

2023 데이터 크리에이터 캠프

DATA CREATOR CAMP



대학부

DASH

팀장 박소영

김연수

유건영

이승우

목차

■ 예선 진행 과정 및 결과 요약

■ 모델 개선 및 학습과정

- Augmentation(ResNet18)
- SimCLR(ResNet50)
- DenseNet
- EfficientNet

■ 결과 분석

- 모델 평가 및 결과 분석
- 마무리


예선 진행 과정

ResNet 모델 및 Batch Size 변경, Augmentation 적용 등 다양한 시도

	Model	Data Augmentation	학습 epoch수	정확도
Case1	ResNet18	X	50	70.39%
Case2	ResNet18	O	48	43%
Case3	ResNet50	X	50	63%

예선 진행 과정

ResNet 모델 및 Batch Size 변경, Augmentation 적용 등 다양한 시도



	Model	Data Augmentation	학습 epoch수	정확도
Case1	ResNet18	X	50	70.39%
Case2	ResNet18	O	48	43%
Case3	ResNet50	X	50	63%

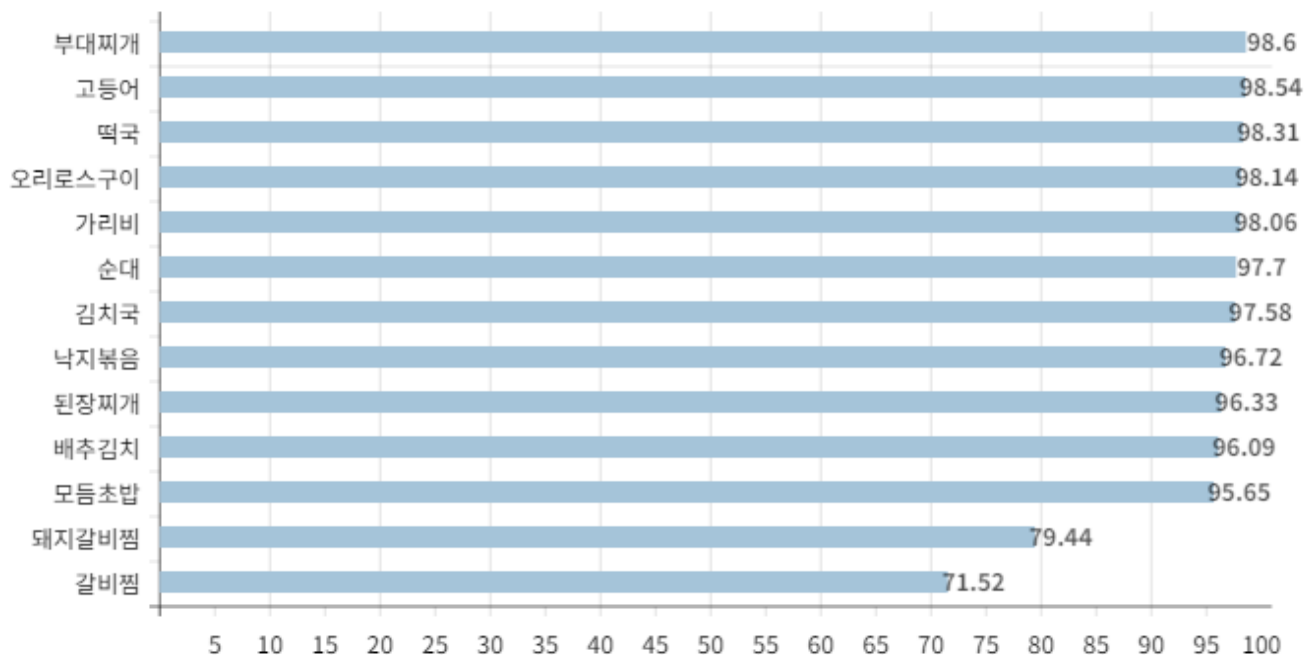
최종 모델 선정 : **Case1**

예선 결과 요약

모델 평가 및 결과 분석

클래스별 정확도

(단위 : %)



Validation Accuracy : **94.16%**

13개 중 11개 음식 분류 정확도 **95% 이상**

모든 클래스의 분류 정확도 **70% 이상**



예선 진행 후기

이미지에서의 음식 색상, 형태의 중요성 확인 -> Data Augmentation 참고



음식
색상

음식
형태



Data
Augmentation

모델 개선 - Augmentaion

Data Augmentation 문제점

```
# 이미지 전처리 및 데이터셋 설정
transform_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.RandomHorizontalFlip(), # 좌우 반전
    transforms.RandomVerticalFlip(), # 상하 반전
    transforms.RandomRotation(90), # 90도 회전
    transforms.ColorJitter(
        brightness=(0.5, 2), # 밝기
        contrast=(0.5, 1.5), # 대비
        saturation=(0.8, 1.5), # 채도
    ),
    transforms.RandomResizedCrop(
        size=(240, 240), # 잘라내고 조절할 크기
        scale=(0.8, 1.2) # 스케일 범위 (줄인 및 zoom 효과)
    )
])
transform_val = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
train_dataset = ImageFolder(train_dir, transform=transform_train)
val_dataset = ImageFolder(val_dir, transform=transform_val)

# 데이터로더 설정
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
```

