음식 메뉴 분류를 통한 메뉴별 후기 정리 서비스

어떤 음식 '인지'

윤태웅

팀장

곽노아

팀원

한정원

팀원

홍성희

팀원

이미지 분류 기술

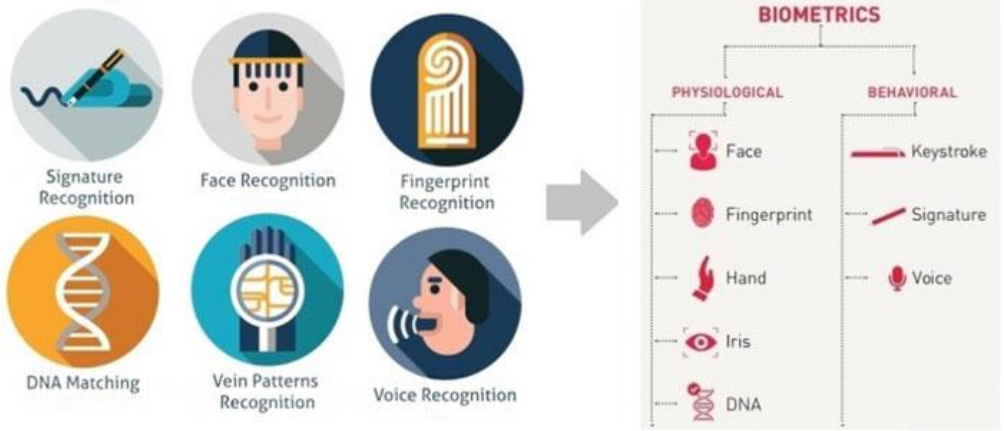
- 이미지의 속성 정보와 이미지의 피사체 구분을 통해 이미지 분석 및 분류
- 이미지 분류 기술을 통해 현재는 인물/패션/인테리어/식품 등 다양한 분야에서 서비스를 제공하고 있음.

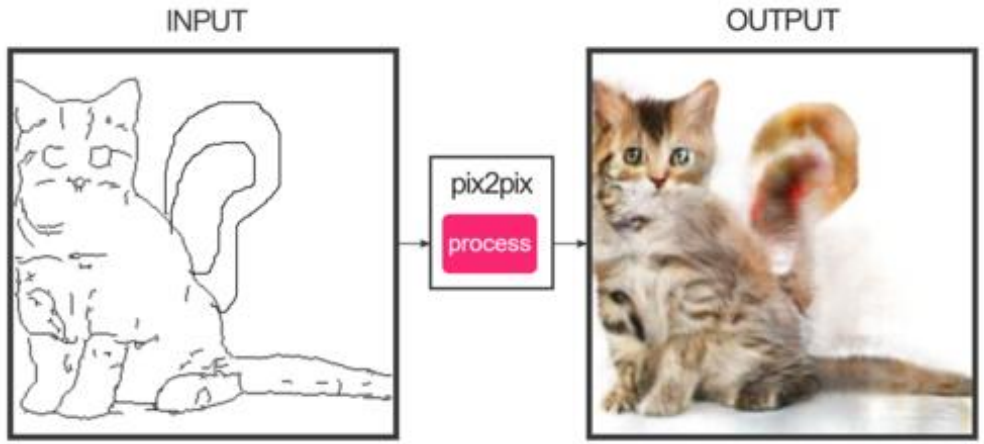
이미지 분류 기술 - 동향

- 식품 이미지 분류 기술은 다양한 분야에 적용해 사용되는 기술임.

