2023 데이터 크리에이터 캠프

DATA CREATOR CAMP

대학부 **DASH**

팀장 박소영 김연수 유건영 이승우

목차

Mission1

1-1. 각 클래스로 하는 분류 데이터셋과 데이터로더 준비

```
# 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)
```

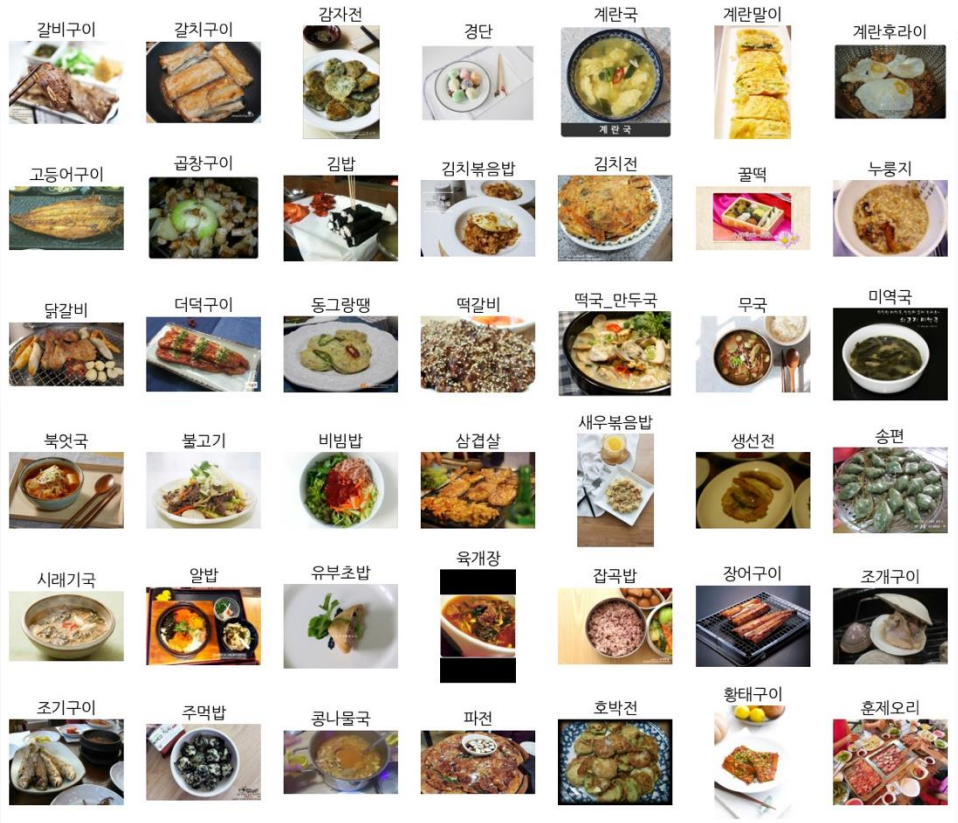
```
#데이터로더 설정
batch_size = 16
trainloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
valloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
```

Mission1

1-1. 총 42장 한번에 시각화(plotting)하여 확인

```
menu_folders = os.listdir(train_dir)
menu_folders = natsort.natsorted(menu_folders)
```

```
plt.figure(figsize=(15, 15))

for i, trainfolder in enumerate(menu_folders):
    trainmenu_path = os.path.join(train_dir, trainfolder)
    imgfiles = os.listdir(trainmenu_path)

    for j, imgfile in enumerate(imgfiles[:1]):  # Take 1 image from each folder
        imgpath = os.path.join(trainmenu_path, imgfile)
        img = mpimg.imread(imgpath)

        name = unicodedata.normalize('NFC', trainfolder)  # Normalize the folder name
        plt.subplot(7, 7, i + 1)
        plt.imshow(img)
        plt.title(name)
        plt.axis('off')

plt.show()
```



Mission1

1-2. ResNet18을 활용하여 모델 정의

```
# 모델 정의 (ResNet-18)
resnet18 = models.resnet18(pretrained=False)

# 마지막 Fully Connected Layer 변경
num_classes = 42  # 클래스 수
resnet18.fc = nn.Linear(resnet18.fc.in_features, num_classes)
# 모델을 GPU로 이동
device = 'cuda' if torch.cuda.is_available() else 'cpu'
resnet18.to(device)
```

Mission1

1-2. ResNet18을 활용하여 모델 학습

```
# Training loop
num_epochs = 50
save_epochs = 1

for epoch in range(num_epochs):
    resnet18.train()
    running_loss = 0.0

    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()

        outputs = resnet18(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 500 == 499:
            print(f'Epoch {epoch + 1}, Batch {i + 1}, Loss: {running_loss / 100:.3f}')
            running_loss = 0.0

    # 학습이 완료된 후 모델 저장
    if (epoch + 1) % save_epochs == 0:
        checkpoint = {
            'epoch': epoch + 1,
            'model_state_dict': resnet18.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss.item(),
        }
        torch.save(checkpoint, model_save_path)
```

Mission1

1-2. Validation 데이터에 대한 모델 검증

```python
# 검증 데이터셋을 이용하여 accuracy를 계산하고 출력
    if epoch % validation_epochs == 0:
        resnet18.eval()  # 모델을 평가 모드로 전환
        correct = 0
        total = 0

        with torch.no_grad():
            for data in val_loader:
                inputs, labels = data[0].to(device), data[1].to(device)
                outputs = resnet18(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
            # 잘못 분류된 경우 확인

            misclassified_mask = predicted != labels
            misclassified_examples.extend([(inputs[i], predicted[i].item(), labels[i].item()) for i, is_misclassified in enumerate(misclassified_mask) if is_misclassified])


        accuracy = 100 * correct / total
        accuracy_list.append(accuracy)  # accuracy 리스트에 추가
        print(f'Validation Accuracy after {epoch+1} epochs: {accuracy:.2f}%')
```

```
Epoch 47, Batch 500, Loss: 0.031
Epoch 47, Batch 1000, Loss: 0.012
Epoch 47, Batch 1500, Loss: 0.020
Epoch 47, Batch 2000, Loss: 0.020
Validation Accuracy after 47 epochs: 71.41%
Epoch 48, Batch 500, Loss: 0.015
Epoch 48, Batch 1000, Loss: 0.018
Epoch 48, Batch 1500, Loss: 0.013
Epoch 48, Batch 2000, Loss: 0.016
Validation Accuracy after 48 epochs: 71.39%
Epoch 49, Batch 500, Loss: 0.028
Epoch 49, Batch 1000, Loss: 0.013
Epoch 49, Batch 1500, Loss: 0.021
Epoch 49, Batch 2000, Loss: 0.015
Validation Accuracy after 49 epochs: 70.82%
Epoch 50, Batch 500, Loss: 0.011
Epoch 50, Batch 1000, Loss: 0.011
Epoch 50, Batch 1500, Loss: 0.019
Epoch 50, Batch 2000, Loss: 0.024
Validation Accuracy after 50 epochs: 70.39%
Continued Training Finished
```

