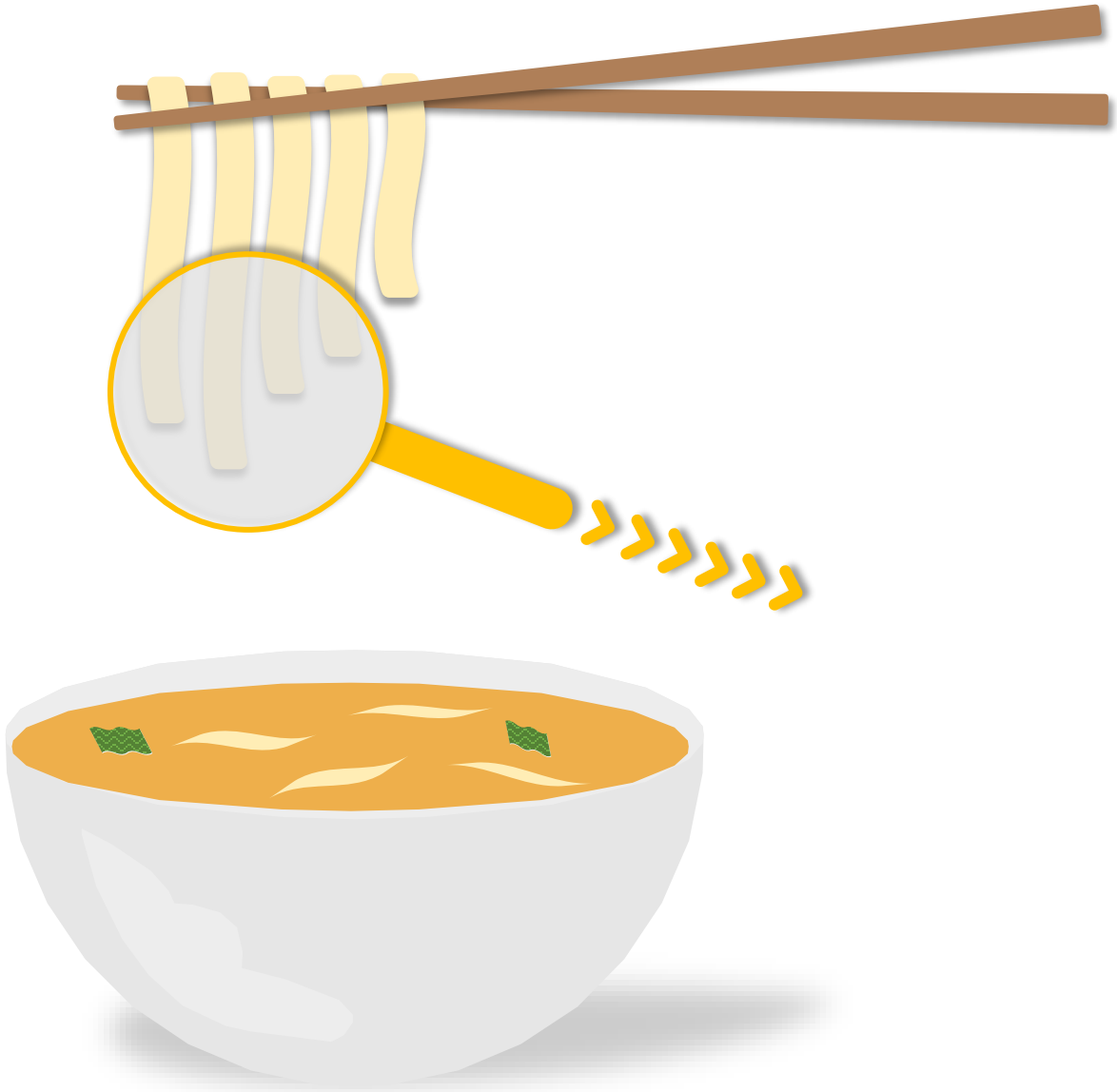
2021-1 YCS1003 Project B

KOREAN RAMEN Classification

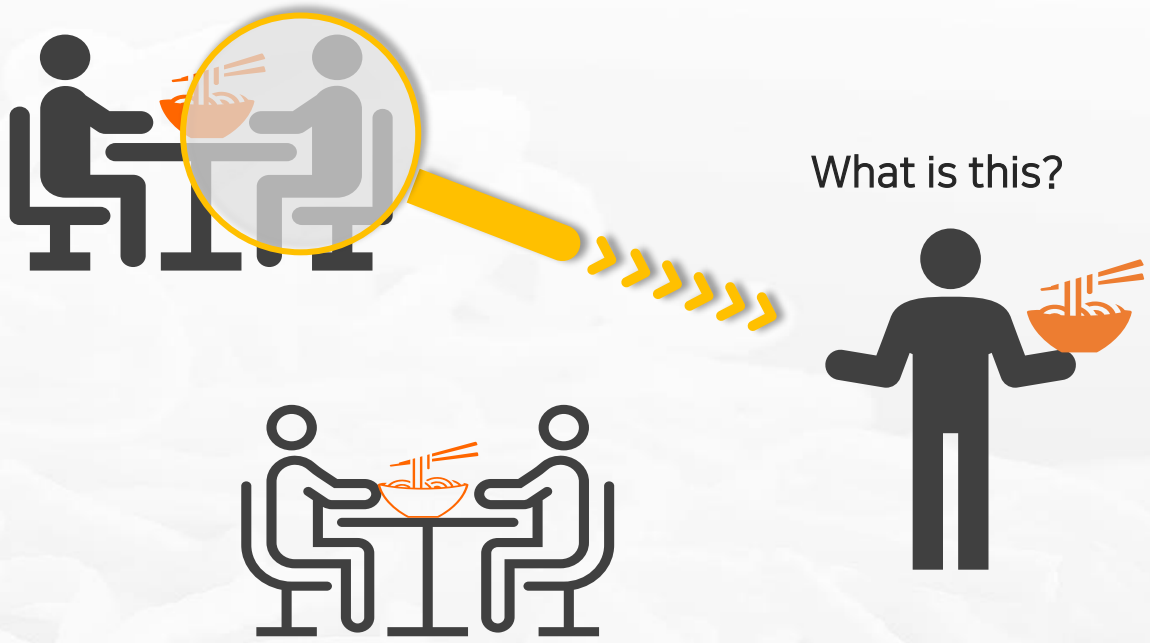
2019272060 Im Hyeon Jeong



CONTENTS

01 문제 선정 이유

1. 문제 선정 이유



02 데이터 수집 및 분류

1. 데이터 수집 방법
2. 후보군 선정
3. 분류 기준 및 순서
4. Dataset 분리 (Train/Test)