- 환자를 위한 식품 추천 서비스
- 음식의 칼로리 계산을 통한 식단 조절 서비스
- 선호 음식 추천 서비스
- 관련 음식 매장 추천 서비스

이미지 분류 기술 - 동향





[생체 인식]

신체적 특성 기반
- 지문, 정맥 패턴, 눈(홍채 및 망막), 얼굴/DNA matching

행동적 특징 기반
- 보행, keystroke, 필적 인식

이용 범위
- 금융 기관, 의료 기관, 스마트폰 인증

[GAN]

- 수동적 인식에서 능동적 인식으로

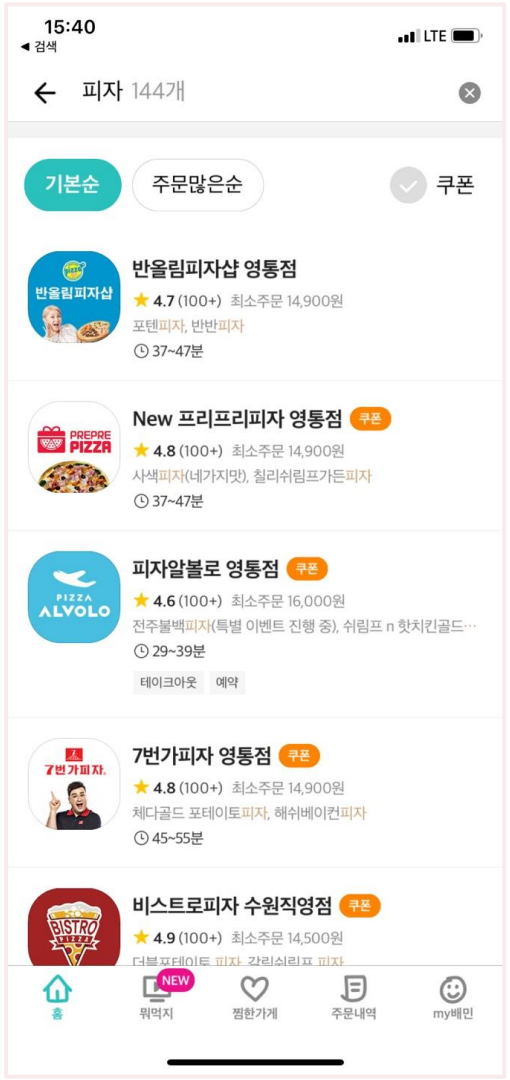
- 생성자(generator)와 구분자(discriminator)로 AI를 나눠 이미지를 제한된 데이터 내에서 생성

이용 범위
- 손상 이미지 복원, 신약 개발

주제 선정 과정

- ‘배달의 민족’, ‘요기요’와 같은 배달 어플리케이션에서 이미지 분류 기술 적용 가능성 탐색
- 어플리케이션 이용 고객의 메뉴 선택 폭 확대를 통한 편의성 증진 초점