Mission2

2-1. Mission 1 결과 분석

분류 정확도 상위 7개 음식

1. 미역국 - 89.80%
2. 육개장 - 88.52%
3. 알밥 - 87.10%
4. 잡곡밥 - 86.41%
5. 꿀떡 - 85.58%
6. 시래기국 - 85.19%
7. 계란국 - 83.51%

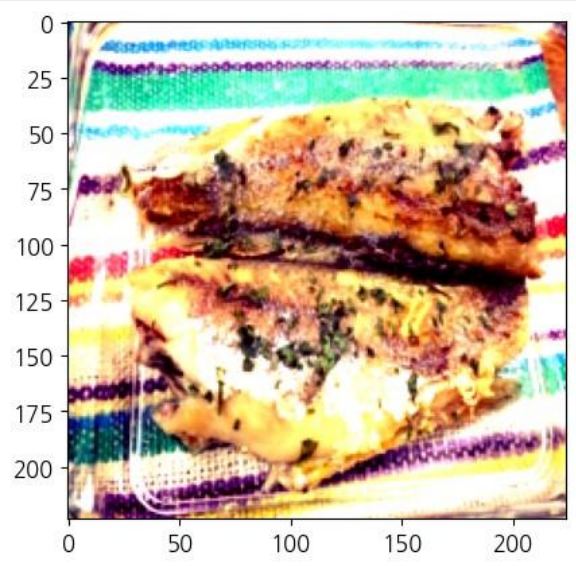
분류 정확도 하위 7개 음식

1. 장어구이 - 48.89%
2. 갈비구이 - 50.59%
3. 생선전 - 51.33%
4. 고등어구이 - 51.61%
5. 감자전 - 54.55%
6. 더덕구이 - 54.74%
7. 주먹밥 - 55.75%