<이전 Data Augmentation 결과>

Accuracy: 70.39% -> 43% 성능이 저하됨

성능 저하의 원인

음식 이미지에서 색을 임의로 바꿈



개선 방향

Data Augmentation과정에서 색을 바꾸지 않음

모델 개선 - Augmentaion

Data Augmentation 수정

```
transform_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=(0, 180)),
    transforms.RandomResizedCrop(
        size=(240, 240),          # 잘라내고 조절할 크기
        scale=(0.8, 1.2)         # 스케일 범위 (줄인 및 줌아웃 효과)
    )
])
transform_val = transforms.Compose([transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456,
0.406), (0.229, 0.224, 0.225))])
```

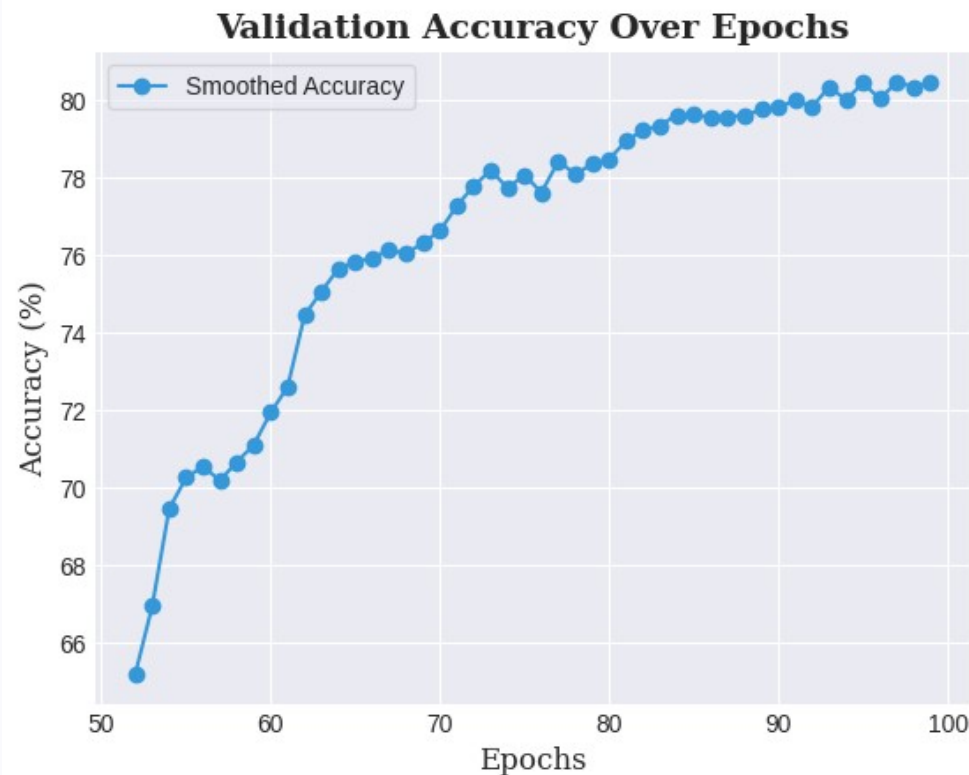
① Color jittering(brightness, contrast, saturation) **제거**
:음식 이미지에서 색을 바꾸는 부분 삭제

② Transforms.RandomRotation(degrees=(0, 180)) **변경**
: 90도만 회전하는 것에서 0~180도 사이
무작위 회전으로 변경