2.1 데이터 수집 및 분류

(1) 데이터 수집 방법

- 구글, 네이버, 다음 홈페이지 이미지 크롤링을 통한 대량의 데이터 1차 수집
- 표지가 아닌 면발이 나온 데이터 선별 (데이터 수집의 어려움 발생)
- 라면별 특징이 잘 드러나는 데이터 2차 수집

(2) 후보군 선정

2020 대한민국 라면 판매순위 점유율 참조

신라면, 진라면, 안성탕면, 참깨라면, 너구리, 육개장 + 홍면

꼬꼬면, 나가사끼짬뽕, 사리곰탕, 치즈볶이, 튀김우동 + 백면

= 총 13개 라면에 대한 분류 진행

(3) 분류 기준 및 순서

전체 분류 -> 색깔별 (Red/White) 분류 -> 전체 재분류

2.2 Data set (Train set / Test set)

Overall_Ramen_dataset

- anseong-tangmyeon
- baeg-myeon
- chamkkae-ramen
- cheese-bokki
- hong-myeon
- Jin_ramen
- kkokko-myeon
- nagasakki-jjamppong
- neoguli
- sali-gomtang
- shin-ramen
- twigim-udong
- yukgaejang

Red_Ramem_dataset

- anseong-tangmyeon
- chamkkae-ramen
- hong-myeon
- Jin_ramen
- neoguli
- shin-ramen
- yukgaejang

White_Ramen_dataset

- baeg-myeon
- cheese-bokki
- kkokko-myeon
- nagasakki-jjamppong
- sali-gomtang
- twigim-udong

Overall_Ramen_dataset

- anseong-tangmyeon
- baeg-myeon
- chamkkae-ramen
- cheese-bokki
- hong-myeon
- Jin_ramen
- kkokko-myeon
- nagasakki-jjamppong
- neoguli
- sali-gomtang
- shin-ramen
- twigim-udong
- yukgaejang

03 모델 설계 및 훈련

1. 모델의 전체 구조 및 목표
2. Overall Ramen Classification
3. Red Ramen Classification
4. White Ramen Classification
5. Overall Ramen Classification (2)

3.1 모델의 전체 구조 및 목표

```
# ImageFolder를 사용하여 폴더 안에 있는 이미지들을 데이터셋으로 사용

train_dataset = datasets.ImageFolder( '/content/drive/MyDrive/Overall_Ramen_dataset/train', train_transforms)
test_dataset = datasets.ImageFolder( '/content/drive/MyDrive/Overall_Ramen_dataset/test', test_transforms)

train_loader = DataLoader( train_dataset, batch_size=batch_size, shuffle=True )
test_loader = DataLoader( test_dataset, batch_size=batch_size, shuffle=True )

class_names = train_dataset.classes
print(class_names)
```

```
['Jin_ramen', 'anseong-tangmyeon', 'baeg-myeon', 'chamkkae-ramen', 'cheese-bokki', 'hong-myeon', 'kkokko-myeon',
```

train set : test set = 3 : 1 비율로 진행

- 사용할 수 있는 data가 적은 만큼 겹치게 되면 훈련의 의미가 사라지기에 주의하여 확인
- 'visualize_model' 의 결과 확인 시 test set을 통해 Answer 파악 (눈으로 구분 X)

```
# 8.9 Finetuning
model_ft = models.resnet18(pretrained=True)
model_ft.fc = nn.Linear(512, num_classes)

model_ft = model_ft.to(device)
loss_func = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
lr_scheduler_ft = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.2)
```

Pretrained resnet18 모델을 기반으로 진행

+ 최종적인 dataset에 대하여 여러 pretrained models의 finetuning 적용

```
# 8.3 하이퍼파라미터

num_epochs = 50
num_classes = 13    # 7, 6
batch_size = 4
learning_rate = 0.001
```

적은 양의 data 에 대한 정확도를 높일 수 있는 방안 탐구

⇒ data가 적어 분류 시간은 비교적 오래 걸리지 않을 것으로 예상되었기에 시간에 따른 효율성보다는 < 분류 자체의 **정확도**를 높이는 것 목표

3.1 모델의 전체 구조

```
# ImageFolder를 사용하여 폴더 안에 있는 이미지들을 데이터셋으로 사용

train_dataset = datasets.ImageFolder( '/content/drive/MyDrive/Overall_Ramen_dataset/train', train_transforms)
test_dataset = datasets.ImageFolder( '/content/drive/MyDrive/Overall_Ramen_dataset/test', test_transforms)

train_loader = DataLoader( train_dataset, batch_size=batch_size, shuffle=True )
test_loader = DataLoader( test_dataset, batch_size=batch_size, shuffle=True )

class_names = train_dataset.classes
print(class_names)
```

```
['Jin_ramen', 'anseong-tangmyeon', 'baeg-myeon', 'chamkkae-ramen', 'cheese-bokki', 'hong-myeon', 'kkokko-myeon',
```

```
# 8.9 Finetuning
model_ft = models.resnet18(pretrained=True)
model_ft.fc = nn.Linear(512, num_classes)

model_ft = model_ft.to(device)
loss_func = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
lr_scheduler_ft = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.2)
```

```
# 8.11 Train

def train(epoch, num_epochs, model, optimizer, scheduler):

    # train phase
    model.train()

    # create a progress bar
    batch_loss_list = []
    progress = ProgressMonitor(length=len(train_dataset))

    for batch, target in train_loader:
        # Move the training data to the GPU
        batch, target = batch.to(device), target.to(device)

        # forward propagation
        output = model( batch )

        # calculate the loss
        loss = loss_func( output, target )

        # clear previous gradient computation
        optimizer.zero_grad()

        # backpropagate to compute gradients
        loss.backward()

        # update model weights
        optimizer.step()

        # update progress bar
        batch_loss_list.append(loss.item())
        progress.update(epoch, num_epochs, batch.shape[0], sum(batch_loss_list)/len(batch_loss_list) )

    # 스케쥴러의 스텝을 증가 - 때가 되면 학습률 변경
    if scheduler:
        scheduler.step()
```

```
# 8.12 Test

def test(model):
    # test phase
    model.eval()

    correct = 0

    # We don't need gradients for test, so wrap in
    # no_grad to save memory
    with torch.no_grad():
        for batch, target in test_loader:
            # Move the training batch to the GPU
            batch, target = batch.to(device), target.to(device)

            # forward propagation
            output = model( batch )

            # get prediction
            output = torch.argmax(output, 1)

            # accumulate correct number
            correct += (output == target).sum().item()

    # Calculate test accuracy
    acc = 100 * float(correct) / len(test_dataset)
    print( 'Test Acc: {}/{} ({:.2f}%)'.format( correct, len(test_dataset), acc ) )

    return acc
```

```
# 8.13

since = time.time()

# initialize the best weights
best_model_weights = copy.deepcopy( model_ft.state_dict() )
best_acc = 0.0

for epoch in range(num_epochs):

    # train
    train(epoch+1, num_epochs, model_ft, optimizer_ft, lr_scheduler_ft )
    # test
    acc = test(model_ft)

    # update the best weights
    if acc > best_acc:
        best_acc = acc
        best_model_weights = copy.deepcopy( model_ft.state_dict() )

# load the best weights
model_ft.load_state_dict( best_model_weights )

# summary
time_elapsed = time.time() - since
print('Training completed in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
print('Best test accuracy: {:4f}'.format(best_acc))
```