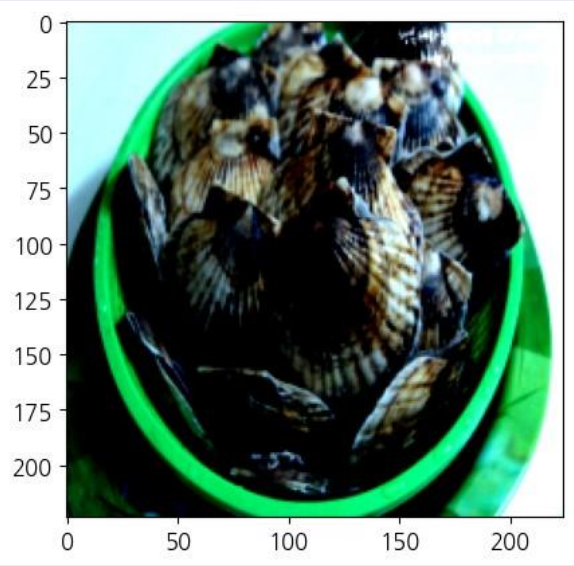
Mission2

2-1. Mission 1 결과 분석

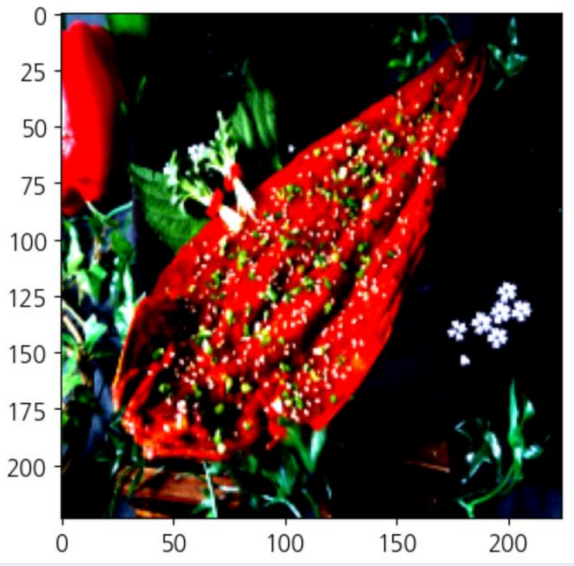
1. '구이'류 음식, 비교적 분류 정확도가 낮은 편에 속함



True label: 조기구이
Predicted label: 장어구이



True label: 조개구이
Predicted label: 갈치구이

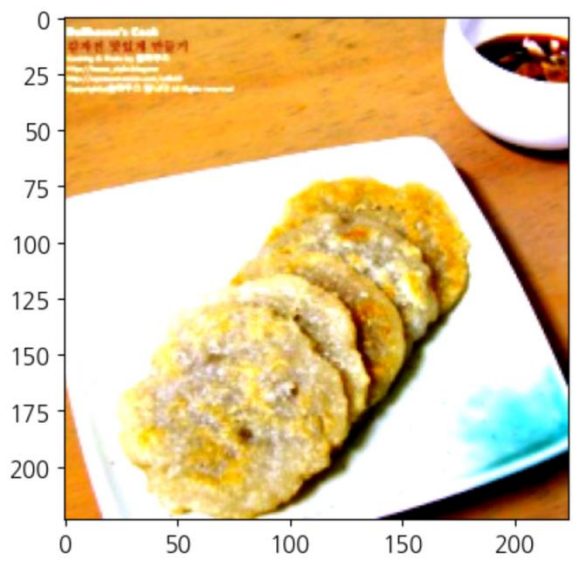


True label: 황태구이
Predicted label: 장어구이

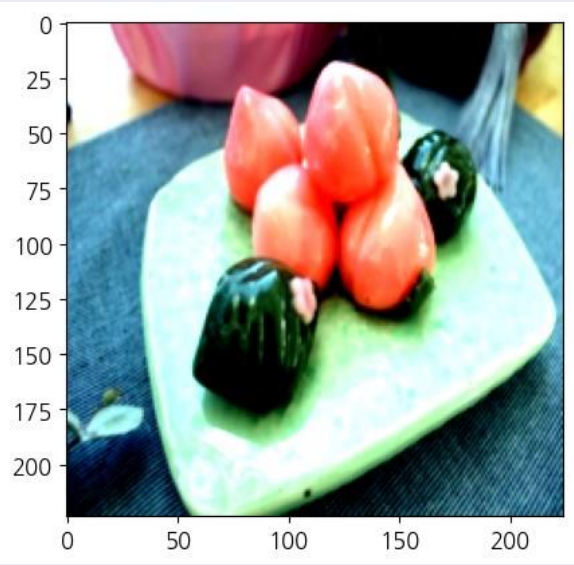
Mission2

2-1. Mission 1 결과 분석

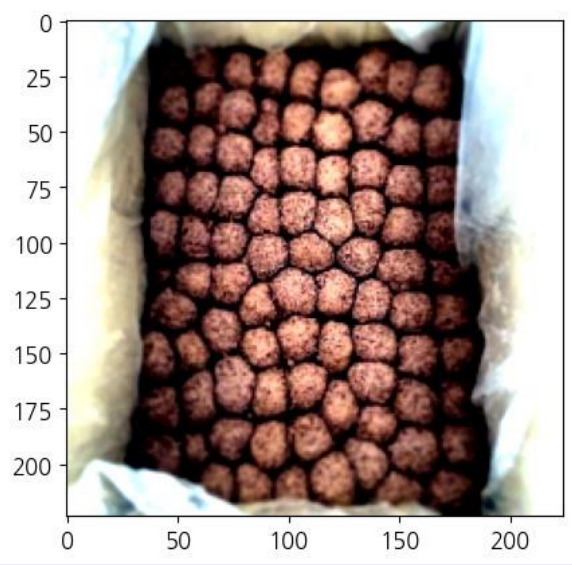
2. 음식 형태(모양)가 비슷한 경우, 오인하는 경향 있음



True label: 감자전
Predicted label: 동그랑땡



True label: 송편
Predicted label: 꿀떡

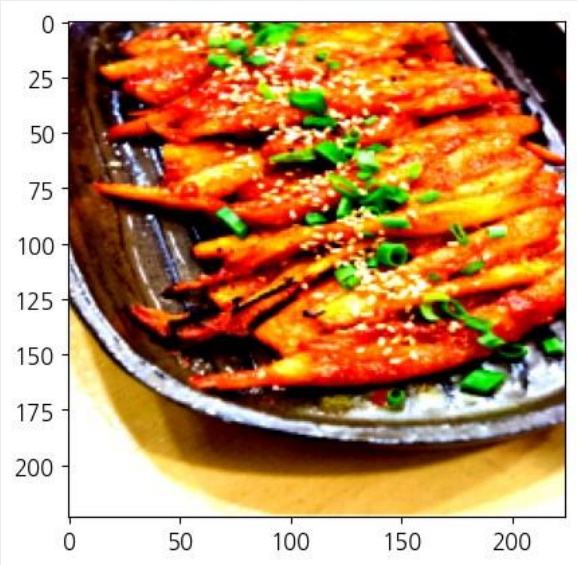


True label: 경단
Predicted label: 꿀떡

Mission2

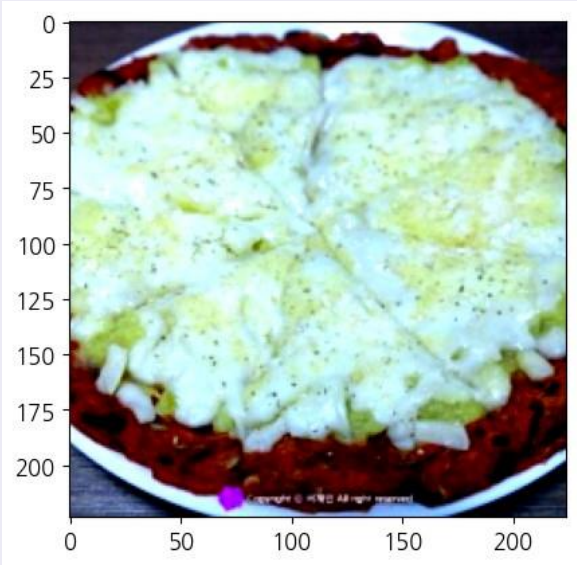
2-1. Mission 1 결과 분석

3. 음식 색상이 비슷한 경우, 오인하는 경향 있음



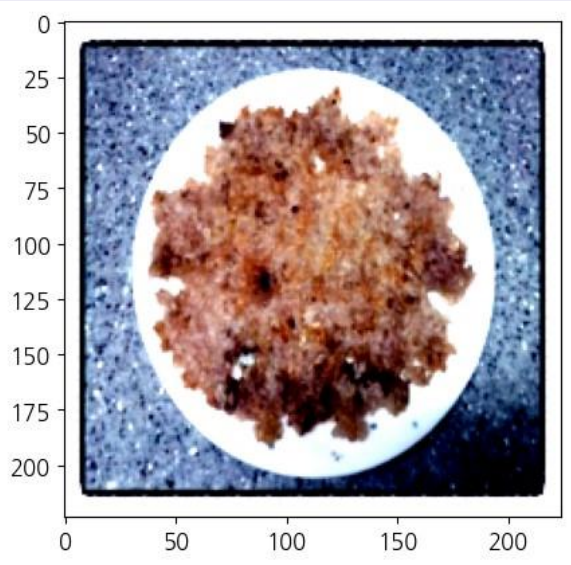
(Red)

