음식 사진 인식을 통한 칼로리정보와 음식추천

관리해조

조장 : 손가현

팀원 : 김정웅

팀원 : 노영재

팀원 : 송현욱

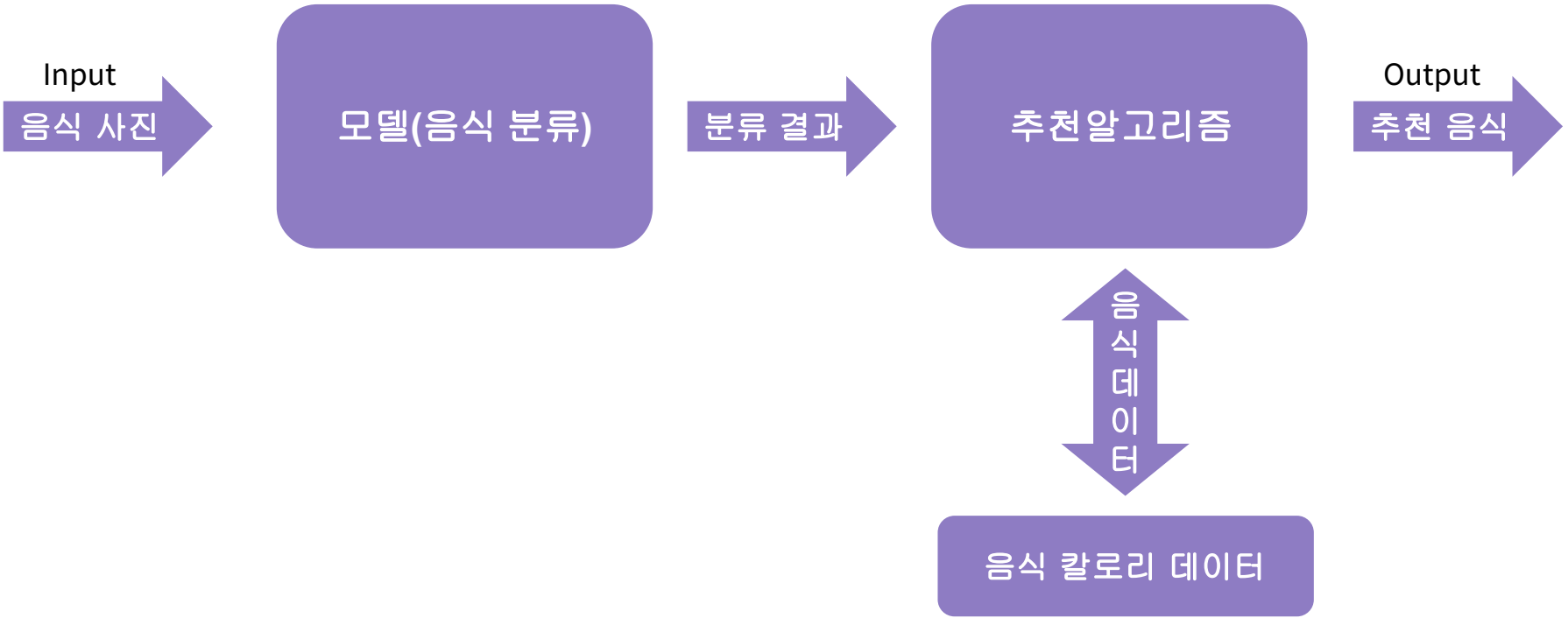
목차

- 프로젝트 배경
- 데이터 수집과 전처리
- 모델링
- 추천 알고리즘
- API 개발
- 결과물
- 개선점
- 문제 상황과 느낀점
- Q&A

프로젝트 배경

- 식단 관리를 하고 싶어하는 고객의 편의성 증진
- 매끼 무엇을 먹을지 고민하는 사람들을 위한 솔루션
- 이미지와 관련된 주제 중 음식 추천이 적합한 데이터가 갖춰져있음