3.2 1st Overall Ramen Classification

Data set

- 대상 : Overall Ramen (Red+White)
- 면발이 잘 드러난 사진만 사용
- 각 라면별 Train set 22개 + Test set 8개 = 30개

Result

Training completed in 5m 60s

Best test accuracy 53.284672

```
Test Acc: 68/137 (49.64%)
Epoch: 34/50 Loss: 1.5253   396 / 396
Test Acc: 67/137 (48.91%)
Epoch: 35/50 Loss: 1.6509   396 / 396
Test Acc: 71/137 (51.82%)
Epoch: 36/50 Loss: 1.6476   396 / 396
Test Acc: 71/137 (51.82%)
Epoch: 37/50 Loss: 1.5872   396 / 396
Test Acc: 73/137 (53.28%)
Epoch: 38/50 Loss: 1.5933   396 / 396
Test Acc: 72/137 (52.55%)
Epoch: 39/50 Loss: 1.5294   396 / 396
Test Acc: 73/137 (53.28%)
Epoch: 40/50 Loss: 1.6114   396 / 396
Test Acc: 64/137 (46.72%)
Epoch: 41/50 Loss: 1.5929   396 / 396
Test Acc: 71/137 (51.82%)
Epoch: 42/50 Loss: 1.5823   396 / 396
Test Acc: 65/137 (47.45%)
Epoch: 43/50 Loss: 1.6256   396 / 396
Test Acc: 69/137 (50.36%)
Epoch: 44/50 Loss: 1.6114   396 / 396
Test Acc: 65/137 (47.45%)
Epoch: 45/50 Loss: 1.5974   396 / 396
Test Acc: 71/137 (51.82%)
Epoch: 46/50 Loss: 1.5733   396 / 396
Test Acc: 70/137 (51.09%)
Epoch: 47/50 Loss: 1.5732   396 / 396
Test Acc: 67/137 (48.91%)
Epoch: 48/50 Loss: 1.5945   396 / 396
Test Acc: 65/137 (47.45%)
Epoch: 49/50 Loss: 1.5292   396 / 396
Test Acc: 71/137 (51.82%)
Epoch: 50/50 Loss: 1.6737   396 / 396
Test Acc: 68/137 (49.64%)
Training completed in 5m 60s
Best test accuracy: 53.284672
```

PRE	신라면	홍면	사리곰탕	참깨라면	사리곰탕	참깨라면	신라면
ANS	진라면	참깨라면	사리곰탕	참깨라면	꼬꼬면	참깨라면	홍면

Analysis

문제점 : 색이 확연히 차이 나는 라면은 분류 O But 색이 비슷할 경우 분류 거의 X

-> 색이 비슷한 라면들도 잘 분류할 수 있도록 2차 훈련 진행 계획

-> Dataset 분리 (Red Ramen/White Ramen)

3.3.1 Red Ramen Classification

Data set

- 대상 : Red Ramen
- 기존의 Dataset (Train / Test) 유지

Result

Training completed in 0m 55s

Best test accuracy 57.142957

```
Epoch: 36/50 Loss: 1.2076  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 37/50 Loss: 1.3880  105 / 105
Test Acc: 18/35 (51.43%)
Epoch: 38/50 Loss: 1.3667  105 / 105
Test Acc: 17/35 (48.57%)
Epoch: 39/50 Loss: 1.3633  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 40/50 Loss: 1.2881  105 / 105
Test Acc: 16/35 (45.71%)
Epoch: 41/50 Loss: 1.2818  105 / 105
Test Acc: 20/35 (57.14%)
Epoch: 42/50 Loss: 1.3629  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 43/50 Loss: 1.2531  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 44/50 Loss: 1.4064  105 / 105
Test Acc: 20/35 (57.14%)
Epoch: 45/50 Loss: 1.2670  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 46/50 Loss: 1.2241  105 / 105
Test Acc: 18/35 (51.43%)
Epoch: 47/50 Loss: 1.3060  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 48/50 Loss: 1.2909  105 / 105
Test Acc: 18/35 (51.43%)
Epoch: 49/50 Loss: 1.2718  105 / 105
Test Acc: 19/35 (54.29%)
Epoch: 50/50 Loss: 1.4167  105 / 105
Test Acc: 19/35 (54.29%)
Training completed in 0m 55s
Best test accuracy: 57.142857
```

면발의 두께 ?

너구리 라면 : 면발 두꺼움

But 다른 라면들은 비슷한 굵기

-> 좋은 분류 기준 X

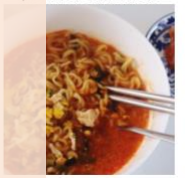
후레이크의 모양 !