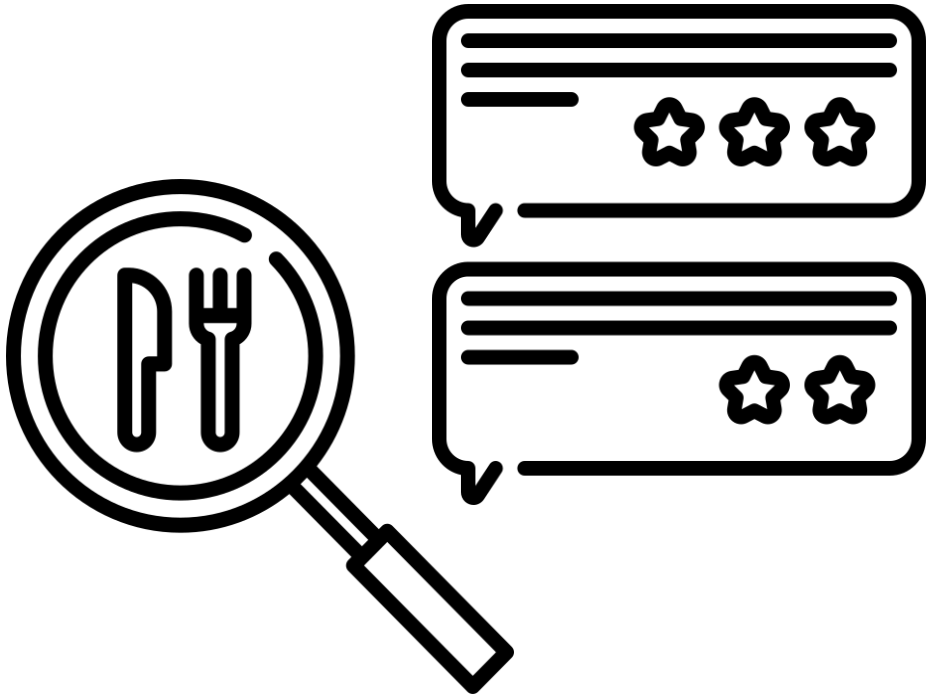
기존의 서비스 방식



- 메뉴 입력 시 검색결과는 매장별로만 나옴. ex)치킨 검색 시 패스트푸드점의 치킨에 대한 정보 부재.

- 원하는 메뉴의 후기만 검색하는 것에 어려움이 있음.