True label: 더덕구이
Predicted label: 김치전



(Yellow)

True label: (치즈가 올라간)김치전
Predicted label: 감자전



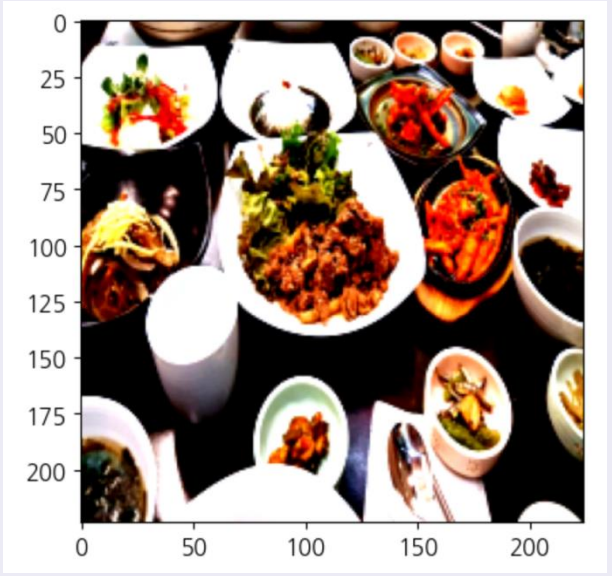
(Brown)