참깨라면 : 계란블럭

너구리 : 다시마 + 너구리모양 후레이크

홍면 : 표고버섯, 풍부한 양의 파

-> 목표 : 후레이크가 잘 나온 data 수집



신라면	신라면	홍면	너구리
홍면	너구리	너구리	너구리

있음 (적은 data로 인한 문제)

하여 재훈련 진행 계획

3.3.2 Red Ramen Classification

Data set

- 대상 : Red Ramen
- Feature가 잘 드러나는 Train set 사용
- Test set은 변경 X

Result

Training completed in 3m 58s

Best test accuracy 67.415730

```
[14]  Epoch: 34/50 Loss: 0.9945   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 35/50 Loss: 0.9456   255 / 255
      Test Acc: 54/89 (60.67%)
      Epoch: 36/50 Loss: 0.9253   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 37/50 Loss: 1.0402   255 / 255
      Test Acc: 59/89 (66.29%)
      Epoch: 38/50 Loss: 0.9557   255 / 255
      Test Acc: 56/89 (62.92%)
      Epoch: 39/50 Loss: 1.0778   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 40/50 Loss: 0.8518   255 / 255
      Test Acc: 56/89 (62.92%)
      Epoch: 41/50 Loss: 0.9578   255 / 255
      Test Acc: 58/89 (65.17%)
      Epoch: 42/50 Loss: 0.9446   255 / 255
      Test Acc: 59/89 (66.29%)
      Epoch: 43/50 Loss: 0.9496   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 44/50 Loss: 0.9393   255 / 255
      Test Acc: 59/89 (66.29%)
      Epoch: 45/50 Loss: 1.0407   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 46/50 Loss: 0.9907   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 47/50 Loss: 0.8850   255 / 255
      Test Acc: 55/89 (61.80%)
      Epoch: 48/50 Loss: 0.9901   255 / 255
      Test Acc: 58/89 (65.17%)
      Epoch: 49/50 Loss: 0.9244   255 / 255
      Test Acc: 57/89 (64.04%)
      Epoch: 50/50 Loss: 0.9176   255 / 255
      Test Acc: 55/89 (61.80%)
      Training completed in 3m 58s
      Best test accuracy: 67.415730
```

PRE	육개장	홍면	신라면	참깨라면	홍면	너구리	안성탕면
ANS	육개장	진라면	신라면	참깨라면	홍면	너구리	안성탕면

Analysis

각 라면별 feature가 잘 드러나는 data 수집

-> train set 변경 후 재훈련 진행

-> feature가 잘 드러난 라면들은 분류 well ; Best test accuracy 약 10% 증가

(한계점 : feature가 잘 드러나지 않는 라면은 분류 잘 X)

3.4.1 White Ramen Classification

Data set

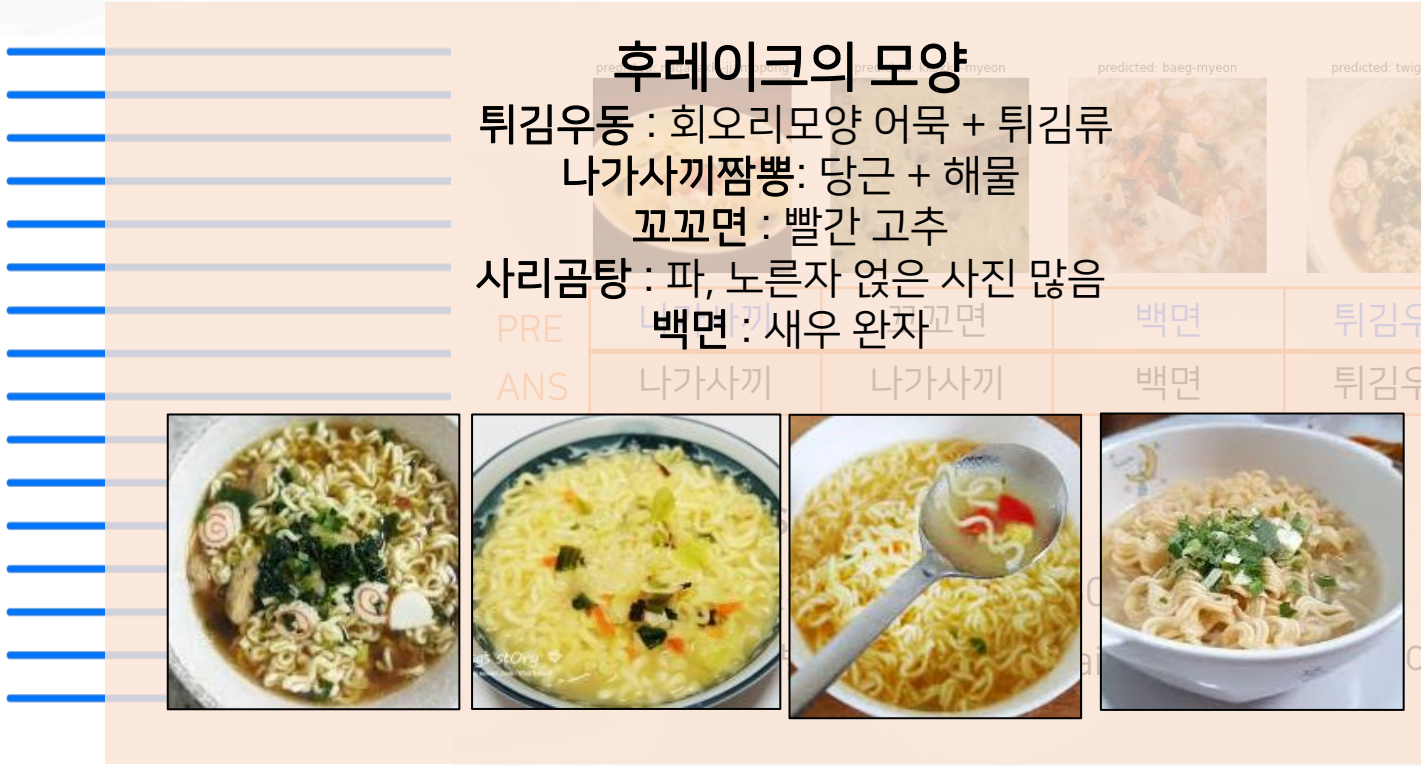
- 대상 : White Ramen
- 기존의 Dataset (Train / Test) 유지

Result

Training completed in 1m 12s

Best test accuracy 67.594666

```
Epoch: 35/50 Loss: 0.3961  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 36/50 Loss: 0.5710  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 37/50 Loss: 0.4716  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 38/50 Loss: 0.2929  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 39/50 Loss: 0.3981  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 40/50 Loss: 0.4585  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 41/50 Loss: 0.5118  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 42/50 Loss: 0.4266  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 43/50 Loss: 0.3995  141 / 141
Test Acc: 40/48 (83.33%)
Epoch: 44/50 Loss: 0.4667  141 / 141
Test Acc: 40/48 (83.33%)
Epoch: 45/50 Loss: 0.4636  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 46/50 Loss: 0.4510  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 47/50 Loss: 0.5470  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 48/50 Loss: 0.5391  141 / 141
Test Acc: 40/48 (83.33%)
Epoch: 49/50 Loss: 0.4433  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 50/50 Loss: 0.5568  141 / 141
Test Acc: 40/48 (83.33%)
Training completed in 2m 14s
Best test accuracy: 89.583333
```