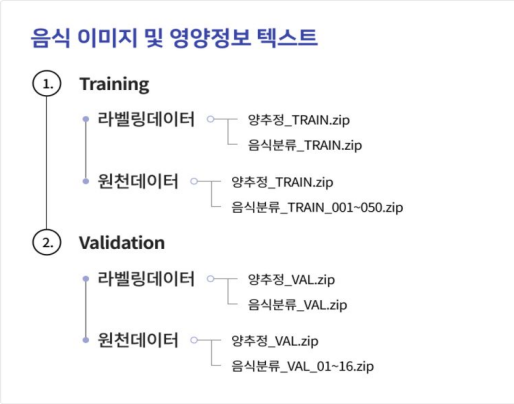
동작 과정



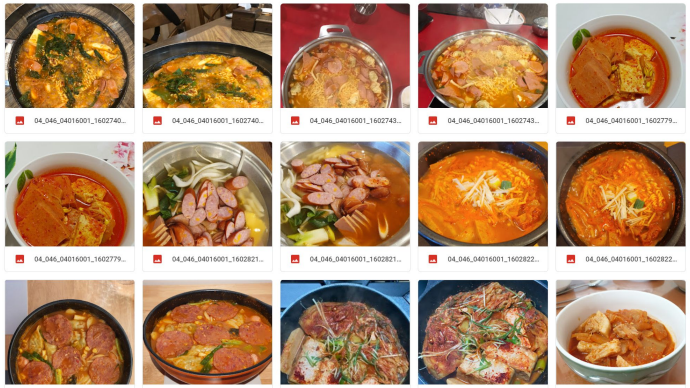
데이터 수집

- 음식 사진 데이터는 한국지능정보사회진흥원의 사업결과이며 AI hub 에서 수집
- 각 음식의 영양소 데이터는 농림수산식품교육문화정보원에서 수집
- 400종류 이상의 음식 분류와 각 2000장 이상의 총 용량 2TB 데이터 중
  5종류의 음식 분류와 각 300장의 이미지 파일을 다운받아 진행

데이터 구성



데이터 예시



데이터 전처리

이미지 넘파이 배열 및 라벨링 전환 작업

```
def imageprocessing(food_list):
  train = []
  test = []
  for idx, j in enumerate(food_list):
    # label = food_list[idx]
    label = [0 for i in range(len(food_list))]
    label[idx] =  1
    pro_food = os.listdir('/content/drive/MyDrive/final_project/food_img/'+j)
    for k in pro_food:
      img =load_img('/content/drive/MyDrive/final_project/food_img/'+j+'/'+k, target_size=(128,128))
      img = img.convert("RGB")
      imgArray = np.uint8(image.img_to_array(img)) / 255.0
      train.append(imgArray)
      test.append(label)
  img_Array = np.array(train)
  # img_label = test
  img_label = np.array(test)
  with open('food.pickle', 'wb') as f:
    pickle.dump(img_Array, f, pickle.HIGHEST_PROTOCOL)
  with open('food_label.pickle', 'wb') as f:
    pickle.dump(img_label, f, pickle.HIGHEST_PROTOCOL)
```