<수정된 Augmentation 결과>

모델 개선 - Augmentaion

Data Augmentation 결과



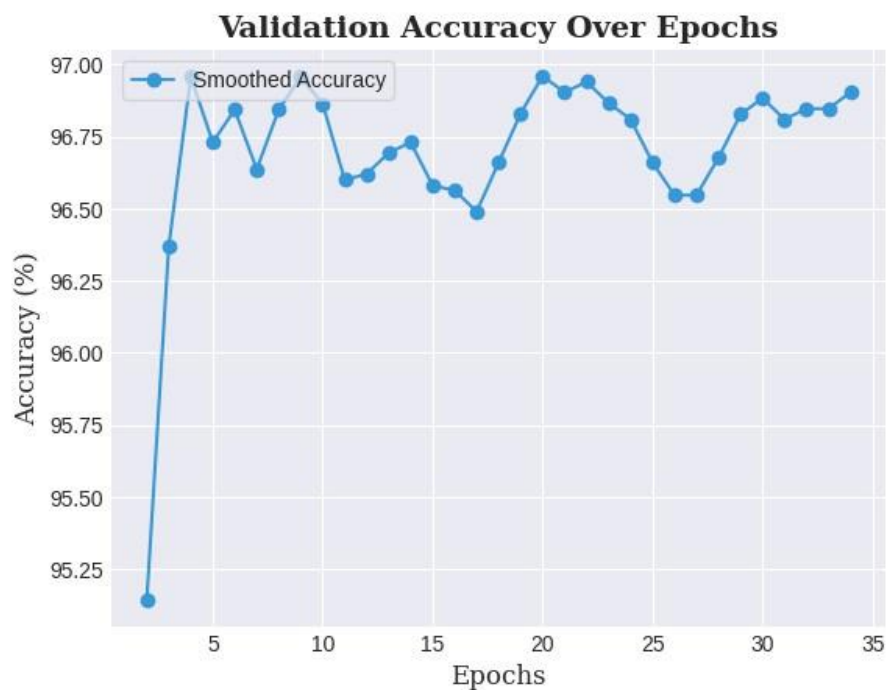
<에폭별 검증 정확도 그래프>

예선 최종 모델(resnet18, 50epoch)에서
Data Augmentation을 적용한 후
50epochs 학습 시킨 결과

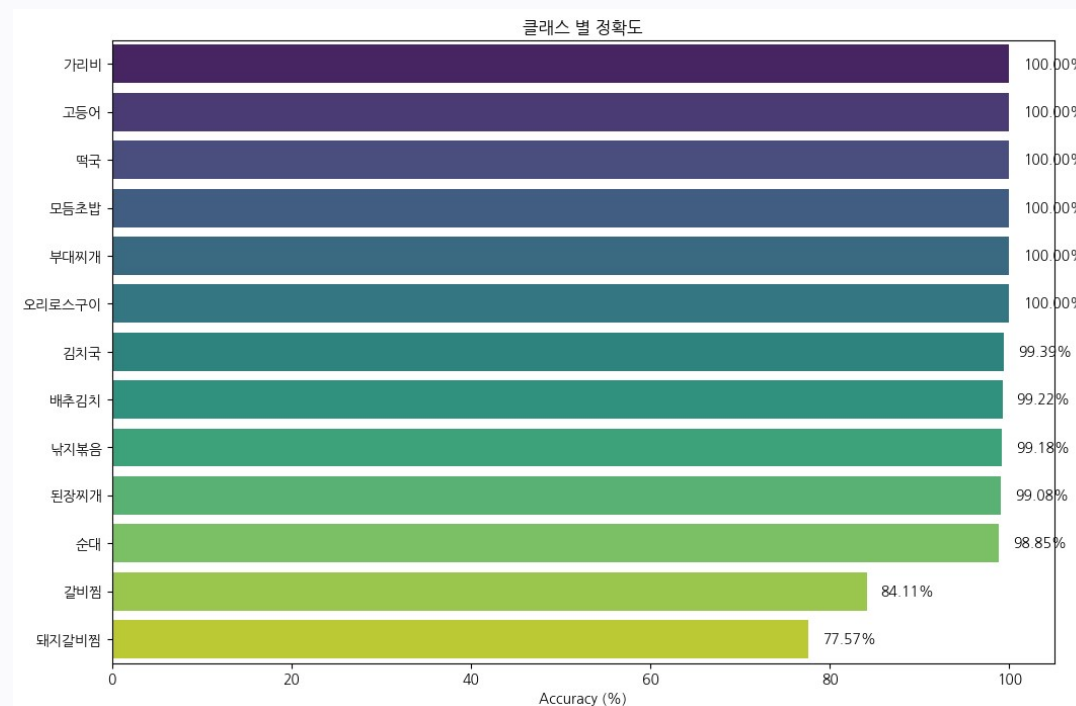
Accuracy: **80.63%**

모델 개선 - Augmentaion

Kfood_health 데이터셋 분류



<35epoch (linear probing) 검증 정확도 그래프>



<메뉴별 정확도 그래프>

최종 모델 검증 정확도: **97%**

모델 개선 - Augmentaion

개선 후 결과 비교

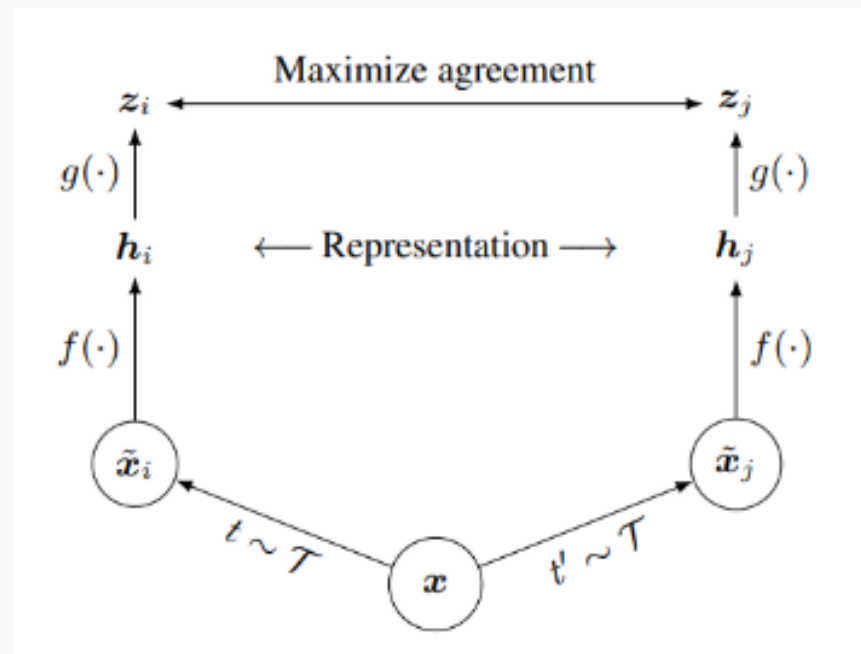
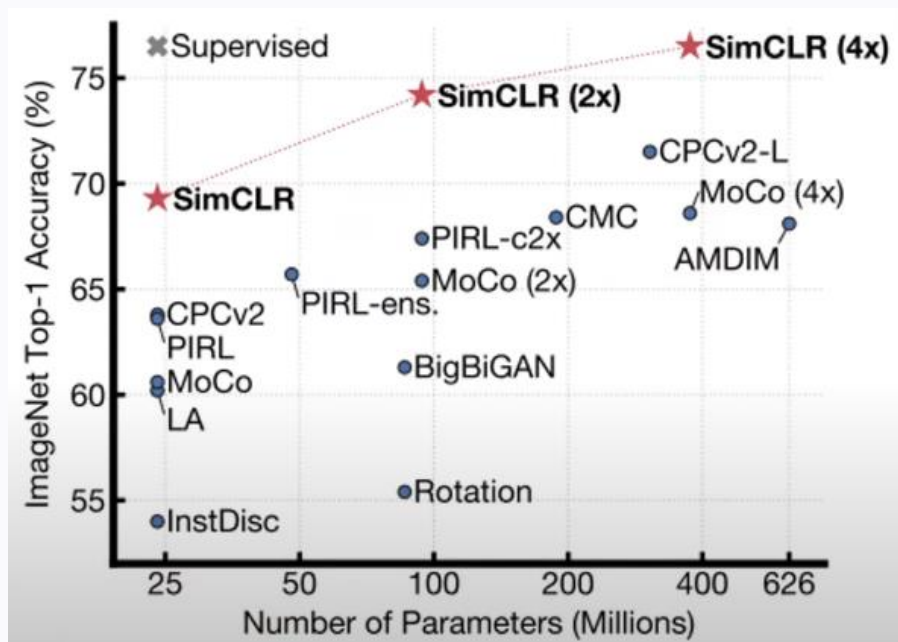
	Data Augmentation	Model	음식 데이터	K-food 데이터
예선 최종 모델	X	ResNet18	70.39%	94.16%
개선 모델	O	ResNet18	80.63%	97%