True label: (노릇노릇 탄)누룽지
Predicted label: 떡갈비

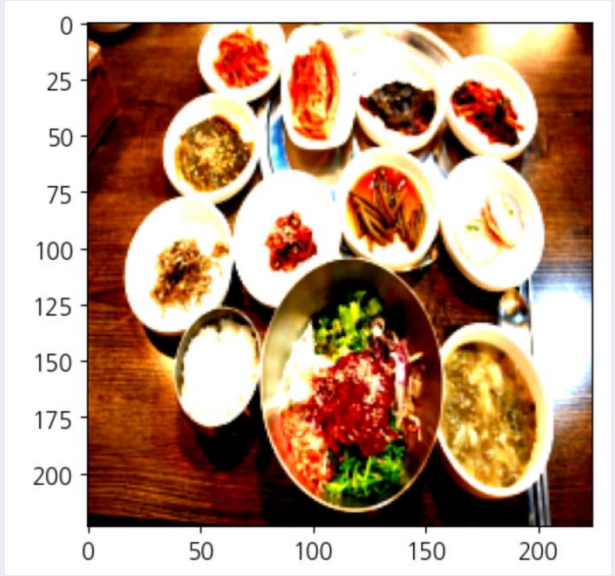
Mission2

2-1. Mission 1 결과 분석

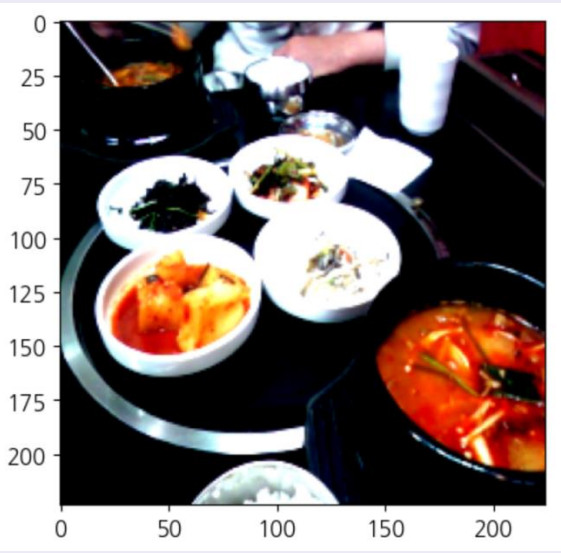
4. 다양한 음식이 동시에 등장하는 경우, 오인하는 경향 있음



True label: 더덕구이
Predicted label: 불고기



True label: 비빔밥
Predicted label: 북엇국

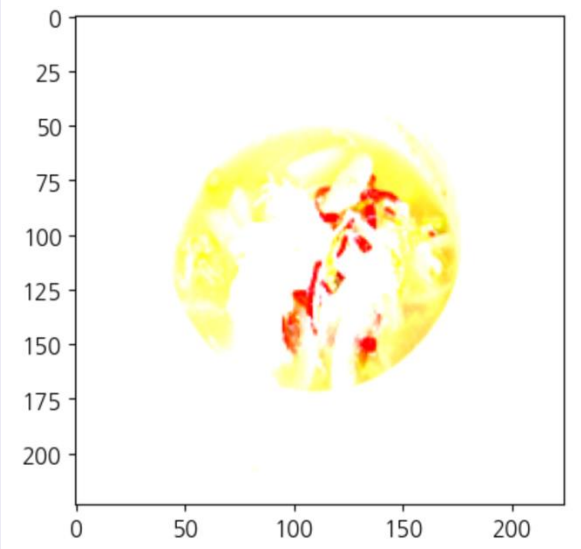


True label: 육개장
Predicted label: 조개구이

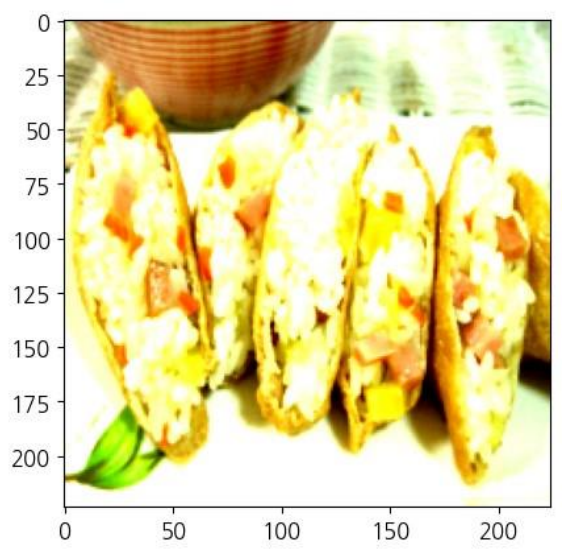
Mission2

2-1. Mission 1 결과 분석

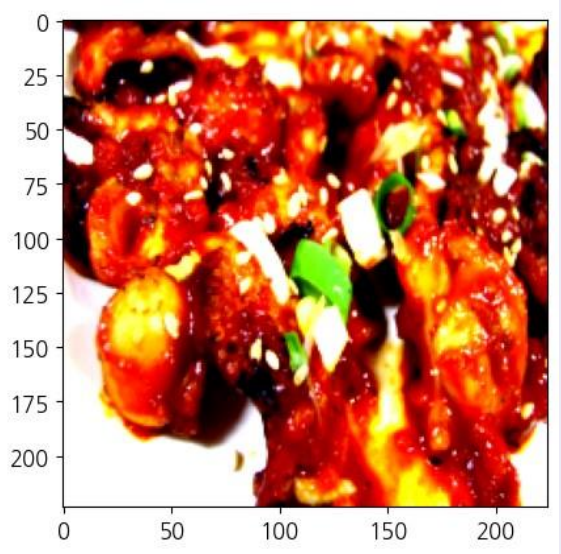
5. 오인한 경우, 대개 색조와 명도 등 음식 고유 색상이 소실됨을 확인



True label: 계란국
Predicted label: 콩나물국



True label: 유부초밥
Predicted label: 새우볶음밥



True label: 장어구이
Predicted label: 황태구이

Mission2

2-2. ① 2-1의 분석을 기반으로 성능 향상을 위한 작업 수행 (Augmentation 적용)

```
# 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 (줌인 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))
])
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)
```

Point 1)
이미지 전처리 과정 중 **밝기, 대비, 채도 등 조절**

기대효과)
데이터의 **일관성을 높여** 시각적 특징을 파악하고
더 나아가, **노이즈 감소**를 통해 성능을 향상시키고자 함

Mission2

2-2. ② 2-1의 분석을 기반으로 성능 향상을 위한 작업 수행 (Batch Size, ResNet 수정)

```
# 데이터셋 경로 설정
data_root = "/content/drive/MyDrive"
train_dir = os.path.join(data_root, "train")
val_dir = os.path.join(data_root, "val")
# 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)) #imageNet mean/std사용
    ]) #imageNet mean/std사용

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

# 데이터로더 설정
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
```

Point 1)
데이터로더 설정 시,
Batch Size를 16(Mission 1) -> **32(Mission 2)로 수정**

기대효과)
기존보다 더 많은 데이터를 한 번에 처리하여
학습 시간을 단축하고자 함

Mission2

2-2. ② 2-1의 분석을 기반으로 성능 향상을 위한 작업 수행 (Batch Size, ResNet 수정)

```python
# 모델 정의 (ResNet-50)
resnet50 = models.resnet50(pretrained=False)

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

resnet50.apply(reset_parameters)

# 마지막 Fully Connected Layer 변경
num_classes = 512  # 클래스 수
resnet50.fc = nn.Linear(2048, num_classes)

# 모델을 GPU로 이동
device = 'cuda' if torch.cuda.is_available() else 'cpu'
resnet50.to(device)
```

Point 2)
ResNet모델을 18(Mission 1) -> **50(Mission 2)으로 수정**

기대효과)
기존보다 **더 깊은 신경망을 통해** 복잡한 특징을 추출하고 더욱 **다양한 정보를 학습**시키고자 함

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Mission2

2-3. ①,② 의 성능 경향 파악

```
Epoch 47, Batch 500, Loss: 0.031
Epoch 47, Batch 1000, Loss: 0.012
Epoch 47, Batch 1500, Loss: 0.020
Epoch 47, Batch 2000, Loss: 0.020
Validation Accuracy after 47 epochs: 71.41%
Epoch 48, Batch 500, Loss: 0.015
Epoch 48, Batch 1000, Loss: 0.018
Epoch 48, Batch 1500, Loss: 0.013
Epoch 48, Batch 2000, Loss: 0.016
Validation Accuracy after 48 epochs: 71.39%
Epoch 49, Batch 500, Loss: 0.028
Epoch 49, Batch 1000, Loss: 0.013
Epoch 49, Batch 1500, Loss: 0.021
Epoch 49, Batch 2000, Loss: 0.015
Validation Accuracy after 49 epochs: 70.82%
Epoch 50, Batch 500, Loss: 0.011
Epoch 50, Batch 1000, Loss: 0.011
Epoch 50, Batch 1500, Loss: 0.019
Epoch 50, Batch 2000, Loss: 0.024
Validation Accuracy after 50 epochs: 70.39%
Continued Training Finished
```

```
Epoch 46, Batch 500, Loss: 0.066
Epoch 46, Batch 1000, Loss: 0.075
Validation Accuracy after 46 epochs: 64.36%
Epoch 47, Batch 500, Loss: 0.082
Epoch 47, Batch 1000, Loss: 0.113
Validation Accuracy after 47 epochs: 63.27%
Epoch 48, Batch 500, Loss: 0.075
Epoch 48, Batch 1000, Loss: 0.073
Validation Accuracy after 48 epochs: 64.15%
Epoch 49, Batch 500, Loss: 0.083
Epoch 49, Batch 1000, Loss: 0.075
Validation Accuracy after 49 epochs: 64.84%
Epoch 50, Batch 500, Loss: 0.082
Epoch 50, Batch 1000, Loss: 0.112
Validation Accuracy after 50 epochs: 63.72%
Continued Training Finished
```

Mission 2,
Validation Accuracy after 48 epochs : 약 43%
(*세션 연결 끊김 문제 발생)

