 데에도 신경을 써야겠다고 생각했다.

- 많은 실험을 통해 모델의 성능을 개선하고, 다양한 하이퍼파라미터를 조정하여 최적의 설정을 찾는 것이 중요함을 알 수 있었다.
- 실험 과정에서 발생하는 다양한 문제들을 해결하며 모델 개발에 대한 실무적인 경험을 쌓을 수 있었다.
- 앞으로는 실험을 더욱 철저히 계획하고 진행하여 모델의 성능을 높이고 실제 문제에 대한 효과적인 해결책을 찾아 나가고자 한다.

부록

VGG-16

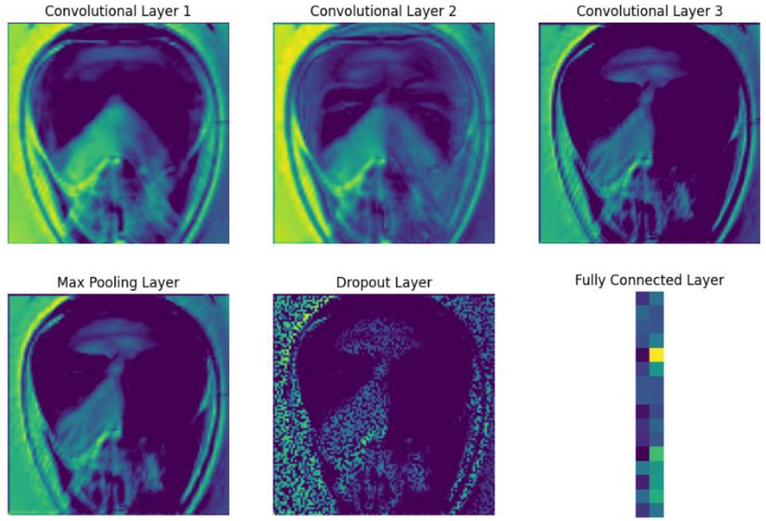



```
Epoch 1, Training loss: 871142677.6560
val loss: 0.6933, Accuracy: 50.00%
Epoch 2, Training loss: 0.6998
val loss: 0.6936, Accuracy: 50.00%
Epoch 3, Training loss: 0.6969
val loss: 0.6933, Accuracy: 50.00%
Epoch 4, Training loss: 0.6954
val loss: 0.6932, Accuracy: 50.00%
Epoch 5, Training loss: 0.6942
val loss: 0.6933, Accuracy: 50.00%
```

직접 구성한 CNN

```python
class MyCNN(nn.Module):
    def __init__(self):
        super(MyCNN, self).__init__()
        # Convolutional layers
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride=1, padding=1)
        # Max pooling layers
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        # Dropout layer
        self.dropout = nn.Dropout(0.5)
        # Fully connected layer
        self.fc = nn.Linear(32 * 4 * 4, 2)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)
        x = x.view(-1, 32 * 4 * 4)
        x = self.dropout(x)
        x = self.fc(x)
        return x
```




```
Epoch 1, Training loss: 0.2665
val loss: 0.2006, Accuracy: 93.38%
Epoch 2, Training loss: 0.2072
val loss: 0.1017, Accuracy: 96.75%
Epoch 3, Training loss: 0.1881
val loss: 0.1543, Accuracy: 93.50%
Epoch 4, Training loss: 0.1706
val loss: 0.1449, Accuracy: 93.00%
Epoch 5, Training loss: 0.1741
val loss: 0.1584, Accuracy: 94.62%
Epoch 6, Training loss: 0.2099
val loss: 0.1593, Accuracy: 93.75%
Epoch 7, Training loss: 0.1817
val loss: 0.1125, Accuracy: 96.00%
Epoch 8, Training loss: 0.1790
val loss: 0.1265, Accuracy: 94.12%
Epoch 9, Training loss: 0.1827
val loss: 0.1081, Accuracy: 95.12%
Epoch 10, Training loss: 0.2176
val loss: 0.1406, Accuracy: 94.62%
```

참고문헌

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations (ICLR).