- 모델 성능이 94.16%에서 97%로 약 3% 향상
- 이미지 특성에 맞는 Data Augmentation이 중요

모델 개선 - SimCLR

A Simple Framework for Contrastive Learning of visual Representations

- 비슷한 이미지끼리는 더 가까운 벡터, 다른 이미지끼리는 더 먼 벡터로!
더욱 다양하고 강력한 특징 학습 가능 (사전 훈련 단계에서 유용할 것으로 판단)



모델 개선 - SimCLR

실행 코드

- 사전에 훈련된 모델을 활용하여 클래스 생성

```
#AugmentedDataset 클래스 생성
class AugmentedDataset(torch.utils.data.Dataset):
    def __init__(self, dataset, transform=None):
        self.dataset = dataset
        self.transform = transform

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        image, label = self.dataset[idx]
        if self.transform:
            augmented_image = self.transform(image)
        else:
            augmented_image = image
        return image, augmented_image, label

# cutout_and_rotate 함수를 transform 인자로 전달
augmented_train_dataset = AugmentedDataset(train_dataset, transform=cutout_and_rotate)
augmented_train_loader = DataLoader(augmented_train_dataset, batch_size=batch_size, shuffle=True, drop_last=True)

class ResNet50Model(nn.Module):
    def __init__(self, num_classes):
        super(ResNet50Model, self).__init__()
        # resnet50 모델 불러오기 (사전 훈련된 가중치 포함)
        self.resnet50 = models.resnet50(pretrained=True)
        self.fc = nn.Linear(self.resnet50.fc.in_features, num_classes)

        # 마지막 완전 연결층 교체 (기존 모델의 출력 크기는 1000)
        self.resnet50.fc = nn.Linear(self.resnet50.fc.in_features, num_classes)

    def forward(self, x):
        # resnet50 모델을 통한 순전파
        return self.resnet50(x)

# 모델
num_classes = 42 # 출력 크기 설정
model = ResNet50Model(num_classes)
```

모델 개선 - SimCLR

실행 코드

- 프레임워크의 일부로 사용되는 손실 함수 구현

```
#SimCLR_LOSS 모델 생성
class SimCLR_Loss(nn.Module):
    def __init__(self, batch_size, temperature):
        super().__init__()
        self.batch_size = batch_size
        self.temperature = temperature

        self.mask = self.mask_correlated_samples(batch_size)
        self.criterion = nn.CrossEntropyLoss(reduction="sum")
        self.similarity_f = nn.CosineSimilarity(dim=2)

    # loss 분모 부분의 negative sample 간의 내적 합만을 가져오기 위한 마스킹 행렬
    def mask_correlated_samples(self, batch_size):
        N = 2 * batch_size
        mask = torch.ones((N, N), dtype=bool)
        mask = mask.fill_diagonal_(0)

        for i in range(batch_size):
            mask[i, batch_size + i] = 0
            mask[batch_size + i, i] = 0
        return mask
```

```
def forward(self, z_i, z_j):
    real_batch_size = z_i.size(0) # 실제 배치 크기
    N = 2 * real_batch_size

    z = torch.cat((z_i, z_j), dim=0)
    sim = self.similarity_f(z.unsqueeze(1), z.unsqueeze(0)) / self.temperature

    # 실제 배치 크기에 맞춰 대각선 요소를 추출합니다.
    sim_i_j = torch.diag(sim, real_batch_size)
    sim_j_i = torch.diag(sim, -real_batch_size)

    # 추출된 대각선 요소의 크기에 맞춰 reshape를 수행합니다.
    positive_samples = torch.cat((sim_i_j, sim_j_i), dim=0).view(-1, 1)
    # 마스크를 실제 텐서 크기에 맞춰 조정합니다.
    mask = self.mask[:N, :N]
    negative_samples = sim[mask].view(N, -1)

    labels = torch.zeros(N, dtype=torch.long, device=z.device)

    logits = torch.cat((positive_samples, negative_samples), dim=1)
    loss = self.criterion(logits, labels)
    loss /= N

    return loss
```