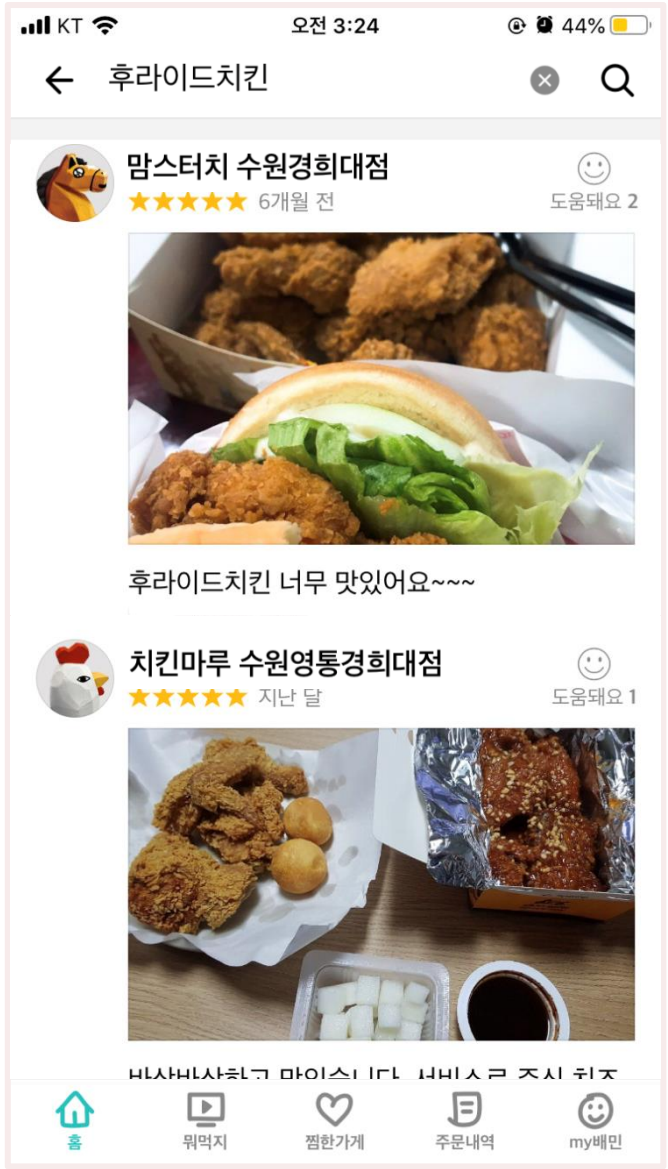
서비스의 방향



→ 올라와 있는 후기의 사진을 어떤 음식인지 분류해 해당 음식에 대한 후기를 모아서 제공하는 서비스

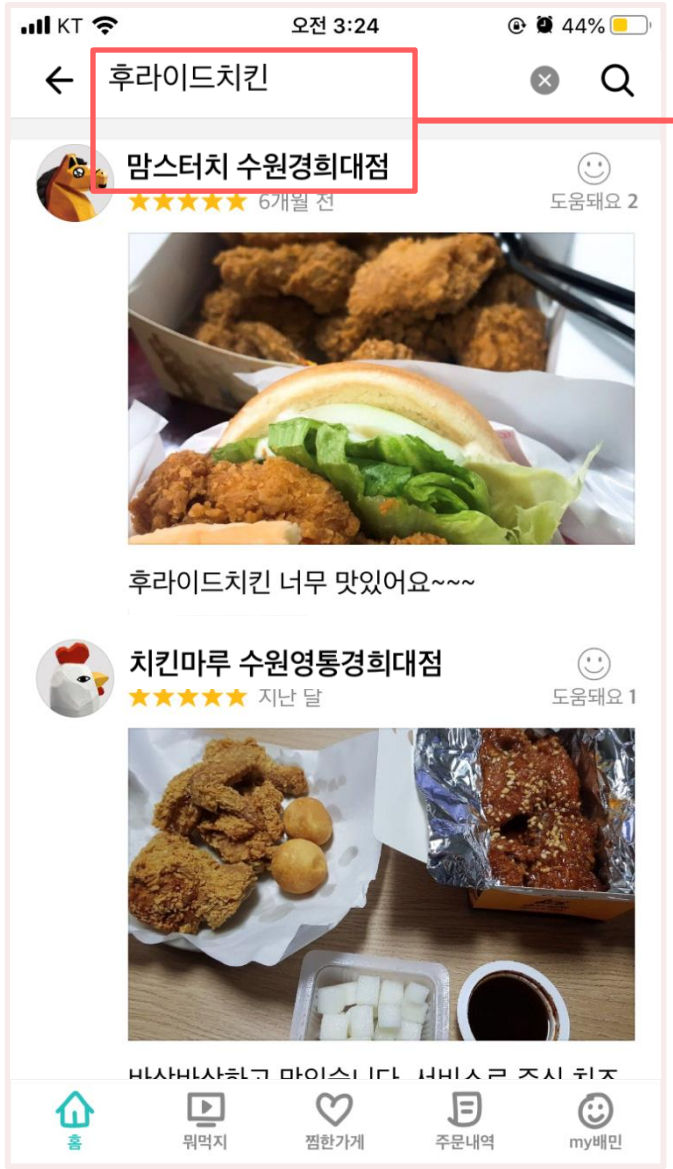
→ 원하는 메뉴에 대한 선택의 폭을 넓혀 보다 다양한 메뉴를 즐길 수 있게 해 주는 서비스