후레이크의 모양

튀김우동 : 회오리모양 어묵 + 튀김류

나가사끼짬뽕: 당근 + 해물

꼬꼬면 : 빨간 고추

사리곰탕 : 파, 노른자 없은 사진 많음

백면 : 새우 완자

PRE	나가사끼	꼬꼬면	백면	튀김우동	튀김우동	나가사끼
ANS	나가사끼	나가사끼	백면	튀김우동	사리곰탕	꼬꼬면

훈련 진행 계획

3.4.2 White Ramen Classification

Data set

- 대상 : White Ramen
- 기존의 Dataset (Train / Test) 유지
- Test set은 변경 X

Result

Training completed in 2m 14s

Best test accuracy 89.583333

```
Epoch: 35/50 Loss: 0.3961  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 36/50 Loss: 0.5710  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 37/50 Loss: 0.4716  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 38/50 Loss: 0.2929  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 39/50 Loss: 0.3981  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 40/50 Loss: 0.4585  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 41/50 Loss: 0.5118  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 42/50 Loss: 0.4266  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 43/50 Loss: 0.3995  141 / 141
Test Acc: 40/48 (83.33%)
Epoch: 44/50 Loss: 0.4667  141 / 141
Test Acc: 40/48 (83.33%)
Epoch: 45/50 Loss: 0.4636  141 / 141
Test Acc: 42/48 (87.50%)
Epoch: 46/50 Loss: 0.4510  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 47/50 Loss: 0.5470  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 48/50 Loss: 0.5391  141 / 141
Test Acc: 40/48 (83.33%)
Epoch: 49/50 Loss: 0.4433  141 / 141
Test Acc: 41/48 (85.42%)
Epoch: 50/50 Loss: 0.5568  141 / 141
Test Acc: 40/48 (83.33%)
Training completed in 2m 14s
Best test accuracy: 89.583333
```

PRE	사리곰탕	치즈볶이	백면	백면	나가사끼	튀김우동
ANS	꼬꼬면	치즈볶이	백면	백면	나가사끼	튀김우동

Analysis

각 라면별 feature가 잘 드러나는 data 수집 (Red Ramen에 비해 easy)

-> train set 변경 후 재훈련 진행

-> feature가 잘 드러난 라면들은 분류 well ; Best test accuracy 증가

- Red Ramen에 비해 훨씬 높은 분류 정확도를 보임 (b/c feature 명확)

3.5 2nd Overall Ramen Classification

Data set

- 대상 : Overall Ramen (Red+White)
- 면발이 잘 드러난 사진만 사용
- 라면별 feature가 잘 드러나는 train set 기반

Result

Training completed in 2m 45s

Best test accuracy 73.437500

PRE	육개장	홍면	육개장	사리곰탕	백면	육개장	홍면
ANS	육개장	홍면	육개장	사리곰탕	백면	육개장	홍면

```
Training completed in 5m 60s
Best test accuracy: 53.284672
```

```
Training completed in 2m 45s
Best test accuracy: 73.437500
```

Analysis

각 라면별 feature가 잘 드러나는 train set data로 재훈련 진행

-> 처음과 비교하여 Best test accuracy 약 20% 증가 & 소요시간 감소

=> 더 정확도를 높여 보고자 함

04 모델 튜닝 및 정확도 향상

1. Pretrained models finetuning
2. Learning rate decay
3. Data augmentation

4.0 적은 양의 data에 대한 정확도 향상

분류 초반 : 데이터가 적어 **오버피팅(Overfitting)** 발생하는 문제 발생

1. 새로운 data에 대한 예측력 ↓ → Train data의 수를 늘려 더 많은 학습 진행
2. Data의 feature가 잘 드러나도록 train data set 변경하여 재학습
3. Test accuracy가 증가하다가 더 이상 증가 X 경우 → Early stopping

더 정확성을 높여보고자 ,, 최종 dataset에 대하여 다양한 방법 시도

1. 여러 pre-trained model 적용 (resnet18, googlenet, vgg16, alexnet, squeezenet1_0)
   (fixed feature extractor도 진행해보았지만 큰 차이 X, 앙상블 학습보다 resnet18 단일 모델 정확도 ↑)
2. Learning rate decay의 gamma 값을 2배로 변경 (0.1 -> 0.2)
3. 더 많은 이미지 Data augmentation 적용

4.1 Pretrained Models Finetuning

```python
import torchvision.models as models
# resnet18 = models.resnet18(pretrained=True)
# alexnet = models.alexnet(pretrained=True)
# vgg16 = models.vgg16(pretrained=True)
# squeezenet1_0 = models.squeezenet1_0(pretrained=True)
# googlenet = models.googlenet(pretrained=True)
```

```python
# Finetuning codes
resnet18        1. model.fc = nn.Linear(512, num_classes)
alexnet         2. model.classifier[6] = nn.Linear(4096,num_classes)
vgg16           3. model.classifier[6] = nn.Linear(4096,num_classes)
squeezenet1_0   4. model.classifier[1] = nn.Conv2d(512, num_classes, kernel_size=(1,1), stride=(1,1))
googlenet       5. model.fc = nn.Linear(1024, num_classes)
```

Resnet 18

```
Epoch: 30/40 Loss: 0.9049  346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 31/40 Loss: 1.0412  346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 32/40 Loss: 0.9249  346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 33/40 Loss: 0.9895  346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 34/40 Loss: 1.0066  346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 35/40 Loss: 0.9657  346 / 346
Test Acc: 94/128 (73.44%)
Epoch: 36/40 Loss: 0.9332  346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 37/40 Loss: 0.9537  346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 38/40 Loss: 1.0524  346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 39/40 Loss: 1.0589  346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 40/40 Loss: 1.0088  346 / 346
Test Acc: 88/128 (68.75%)
Training completed in 2m 45s
Best test accuracy: 73.437500
```