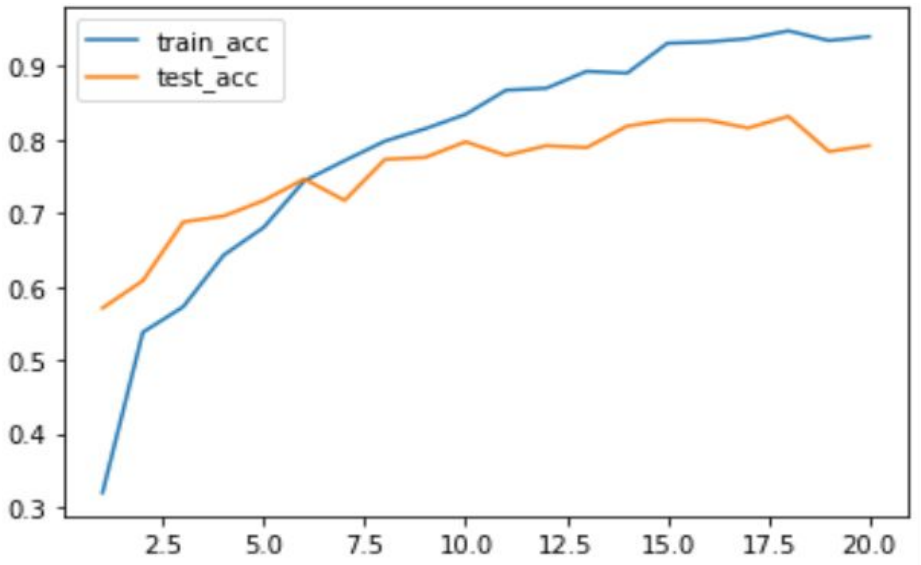
결과

```
[[[[0.73333333 0.56078431 0.41568627]
   [0.74117647 0.56862745 0.42352941]
   [0.75686275 0.58431373 0.43137255]
   ...
   [0.68627451 0.52156863 0.36470588]
   [0.69411765 0.52941176 0.37254902]
   [0.68627451 0.51372549 0.36862745]]

  [[0.7254902  0.55294118 0.40784314]
   [0.70980392 0.5372549  0.39215686]
   [0.73333333 0.56078431 0.40784314]
   ...
   [0.69019608 0.5254902  0.36862745]
   [0.69019608 0.5254902  0.36862745]
   [0.68235294 0.50980392 0.36470588]]


    [[1 0 0 0 0]
     [1 0 0 0 0]
     [1 0 0 0 0]
     ...
     [0 0 0 0 1]
     [0 0 0 0 1]
     [0 0 0 0 1]]
```

모델링

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 128, 128, 32)      896

max_pooling2d (MaxPooling2D  (None, 64, 64, 32)        0
)

conv2d_5 (Conv2D)            (None, 64, 64, 64)        18496

max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)        0
2D)

conv2d_6 (Conv2D)            (None, 32, 32, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)       0
2D)

conv2d_7 (Conv2D)            (None, 16, 16, 256)       295168

max_pooling2d_3 (MaxPooling  (None, 8, 8, 256)         0
2D)

flatten_1 (Flatten)          (None, 16384)             0

dense_2 (Dense)              (None, 512)               8389120

dropout (Dropout)            (None, 512)               0

dense_3 (Dense)              (None, 5)                 2565

=================================================================
Total params: 8,780,101
Trainable params: 8,780,101
Non-trainable params: 0
_________________________________________________________________
```