양의 샘플, 음의 샘플 추출하고
양의 샘플(정답), 음의 샘플(오답) 처리하여
손실 계산

모델 개선 - SimCLR

실행 코드

- **cutout 적용**하여 이미지 일부를 임의 삭제 및 회색 마킹

```
def cutout_and_rotate(image):  
    # 이미지 복사  
    image = image.clone()  
  
    # Cutout 적용  
    x_start = np.random.randint(20) # cutout 시작할 x축 위치  
    y_start = np.random.randint(20) # cutout 시작할 y축 위치  
    image[:, x_start:x_start+9, y_start:y_start+9] = 0.5 # 회색으로 마킹  
  
    # 이미지 회전  
    image = torch.rot90(image, 1, [1, 2]) # 마지막 두 axis 기준 90도 회전  
  
    return image
```

```
# 이미지 전처리 설정  
transform = transforms.Compose([  
    transforms.Resize((224, 224), antialias=True),  
    transforms.ToTensor(),  
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),  
    transforms.RandomHorizontalFlip(),  
    transforms.RandomVerticalFlip(),  
    transforms.RandomRotation(degrees=(0, 180)),  
    transforms.RandomResizedCrop(size=(240, 240), scale=(0.8, 1.2)),  
    transforms.Lambda(cutout_and_rotate)  
])
```

전체적인 이미지는 물론, 덜 중요한 부분에도 주목하도록 유도

모델 개선 - SimCLR

결과

- 최종 정확도 94.50% (0.34% ↑)

train

```
Epoch : 71, Avg Loss : 1.5174  
100%|██████████| 2135/2135 [12:46<00:00, 2.79it/s]  
Epoch : 72, Avg Loss : 1.5176  
100%|██████████| 2135/2135 [12:46<00:00, 2.79it/s]  
Epoch : 73, Avg Loss : 1.5174  
이어서 학습이 완료되었습니다.
```

kfood_health_val

```
Epoch [21/30], Loss: 0.0715, Validation Accuracy: 89.06%  
Epoch [22/30], Loss: 0.0712, Validation Accuracy: 93.42%  
Epoch [23/30], Loss: 0.1031, Validation Accuracy: 94.50%
```


모델 개선 - EfficientNet

EfficientNet_b0

● 모델 정의

```
# 모델 정의 (efficientnet_b0)
efficientnet_b0 = timm.create_model('efficientnet_b0', pretrained=True)

# Reset Parameters (가중치 초기화)
def reset_parameters(module):
    if hasattr(module, 'reset_parameters'):
        module.reset_parameters()

efficientnet_b0.apply(reset_parameters)

# 마지막 Fully Connected Layer 변경
num_classes = 42 # 클래스 수
efficientnet_b0.fc = nn.Linear(2048, num_classes)

# 모델을 GPU로 이동
device = 'cuda' if torch.cuda.is_available() else 'cpu'
efficientnet_b0.to(device)
```