Q. 기대와 달리 정확도가 떨어진 이유는 무엇일까?

Mission 1	Mission 2 ①	Mission 2 ②
-Batch Size = 16	-Batch Size = 32	-Batch Size = 32
-ResNet18	-ResNet18	-ResNet50
-Epoch 50	-Epoch 50	-Epoch 50
	-Augmentation	

A. 하이퍼파라미터 조정에 있어,
최적의 조합을 찾지 못 한 것이
정확도에 영향을 미친 것으로 파악된다.

Mission3

3-1. 건강관리를 위한 음식 이미지 데이터셋과 데이터로더 준비

```
# 데이터셋 경로 설정 (kfood_health)
data_root = "/content/drive/MyDrive"
train_dir = os.path.join(data_root, "kfood_health_train")
val_dir = os.path.join(data_root, "kfood_health_val")
```

```
# 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)) #imageNet mean/std사용
    ]) #imageNet mean/std사용

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

# 데이터로더 설정
batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
```

Mission3

3-1. 건강관리를 위한 음식 이미지 데이터 각 클래스별 하나씩 시각화

```
menu_folders = os.listdir(train_dir)
menu_folders = natsort.natsorted(menu_folders)
plt.figure(figsize=(15, 15))

for i, trainfolder in enumerate(menu_folders):
    trainmenu_path = os.path.join(train_dir, trainfolder)
    imgfiles = os.listdir(trainmenu_path)

    for j, imgfile in enumerate(imgfiles[:1]):  # Take 1 image from each folder  # 각 폴더에서 이미지 선택
        imgpath = os.path.join(trainmenu_path, imgfile)
        img = mpimg.imread(imgpath)

        name = unicodedata.normalize('NFC', trainfolder)  # Normalize the folder name
        plt.subplot(7, 7, i + 1)
        plt.imshow(img)                                   # 클래스 (폴더명) 정규화 (자모음 분리 방지)
        plt.title(name)
        plt.axis('off')

plt.show()
```

Mission3

3-1. 건강관리를 위한 음식 이미지 데이터 각 클래스별 하나씩 시각화

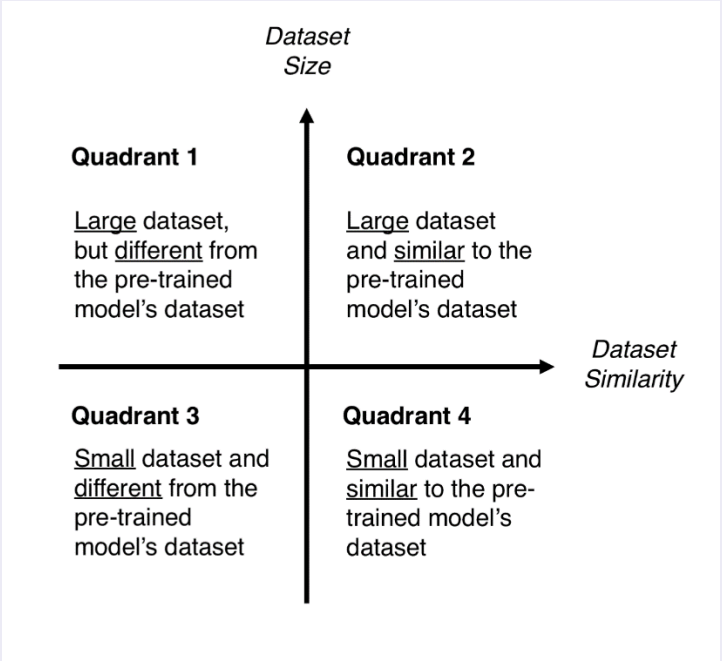
Mission3

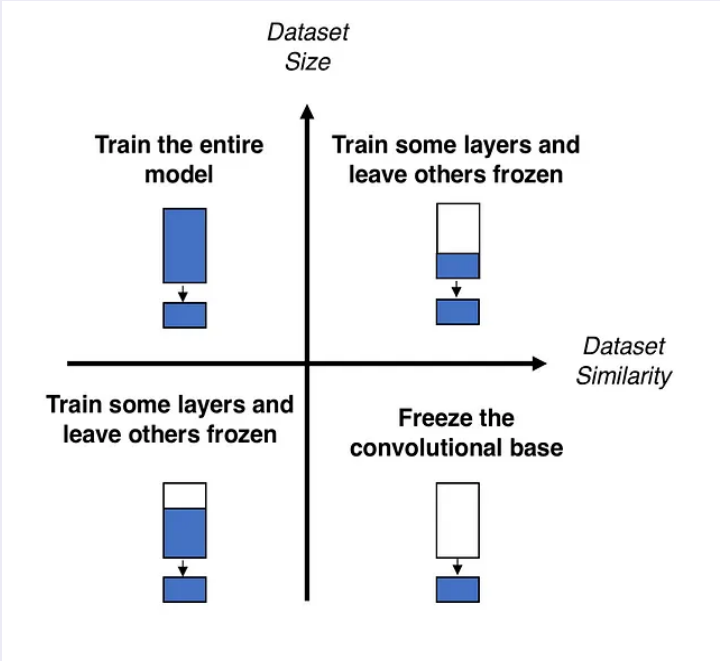
3-1. Mission2에서 학습시킨 모델과 동일한 모델 정의

```
# 모델 정의 (ResNet-18)
resnet18 = models.resnet18(pretrained=False).to(device)

# 마지막 Fully Connected Layer 변경
num_classes = 42  # 클래스 수
resnet18.fc = nn.Linear(resnet18.fc.in_features, num_classes)

# 모델을 GPU로 이동
device = 'cuda' if torch.cuda.is_available() else 'cpu'
resnet18.to(device)
```

Mission3

3-1. Task에 따른 Transfer learning 전략



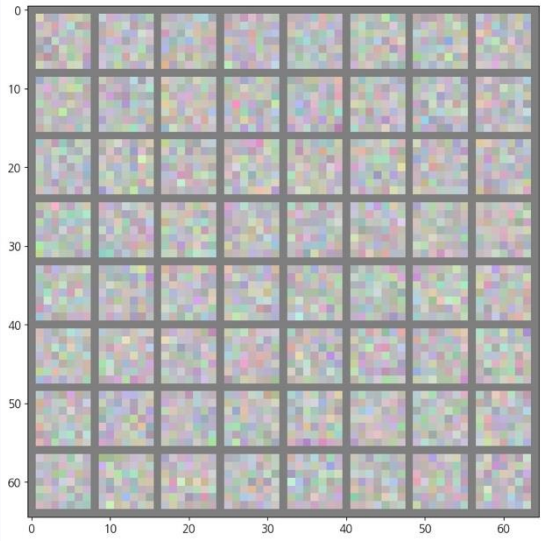


건강관리를 위한 음식 데이터는 Mission2에서 진행된 데이터와 **유사성이 높고** 데이터 수는 **크지 않다**.
따라서 Classifier의 FC Layer만 새로 정의하는 Linear probing을 사용하여 모델을 만들었다.