```
model1.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])
h1 = model1.fit(xtrain, ytrain, epochs=20, batch_size=20, validation_data=(xtest,ytest))
```



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 64)      1792

conv2d_1 (Conv2D)            (None, 124, 124, 32)      18464

max_pooling2d (MaxPooling2D  (None, 62, 62, 32)        0
)

dropout (Dropout)            (None, 62, 62, 32)        0

flatten (Flatten)            (None, 123008)            0

dense (Dense)                (None, 64)                7872576

dropout_1 (Dropout)          (None, 64)                0

dense_1 (Dense)              (None, 5)                 325

=================================================================
Total params: 7,893,157
Trainable params: 7,893,157
Non-trainable params: 0
_________________________________________________________________
```

```
[21] model.evaluate(new_data[:100],label[:100])

4/4 [==============================] - 2s 335ms/step - loss: 21.67
[21.6796817779541, 0.0]
```

```
conv2d_3 (Conv2D)            (None, 128, 128, 64)      1792

activation (Activation)      (None, 128, 128, 64)      0

max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)        0
2D)

dropout_2 (Dropout)          (None, 64, 64, 64)        0

conv2d_4 (Conv2D)            (None, 64, 64, 64)        36928

activation_1 (Activation)    (None, 64, 64, 64)        0

conv2d_5 (Conv2D)            (None, 62, 62, 32)        18464

max_pooling2d_2 (MaxPooling  (None, 31, 31, 32)        0
2D)

dropout_3 (Dropout)          (None, 31, 31, 32)        0

flatten_1 (Flatten)          (None, 30752)             0

dense_2 (Dense)              (None, 64)                1968192

dropout_4 (Dropout)          (None, 64)                0

dense_3 (Dense)              (None, 5)                 325

=================================================================
Total params: 2,025,701
Trainable params: 2,025,701
Non-trainable params: 0
_________________________________________________________________
```

```
model2.evaluate(new_data[:100],label[:100])

4/4 [==============================] - 1s 289ms/step - loss: 50
[50.568077087402344, 0.0]
```

```
conv2d (Conv2D)              (None, 128, 128, 128)     3584

activation (Activation)      (None, 128, 128, 128)     0

max_pooling2d (MaxPooling2D  (None, 64, 64, 128)       0
)

dropout (Dropout)            (None, 64, 64, 128)       0

conv2d_1 (Conv2D)            (None, 64, 64, 64)        73792

activation_1 (Activation)    (None, 64, 64, 64)        0

flatten (Flatten)            (None, 262144)            0

dense (Dense)                (None, 64)                16777280

dropout_1 (Dropout)          (None, 64)                0

dense_1 (Dense)              (None, 5)                 325

=================================================================
Total params: 16,854,981
Trainable params: 16,854,981
Non-trainable params: 0
_________________________________________________________________
```

```
model3.evaluate(new_data[:100],label[:100])

4/4 [==============================] - 2s 403ms/step - loss: 33.9
[33.91722106933594, 0.0]
```

```
conv2d (Conv2D)              (None, 128, 128, 128)     3584

activation (Activation)      (None, 128, 128, 128)     0

max_pooling2d (MaxPooling2D  (None, 64, 64, 128)       0
)

dropout (Dropout)            (None, 64, 64, 128)       0

conv2d_1 (Conv2D)            (None, 62, 62, 64)        73792

max_pooling2d_1 (MaxPooling  (None, 31, 31, 64)        0
2D)

dropout_1 (Dropout)          (None, 31, 31, 64)        0

flatten (Flatten)            (None, 61504)             0

dense (Dense)                (None, 64)                3936320

dropout_2 (Dropout)          (None, 64)                0

dense_1 (Dense)              (None, 5)                 325

=================================================================
Total params: 4,014,021
Trainable params: 4,014,021
Non-trainable params: 0
_________________________________________________________________
```

```
model8.evaluate(new_data[:100],label[:100])
```

```
4/4 [==============================] - 2s 566ms/step - loss: 1.
[1.0512664318084717, 0.8700000047683716]
```

```
conv2d_6 (Conv2D)            (None, 128, 128, 128)     3584

activation (Activation)      (None, 128, 128, 128)     0

max_pooling2d_6 (MaxPooling  (None, 64, 64, 128)       0
2D)

dropout_6 (Dropout)          (None, 64, 64, 128)       0

conv2d_7 (Conv2D)            (None, 64, 64, 128)       147584

activation_1 (Activation)    (None, 64, 64, 128)       0

conv2d_8 (Conv2D)            (None, 62, 62, 64)        73792

max_pooling2d_7 (MaxPooling  (None, 31, 31, 64)        0
2D)

dropout_7 (Dropout)          (None, 31, 31, 64)        0

flatten_2 (Flatten)          (None, 61504)             0

dense_4 (Dense)              (None, 64)                3936320

dropout_8 (Dropout)          (None, 64)                0

dense_5 (Dense)              (None, 5)                 325

=================================================================
Total params: 4,161,605
Trainable params: 4,161,605
Non-trainable params: 0
_________________________________________________________________
```

```
modelyj.evaluate(new_data[:100],label[:100])
```

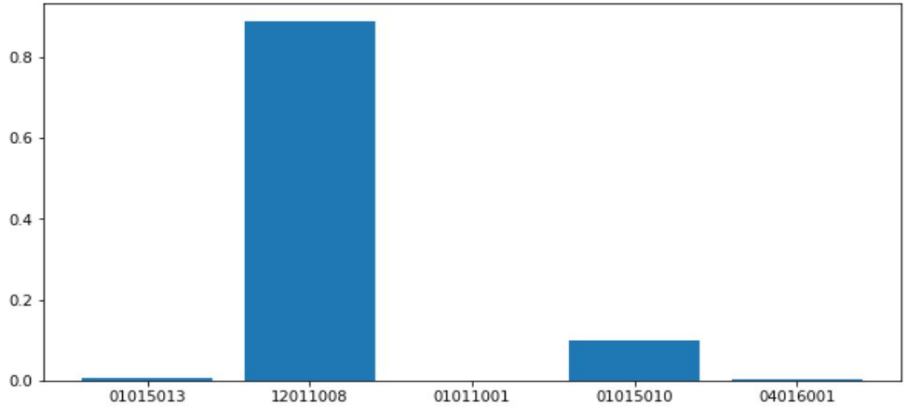
```
4/4 [==============================] - 4s 71ms/step - loss: 0.
[0.31864073872566223, 0.8500000238418579]
```