Googlenet

```
Epoch: 30/40 Loss: 1.5603  346 / 346
Test Acc: 74/128 (57.81%)
Epoch: 31/40 Loss: 1.5205  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 32/40 Loss: 1.4927  346 / 346
Test Acc: 77/128 (60.16%)
Epoch: 33/40 Loss: 1.4947  346 / 346
Test Acc: 76/128 (59.38%)
Epoch: 34/40 Loss: 1.5528  346 / 346
Test Acc: 80/128 (62.50%)
Epoch: 35/40 Loss: 1.4755  346 / 346
Test Acc: 75/128 (58.59%)
Epoch: 36/40 Loss: 1.4582  346 / 346
Test Acc: 74/128 (57.81%)
Epoch: 37/40 Loss: 1.5323  346 / 346
Test Acc: 78/128 (60.94%)
Epoch: 38/40 Loss: 1.4808  346 / 346
Test Acc: 71/128 (55.47%)
Epoch: 39/40 Loss: 1.5071  346 / 346
Test Acc: 75/128 (58.59%)
Epoch: 40/40 Loss: 1.5361  346 / 346
Test Acc: 74/128 (57.81%)
Training completed in 3m 23s
Best test accuracy: 62.500000
```

Vgg16

```
Epoch: 30/40 Loss: 0.8232  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 31/40 Loss: 0.8299  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 32/40 Loss: 0.8400  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 33/40 Loss: 0.8429  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 34/40 Loss: 0.8274  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 35/40 Loss: 0.8455  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 36/40 Loss: 0.8982  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 37/40 Loss: 0.8306  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 38/40 Loss: 0.8860  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 39/40 Loss: 0.8420  346 / 346
Test Acc: 72/128 (56.25%)
Epoch: 40/40 Loss: 0.8580  346 / 346
Test Acc: 72/128 (56.25%)
Training completed in 7m 58s
Best test accuracy: 58.593750
```

Alexnet

```
Epoch: 30/40 Loss: 1.0254  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 31/40 Loss: 1.0592  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 32/40 Loss: 1.0349  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 33/40 Loss: 1.0526  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 34/40 Loss: 1.1205  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 35/40 Loss: 1.0316  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 36/40 Loss: 0.9966  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 37/40 Loss: 1.0590  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 38/40 Loss: 1.1190  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 39/40 Loss: 1.0269  346 / 346
Test Acc: 61/128 (47.66%)
Epoch: 40/40 Loss: 1.0009  346 / 346
Test Acc: 61/128 (47.66%)
Training completed in 2m 52s
Best test accuracy: 50.781250
```

Squeezenet1_0

```
Epoch: 30/40 Loss: 1.1124  346 / 346
Test Acc: 66/128 (51.56%)
Epoch: 31/40 Loss: 1.0865  346 / 346
Test Acc: 66/128 (51.56%)
Epoch: 32/40 Loss: 1.0560  346 / 346
Test Acc: 66/128 (51.56%)
Epoch: 33/40 Loss: 1.0712  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 34/40 Loss: 1.0412  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 35/40 Loss: 1.0791  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 36/40 Loss: 1.1229  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 37/40 Loss: 1.0407  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 38/40 Loss: 1.1184  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 39/40 Loss: 1.1072  346 / 346
Test Acc: 67/128 (52.34%)
Epoch: 40/40 Loss: 1.0555  346 / 346
Test Acc: 67/128 (52.34%)
Training completed in 2m 35s
Best test accuracy: 53.125000
```

4.2 Learning rate Decay

```
# 8.10  Fixed feature extractor
model_ft = models.resnet18(pretrained=True)
for param in model_ft.parameters():
  param.requires_grad = False
model_ft.fc = nn.Linear(model_ft.fc.in_features,num_classes)
model_ft = model_ft.to(device)
loss_func = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
lr_scheduler_ft = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.2) # learning rate dacay
```

Learning rate gamma 값
2배로 변경 (0.1 ->0.2)

```
Epoch: 33/40 Loss: 0.9895      346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 34/40 Loss: 1.0066      346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 35/40 Loss: 0.9657      346 / 346
Test Acc: 94/128 (73.44%)
Epoch: 36/40 Loss: 0.9332      346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 37/40 Loss: 0.9537      346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 38/40 Loss: 1.0524      346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 39/40 Loss: 1.0589      346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 40/40 Loss: 1.0088      346 / 346
Test Acc: 89/128 (69.75%)
Training completed in 2m 45s
Best test accuracy: 73.437500
```

```
Epoch: 33/40 Loss: 0.9616      346 / 346
Test Acc: 95/128 (74.22%)
Epoch: 34/40 Loss: 0.8903      346 / 346
Test Acc: 96/128 (75.00%)
Epoch: 35/40 Loss: 0.8814      346 / 346
Test Acc: 87/128 (67.97%)
Epoch: 36/40 Loss: 0.8901      346 / 346
Test Acc: 95/128 (74.22%)
Epoch: 37/40 Loss: 0.8517      346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 38/40 Loss: 0.8501      346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 39/40 Loss: 0.9213      346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 40/40 Loss: 0.8306      346 / 346
Test Acc: 96/128 (75.00%)
Training completed in 2m 44s
Best test accuracy: 75.000000
```

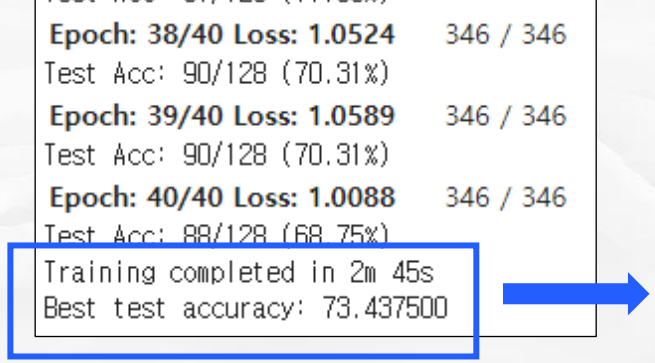
4.3 Data Augmentation

```
# 8.5

# Training을 위한 데이터 확장과 데이터셋 정규화
# 데이터의 수가 적다고 느껴 data augumentation을 다양하게 진행함

train_transforms = transforms.Compose([
        transforms.RandomResizedCrop(224),  # 랜덤 리사이즈 후 크롭
        transforms.RandomHorizontalFlip(),  # 랜덤 수평 플립
        transforms.RandomHorizontalFlip(),  # 좌우반전
        transforms.RandomRotation(degrees=10), # 20도 회전
        transforms.ColorJitter(), # 색 관련 변화
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
```

```
Epoch: 36/40 Loss: 0.9332      346 / 346
Test Acc: 92/128 (71.88%)
Epoch: 37/40 Loss: 0.9537      346 / 346
Test Acc: 91/128 (71.09%)
Epoch: 38/40 Loss: 1.0524      346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 39/40 Loss: 1.0589      346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 40/40 Loss: 1.0088      346 / 346
Test Acc: 88/128 (68.75%)
Training completed in 2m 45s
Best test accuracy: 73.437500
```