● 데이터 전처리 및 데이터셋 설정

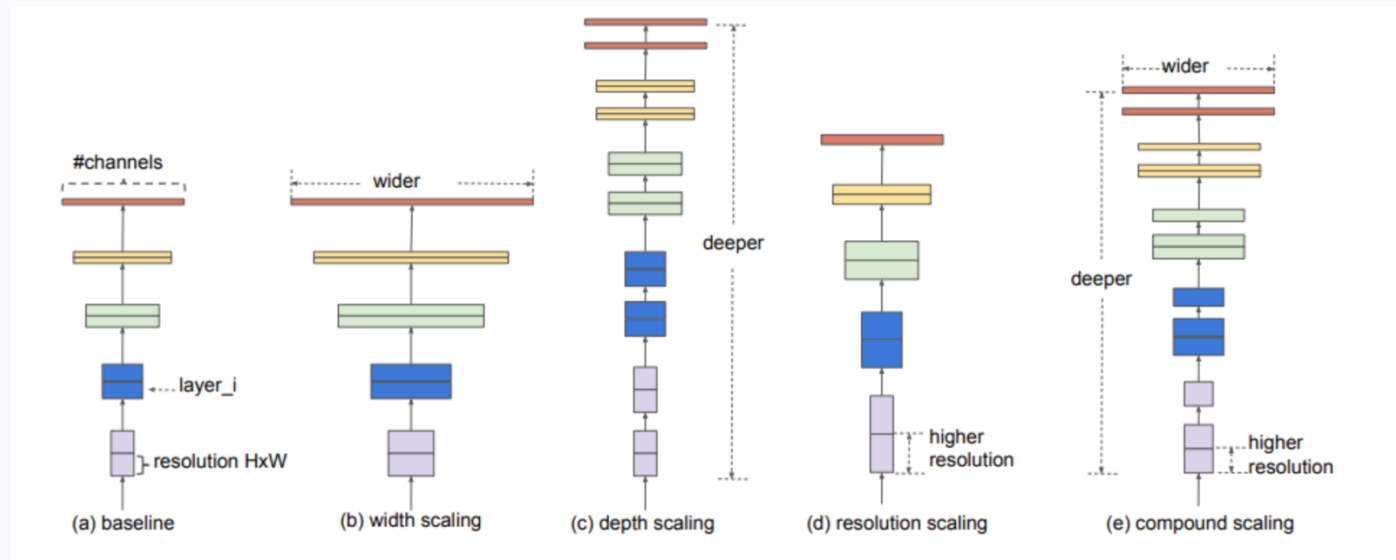
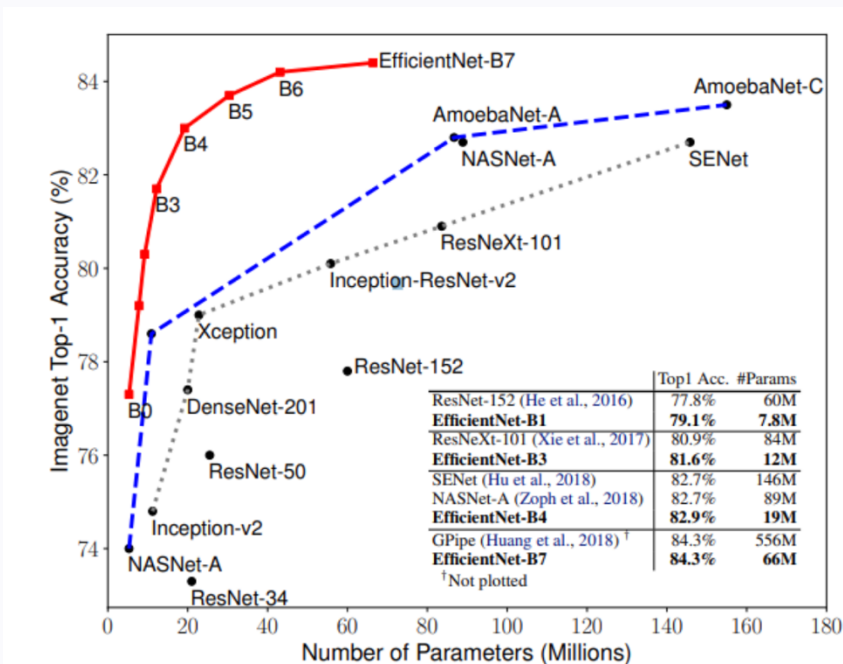
```
# 이미지 전처리 및 데이터셋 설정
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(degrees=(0, 180)),
    transforms.RandomResizedCrop(
        size=(240, 240), # 잘라내고 조절할 크기
        scale=(0.8, 1.2) # 스케일 범위 (줄임 및 zoom 효과)
    )
])

train_dataset = ImageFolder(train_dir, transform=transform)
val_dataset = ImageFolder(val_dir, transform=transform)

# 데이터로더 설정
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
```

모델 개선 - EfficientNet

- Neural architecture 사용 -> new baseline network 설계, scale up
- 이전 ConvNet 보다 더 높은 정확성과 효율성을 가진 모델