CNN을 이용한 음식 분류 모델 결과

input

output

result



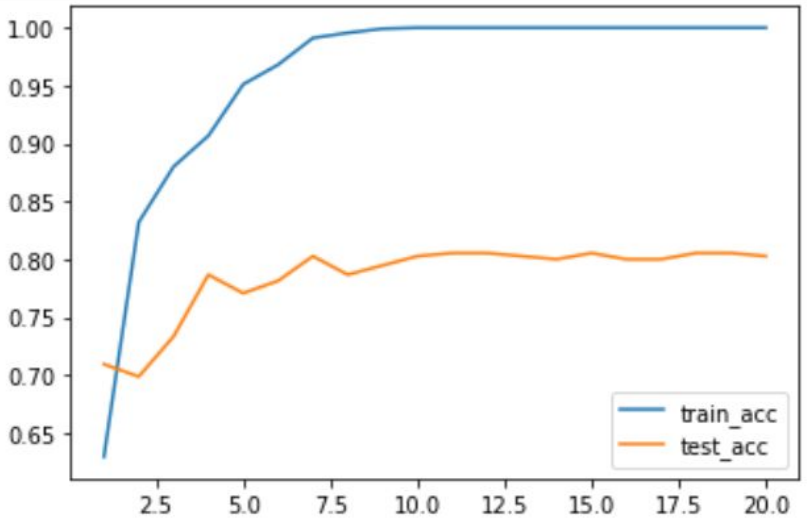
제육을 넣을 경우 김치로 인식하는 경우가 있었다. 실제 정확도가 떨어졌다.

→ 정확도를 높이기 위해 잘 학습된 전이학습의 모델을 사용

모델링

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 4, 4, 2048)        20861480

 flatten_2 (Flatten)         (None, 32768)             0

 dense_4 (Dense)             (None, 512)               16777728

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 37,641,773
Trainable params: 16,780,293
Non-trainable params: 20,861,480
_________________________________________________________________
```

```
model3.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc"])
h1 = model3.fit(xtrain, ytrain, epochs=20, batch_size=20, validation_data=(xtest,ytest))
```

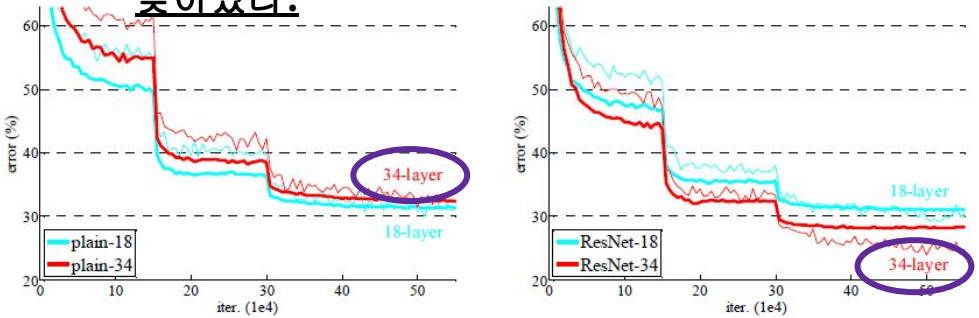


Transfer learning 모델

```
transforms_train = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(), # 데이터 증진(augmentation)
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # 정규화(normalization)
])

transforms_test = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Transfer learning 모델