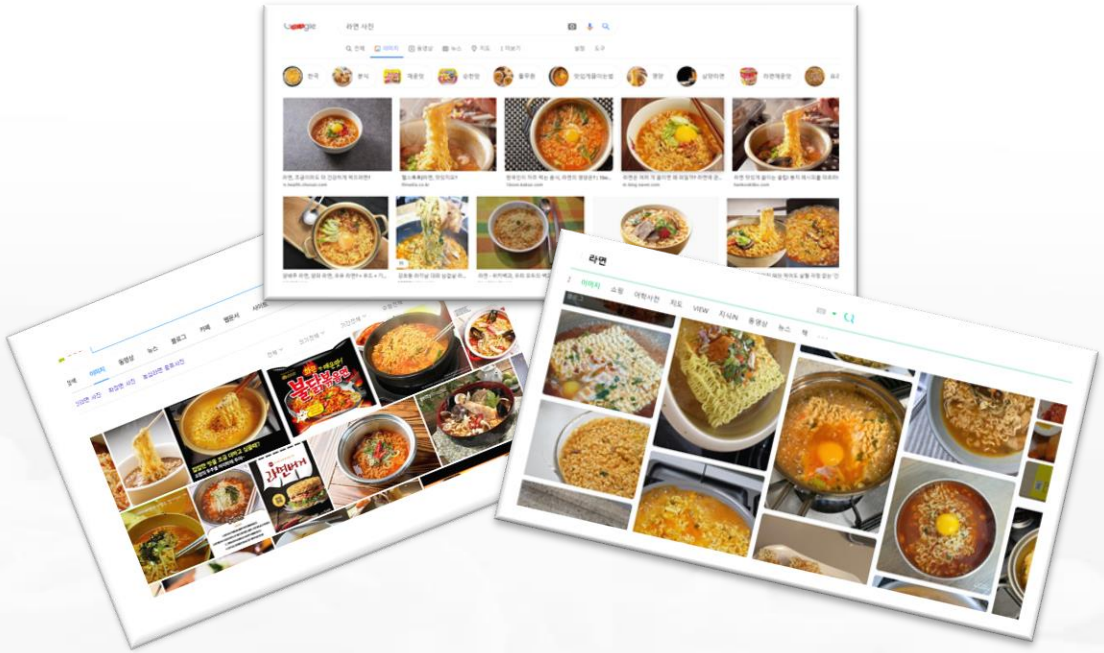


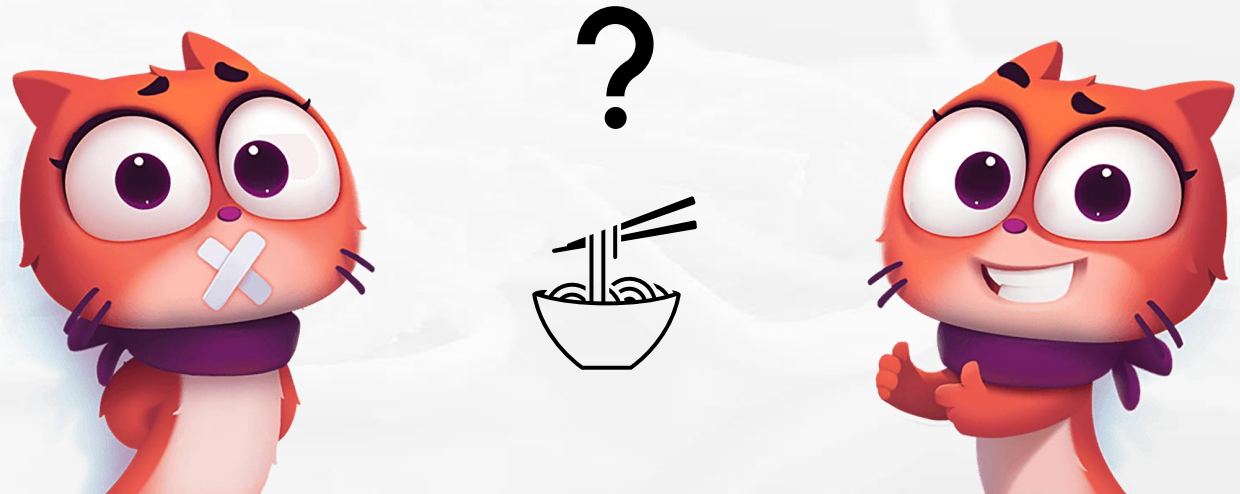
```
Epoch: 36/40 Loss: 0.8871      346 / 346
Test Acc: 98/128 (76.56%)
Epoch: 37/40 Loss: 0.8973      346 / 346
Test Acc: 90/128 (70.31%)
Epoch: 38/40 Loss: 0.9120      346 / 346
Test Acc: 93/128 (72.66%)
Epoch: 39/40 Loss: 0.8716      346 / 346
Test Acc: 94/128 (73.44%)
Epoch: 40/40 Loss: 0.8268      346 / 346
Test Acc: 98/128 (76.56%)
Training completed in 2m 44s
Best test accuracy: 76.562500
```

05 결과 시연 및 평가

1. 결과 시연
2. 결론 및 한계
3. 더 나아가고 싶은 점

5. 1 결과 시연 (현실 적합성)





5. 1 결과 시연 (현실 적합성)

5.1 결과 시연 (현실 적합성)

Data set

- 대상 : Overall Ramen
- Train set : 최종적으로 수집된 dataset (인터넷 자료)
- Test set : 직접 촬영한 dataset