Mission3

3-1. 학습이 안된 모델의 첫번째 layer의 filter 확인

```
for w in resnet18.parameters():
    w = w.data.cpu()
    print(w.shape)
    break

# 가중치 renormalization
min_w = torch.min(w)
w1 = (-1/(2 * min_w)) * w + 0.5

# make grid to display it
grid_size = len(w1)
x_grid = [w1[i] for i in range(grid_size)]
x_grid = torchvision.utils.make_grid(x_grid, nrow=8, padding=1)

plt.figure(figsize=(10, 10))
imshow(x_grid)
```

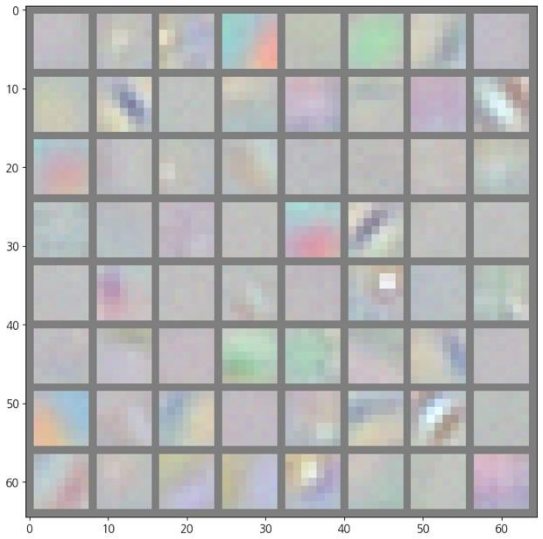


초기화된 모델의 첫번째 layer의 filter

Mission3

3-1. 사전 학습된 모델의 첫번째 layer의 filter 확인

```
for w in resnet18.parameters():
    w = w.data.cpu()
    print(w.shape)
    break

# 가중치 renormalization
min_w = torch.min(w)
w1 = (-1/(2 * min_w)) * w + 0.5

# make grid to display it
grid_size = len(w1)
x_grid = [w1[i] for i in range(grid_size)]
x_grid = torchvision.utils.make_grid(x_grid, nrow=8, padding=1)

plt.figure(figsize=(10, 10))
imshow(x_grid)
```



학습된 모델의 첫번째 layer의 filter

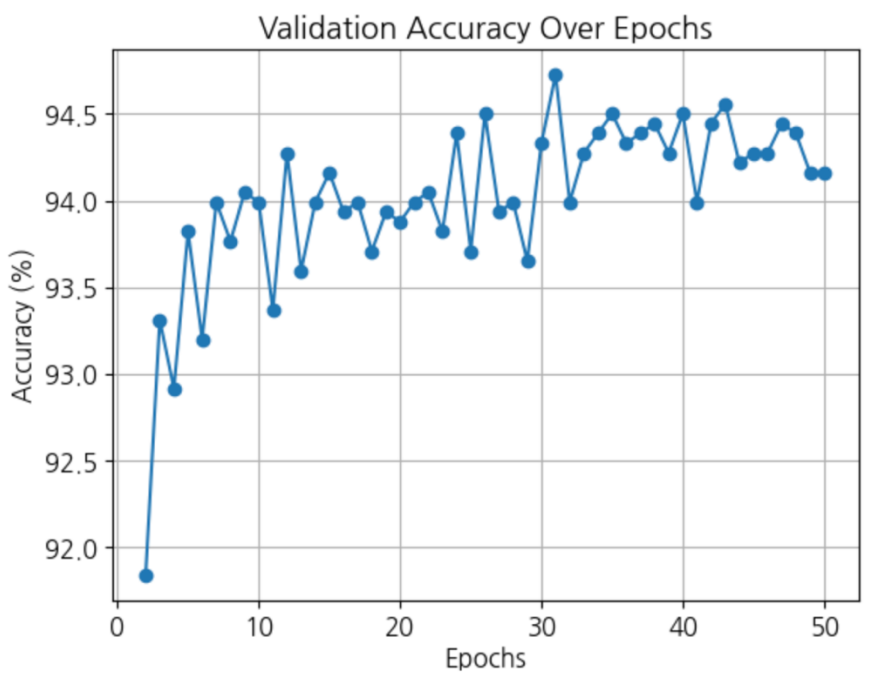
Mission3

3-1. 학습된 모델의 파라미터 적용 및 linear probing 진행

```
model_load_path='/content/drive/MyDrive/model_path/mission1.pt'
checkpoint = torch.load(model_load_path)

# 모델정의 : 사전에 학습된 파라마터 적용
resnet18.load_state_dict(checkpoint['model_state_dict'])

# 모델 아키텍처 수정 (linear probing)
num_classes = 13  # 새로운 데이터셋의 클래스 수
resnet18.fc = nn.Linear(512, num_classes)

# 모델을 GPU로 이동
device = 'cuda' if torch.cuda.is_available() else 'cpu'
resnet18.to(device)

# Loss 및 Optimizer 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(resnet18.parameters(), lr=0.01, momentum=0.9)
```