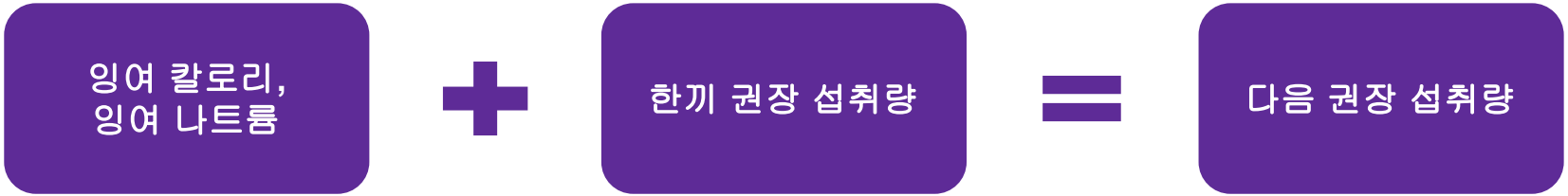
ResNet은 깊이가 깊어지면 에러확률이 더 낮아졌다.




```
# 사전 훈련 모델로 resnet34 사용
model = models.resnet34(pretrained=True)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 5)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

그래프 출처 : Deep Residual Learning for Image Recognition

추천 알고리즘



1. 입력값으로 성별을 받아 1일 권장 칼로리(남:2500kcal, 여:2000kcal)를 정합니다.
2. 분류 결과인 각 음식의 합인 총 섭취 칼로리양을 1일 권장 칼로리 기준으로 식사 한끼 분량에서 가감합니다. -> 잉여 칼로리
3. 다음 식사 한끼 칼로리에서 잉여 칼로리를 더합니다. -> 다음 권장 칼로리량
4. 음식 데이터에서 유클리디안 거리 기준으로 다음 식사 칼로리와 가장 가까운 음식을 뽑아냅니다.
5. 칼로리 기준으로 추출한 음식 중 최 상위 5개 중 위와 같은 방법으로 한끼 나트륨 권장 섭취량 기준으로 최종 거리가 가장 짧은 2가지 음식을 추천합니다.

음식 영양소 데이터 출처: 농림수산식품교육문화정보원
성인 기준 1일 권장 칼로리 섭취량 출처 :https://m.mk.co.kr/news/it/view/2021/09/864261/

웹 API 개발

Web API를 개방하여 외부프로그램이 분류모델을 이용할 수 있도록 함
Ngrok 을 이용하여 공인 ip주소를 할당받아 서버를 외부에 개방

```
run_with_ngrok(app)
app.run()
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Running on http://60cb-34-78-155-229.ngrok.io
 * Traffic stats available on http://127.0.0.1:4040
```

프로젝트 결과물

프로젝트 결과물

```
rice.jpg 분석중
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2651k  100    26  100 2651k      7   746k  0:00:03  0:00:03 --:--:--  746k

jaeyook.jpg 분석중
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2773k  100    26  100 2773k     11  1218k  0:00:02  0:00:02 --:--:-- 1218k

kimch.jpg 분석중
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 3728k  100    26  100 3728k      7  1068k  0:00:03  0:00:03 --:--:-- 1068k

curry.jpg 분석중
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  743k  100    26  100  743k     11   339k  0:00:02  0:00:02 --:--:--  339k
성별을 입력하세요 (m or f) : m
음식리스트: ['rice.jpg', 'jaeyook.jpg', 'kimch.jpg', 'curry.jpg']

먹은 음식 : ['쌀밥', '제육(돼지고기)', '김치(배추김치)', '카레라이스']
해당 칼로리 : [405.0, 124.0, 7.0, 433.8]
해당 나트륨 : [8.0, 47.0, 458.0, 1180.0]

추천 음식 : ['돼지고기덮밥' '회덮밥']
/content/drive/MyDrive/final_project/test#
```

실제값

분류값

추천값

개선점

1. 음식 양 반영
2. 객체 감지 모델 필요성
3. 학습되지 않은 종류의 음식데이터 입력시 다른 결과 출력
4. 어플리케이션 제작 및 배포

문제 상황과 느낀점

- **한정된 자원**

- 스토리지 부족으로 많은 양의 데이터를 모두 사용할 수 없었고, 좀 더 다양한 종류를 사용하기 위해 데이터를 한 종류씩 다운로드하여 확인 후 학습데이터로 만들었습니다.
- 하드웨어의 성능제한으로 더 다양한 모델링을 실행하지 못했습니다.

- **개인 맞춤형 음식 추천 알고리즘**
- **협업과 원활한 의사소통**

감사합니다

Q & A