Result

Training completed in 5m 49s

Best test accuracy 81.818182

```
# ImageFolder를 사용하여 폴더 안에 있는 이미지들을 데이터셋으로 사용

train_dataset = datasets.ImageFolder( '/content/drive/MyDrive/My_Ramen_dataset/train', train_transforms)
test_dataset = datasets.ImageFolder( '/content/drive/MyDrive/My_Ramen_dataset/test', test_transforms)

train_loader = DataLoader( train_dataset, batch_size=batch_size, shuffle=True )
test_loader = DataLoader( test_dataset, batch_size=batch_size, shuffle=True )

class_names = train_dataset.classes
print(class_names)
```

```
Epoch: 31/40 Loss: 0.6563   291 / 291
Test Acc: 8/11 (72.73%)
Epoch: 32/40 Loss: 0.7336   291 / 291
Test Acc: 9/11 (81.82%)
Epoch: 33/40 Loss: 0.7037   291 / 291
Test Acc: 8/11 (72.73%)
Epoch: 34/40 Loss: 0.6199   291 / 291
Test Acc: 9/11 (81.82%)
Epoch: 35/40 Loss: 0.6687   291 / 291
Test Acc: 8/11 (72.73%)
Epoch: 36/40 Loss: 0.6729   291 / 291
Test Acc: 9/11 (81.82%)
Epoch: 37/40 Loss: 0.7103   291 / 291
Test Acc: 8/11 (72.73%)
Epoch: 38/40 Loss: 0.6435   291 / 291
Test Acc: 9/11 (81.82%)
Epoch: 39/40 Loss: 0.6565   291 / 291
Test Acc: 9/11 (81.82%)
Epoch: 40/40 Loss: 0.6115   291 / 291
Test Acc: 8/11 (72.73%)
Training completed in 5m 49s
Best test accuracy: 81.818182
```

PRE	너구리	꼬꼬면	사리곰탕	홍면	백면	튀김우동	치즈볶이
ANS	너구리	꼬꼬면	사리곰탕	홍면	백면	튀김우동	치즈볶이

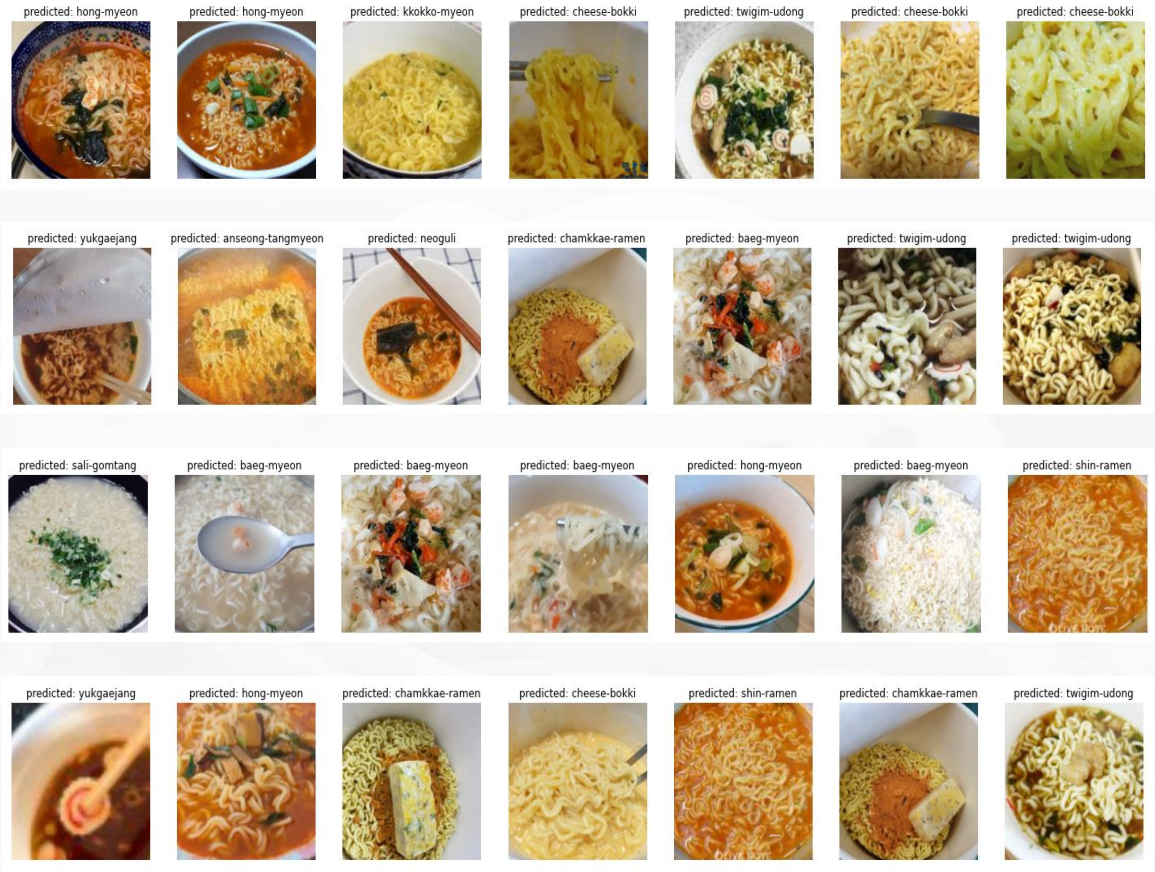
Analysis

약 81%의 높은 정확도 --> 모델의 현실 적합성 가능성 확인

한계 : Test Acc의 크기가 일정하게 나타남

→ 더 많은 양의 현실 test set 수집 계획

5. 2 결론 및 더 나아가고 싶은 점



■ 언뜻 눈으로 보면 구분하기 힘든 라면들을 인공지능을 통해 분류할 수 있었음
그러나 국물이 많아 면발이 잘 보이지 않거나 후레이크가 보이지 않은 경우
분류가 어려울 수 있다는 한계점이 있음 → 사진을 잘 찍어서 올리면 해결 가능

■ 적은 양의 data에 대한 분류 정확도를 높이는 어려움을 체험하였으며
최대한 성능을 끌어올릴 수 있는 방안을 찾기 위해 여러 방안을 시도함

■ 예측 대상이 되는 데이터가 적합한 feature를 잘 형성했는지가 성능 결정
→ 적합한 feature 형성 위한 train/test data collecting의 중요성
(Data 수집 기준을 변경하여 train data set 수정하여 재훈련)
ex) 면발 위주 (무작위) -> 후레이크 모양

■ 라면을 좋아하는 한국 사람들을 위한, 그리고 외국인들에게 한국 라면 홍보를 위한 더 많은 종류의 라면을 분류할 수 있는 모델을 만들어보고 싶음

THANK YOU