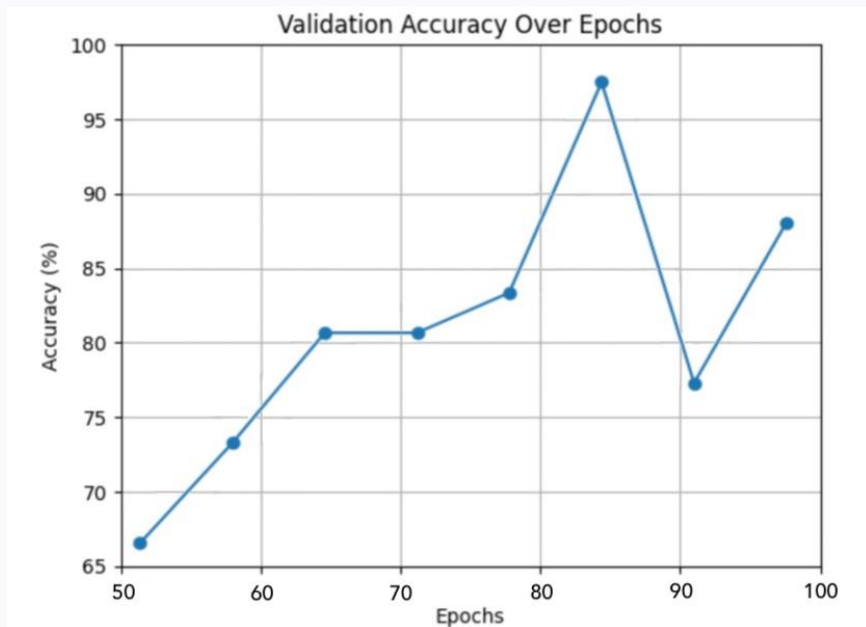


< EfficientNet network >

모델 개선 - EfficientNet

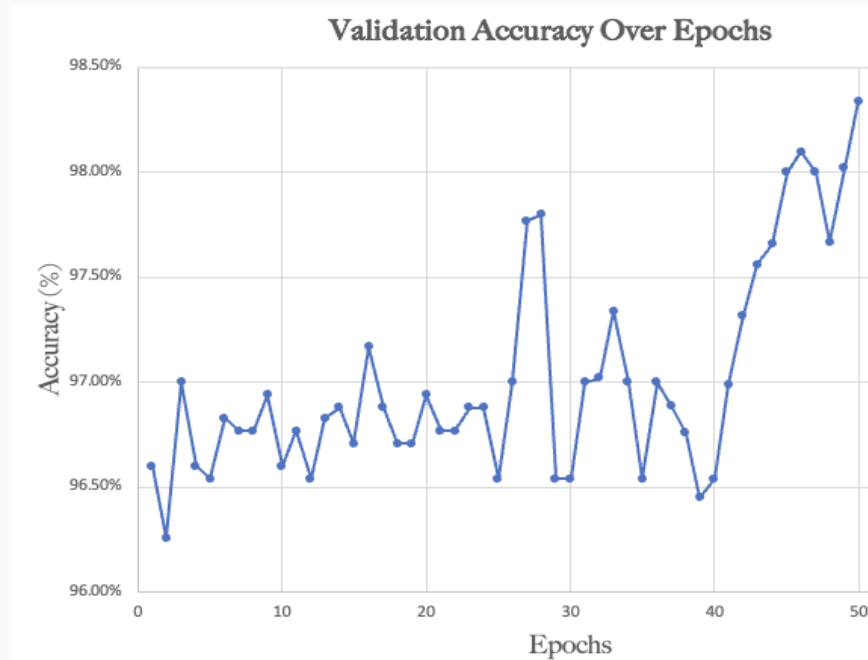
Augmentation 및 kfoodhealth 데이터 분류

- EfficientNet



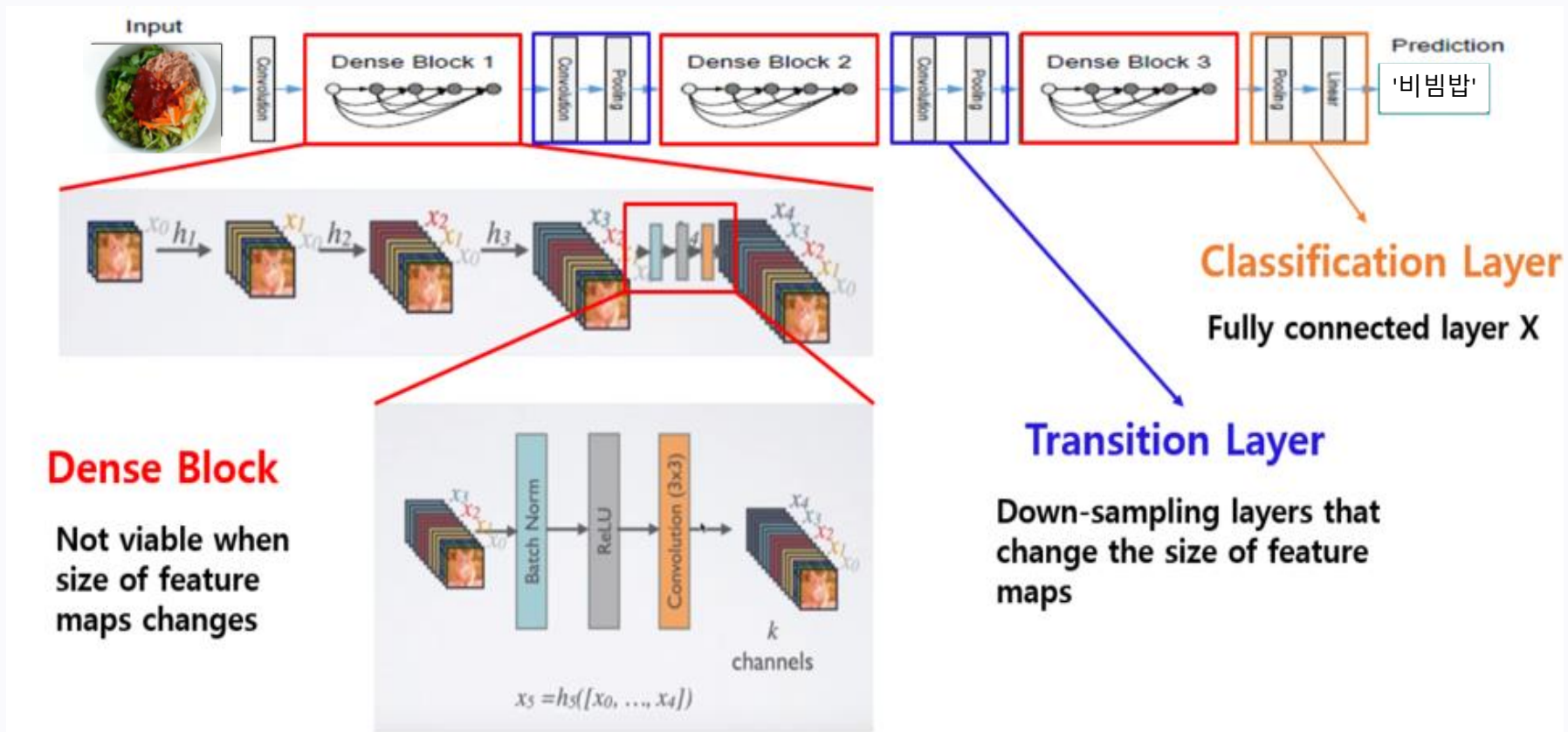
< 에폭 별 검증 정확도 그래프 >

- Kfoodhealth 데이터셋 분류



모델 개선 - DenseNet

- Resnet의 Skip-connection에서 발전된 Dense-connection
- Bottleneck layers를 적용



모델 개선 - DenseNet

실행 코드

● 모델정의

```
# 모델 정의(densenet)
densenet = models.densenet161(pretrained=True)

# Reset Parameters (가중치 초기화)
def reset_parameters(module):
    if hasattr(module, 'reset_parameters'):
        module.reset_parameters()

densenet.apply(reset_parameters)

# 마지막 Fully Connected Layer 변경
num_classes = 42 # 클래스 수
densenet.fc = nn.Linear(2048, num_classes)

# 모델을 GPU로 이동
device = 'cuda' if torch.cuda.is_available() else 'cpu'
densenet.to(device)
```