Mission3

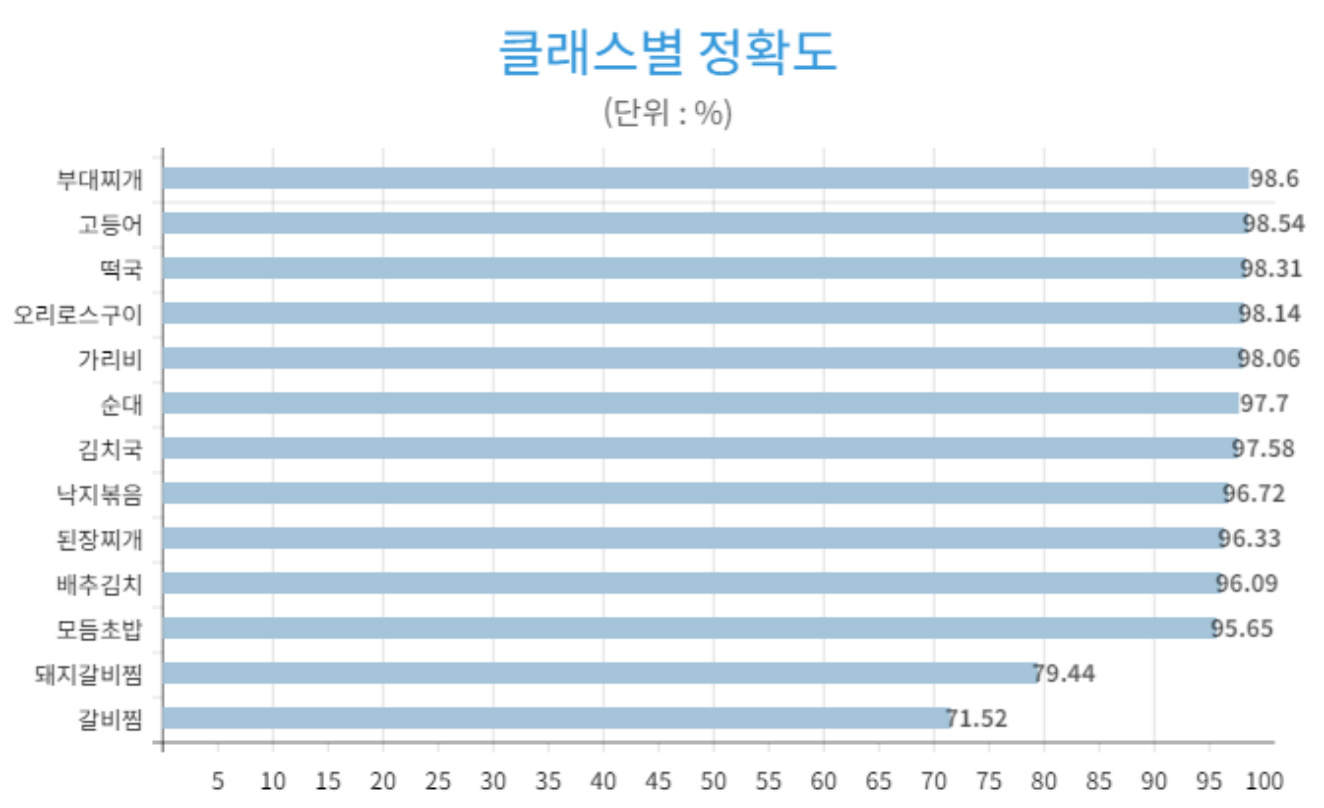
3-2. epoch별 정확도 시각화

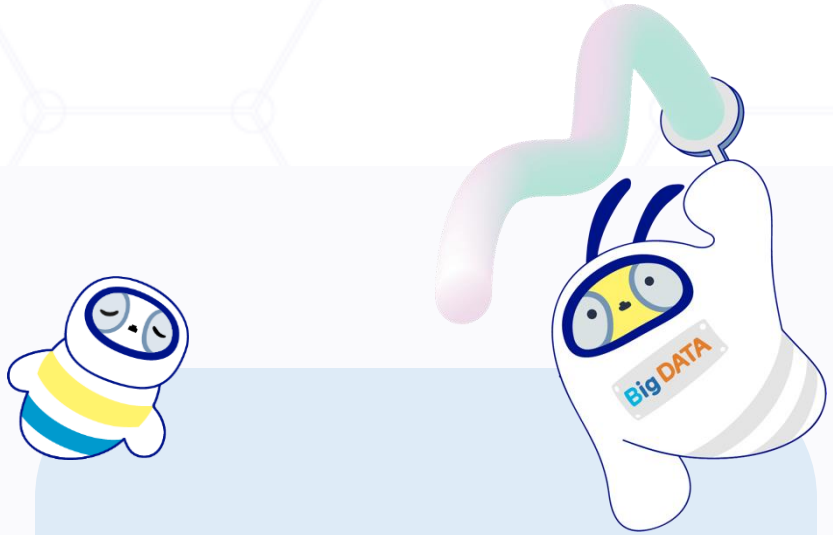


Resnet 18-50epochs에서의 accuracy 시각화 - **상승과 하락을 반복**하며 **93~4%에서 수렴**

결과 분석

모델 평가 및 결과 분석





Validation Accuracy : 94.16%

13개 중 **11개 음식 분류 정확도 95% 이상**

모든 클래스의 분류 정확도 70% 이상

잘한 점

- 다양한 분석 및 시각화 (resnet18, 50) 방식 도전

개선할 점

- 분석의 정확도 향상을 위한 하이퍼파라미터의 조정

아쉬운 점

- Google Colab Pro의 컴퓨팅 제한 이슈로 인해 진행이 어려웠음
- 성능향상을 위한 (resnet18, 50 등) 방식을 시도하였으나 기대한 결과에서 어긋남

기대효과

- 해당 모델을 바탕으로 영양 정보 제공 / 식단 추천 서비스 등 제공
- 음식 사진 분석 및 영양 성분 예측으로 사용자들의 건강한 식습관 형성에 기여

감사합니다.

대학부 DASH

팀장 박소영 김연수 유건영 이승우

2023 데이터 크리에이터 캠프
DATA CREATOR CAMP