● 데이터 전처리 및 데이터셋 설정

```
# 이미지 전처리 및 데이터셋 설정
transform = transforms.Compose([transforms.Resize((224, 224)),
                                transforms.ToTensor(),
                                transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])

train_dataset = ImageFolder(train_dir, transform=transform)
val_dataset = ImageFolder(val_dir, transform=transform)

# 데이터로더 설정
batch_size = 16
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
```

모델 개선 - DenseNet

Train 데이터 분류 결과(에폭별 Loss)

```
Epoch 1, Batch 500, Loss: 18.559
Epoch 1, Batch 1000, Loss: 16.437
Epoch 1, Batch 1500, Loss: 15.660
Epoch 1, Batch 2000, Loss: 14.794
Epoch 2, Batch 500, Loss: 14.316
Epoch 2, Batch 1000, Loss: 13.731
Epoch 2, Batch 1500, Loss: 13.429
Epoch 2, Batch 2000, Loss: 13.126
Epoch 3, Batch 500, Loss: 12.565
Epoch 3, Batch 1000, Loss: 12.028
Epoch 3, Batch 1500, Loss: 11.658
Epoch 3, Batch 2000, Loss: 11.204
Epoch 4, Batch 500, Loss: 10.578
Epoch 4, Batch 1000, Loss: 10.543
Epoch 4, Batch 1500, Loss: 10.382
Epoch 4, Batch 2000, Loss: 10.089
```

```
Epoch 34, Batch 500, Loss: 0.193
Epoch 34, Batch 1000, Loss: 0.197
Epoch 34, Batch 1500, Loss: 0.383
Epoch 34, Batch 2000, Loss: 0.346
Epoch 35, Batch 500, Loss: 0.271
Epoch 35, Batch 1000, Loss: 0.169
Epoch 35, Batch 1500, Loss: 0.247
Epoch 35, Batch 2000, Loss: 0.229
Epoch 36, Batch 500, Loss: 0.149
Epoch 36, Batch 1000, Loss: 0.156
Epoch 36, Batch 1500, Loss: 0.182
Epoch 36, Batch 2000, Loss: 0.253
Epoch 37, Batch 500, Loss: 0.235
Epoch 37, Batch 1000, Loss: 0.186
Epoch 37, Batch 1500, Loss: 0.271
Epoch 37, Batch 2000, Loss: 0.220
Epoch 38, Batch 500, Loss: 0.173
Epoch 38, Batch 1000, Loss: 0.216
Epoch 38, Batch 1500, Loss: 0.179
Epoch 38, Batch 2000, Loss: 0.193
Epoch 39, Batch 500, Loss: 0.197
Epoch 39, Batch 1000, Loss: 0.146
```

최종 결과

모델 결과 비교

	Data Augmentation	Model	정확도(K-food 데이터)
예선 최종 모델	X	ResNet18	94.16%
Data Augmentation	O	ResNet18	97%
SimCLR	O	ResNet50	94.5%
EfficientNet	O	EfficientNet-B0	98%

최종 모델 : Data Augmentation을 적용한 EfficientNet-B0모델

결론

성능 향상 요인

1

Data Augmentation 추가

음식 이미지의 특성을 고려하여 색 변화를 하지 않고 Augmentation을 진행
94.16%에서 97%로 예선 최종 모델 대비 약 3% 성능 향상
Data Augmentation은 성능을 향상시키는 중요한 요소

2

SimCLR 모델

ResNet18모델에서 ResNet50모델로 변경 및 SimCLR 적용
다양한 증강기술 적용시도, 94.16%에서 94.5%로 성능 향상
ResNet 모델에서 얻은 특징을 활용하여 **미세한 구조**와 **패턴**까지 학습

3

EfficientNet-b0 모델

ResNet18모델에서 EfficientNet-b0 모델로 변경하여 진행
94.16%에서 98%로 예선 최종 모델 대비 약 4% 성능 향상
ResNet 모델보다 더 **높은 정확도**와 **효율성**을 가짐

결론

소감

아쉬운 점

본래 모델 가중치 사용하여
가중치 초기화한 모델과
결과 비교 분석하려 했으나
제한된 시간으로 진행 못함

개선사항

이미지데이터를 4분할하여
학습

Augmentation 변경
ex)명도 추가

배운점

ResNet, EfficientNet, SimCLR
과 같은 다양한 딥러닝 모델
델의 특징을 알게 됨

데이터 증강 및 모델 변경
등 다양한 방법을 적용하여
모델의 성능을 향상시킬 수
있다는 것을 배움

감사합니다.

대학부

DASH

팀장 박소영

김연수

유건영

이승우

2023 데이터 크리에이터 캠프
DATA CREATOR CAMP