기술 목적



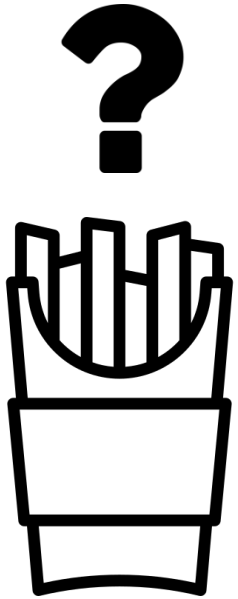
특정 메뉴 검색을 통한 모든 판매 매장의 리뷰 검색 활성화

기술 목적



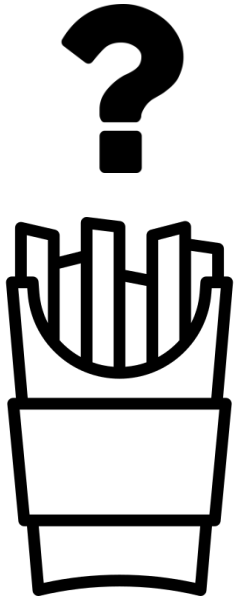
특정 메뉴 검색을 통한 모든 판매 매장의 리뷰 검색 활성화

기술 목적



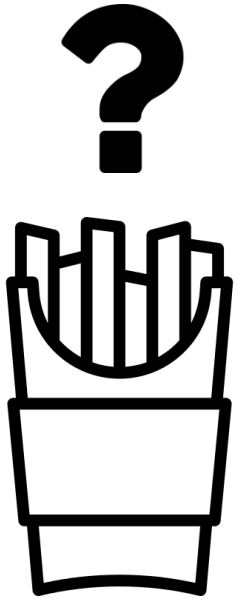
검색 결과에 표시되는 매장 증가를 통한
고객의 검색 편의성 향상

기술 목적



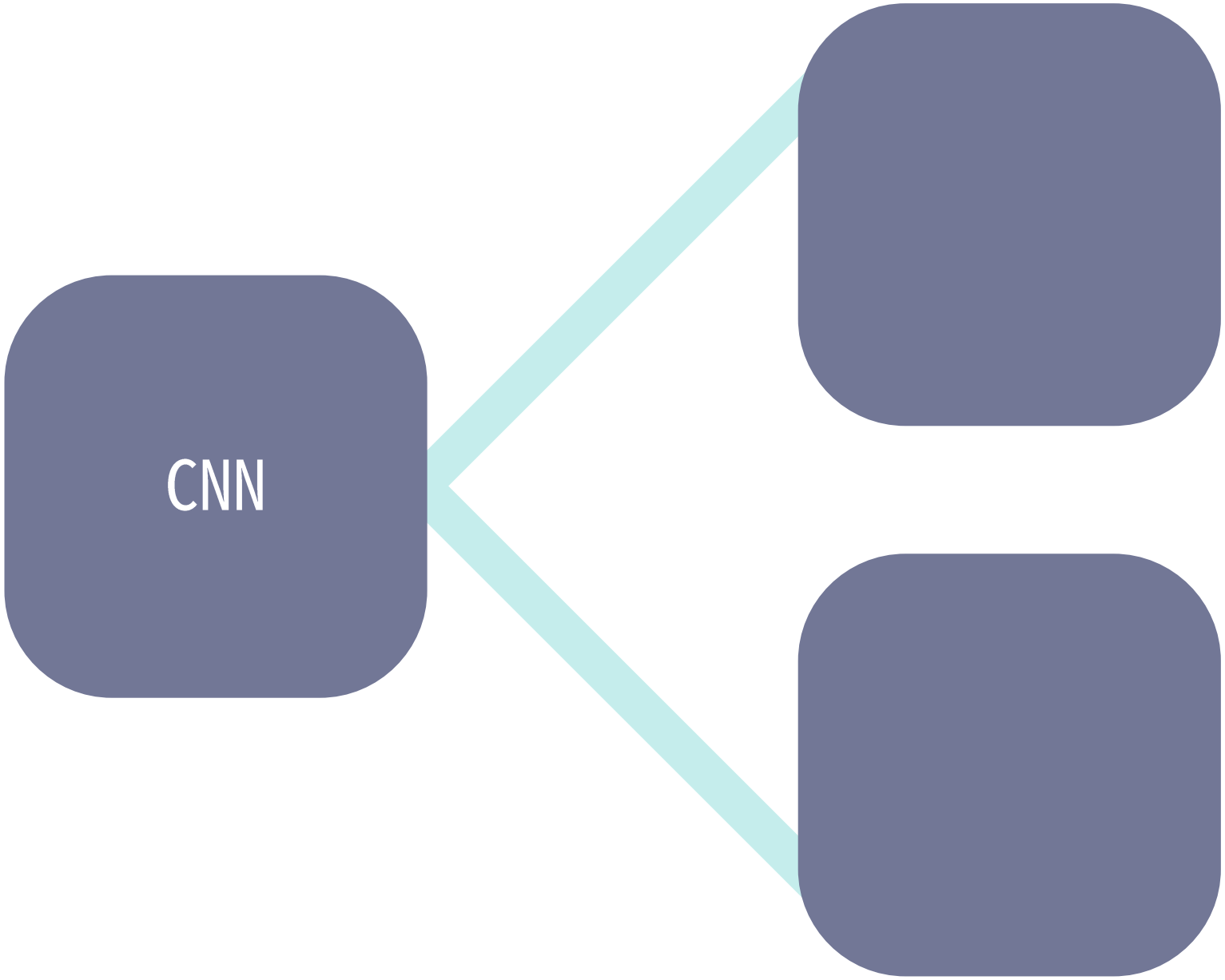
검색 결과에 표시되는 매장 증가를 통한
고객의 검색 편의성 향상

기술 목적

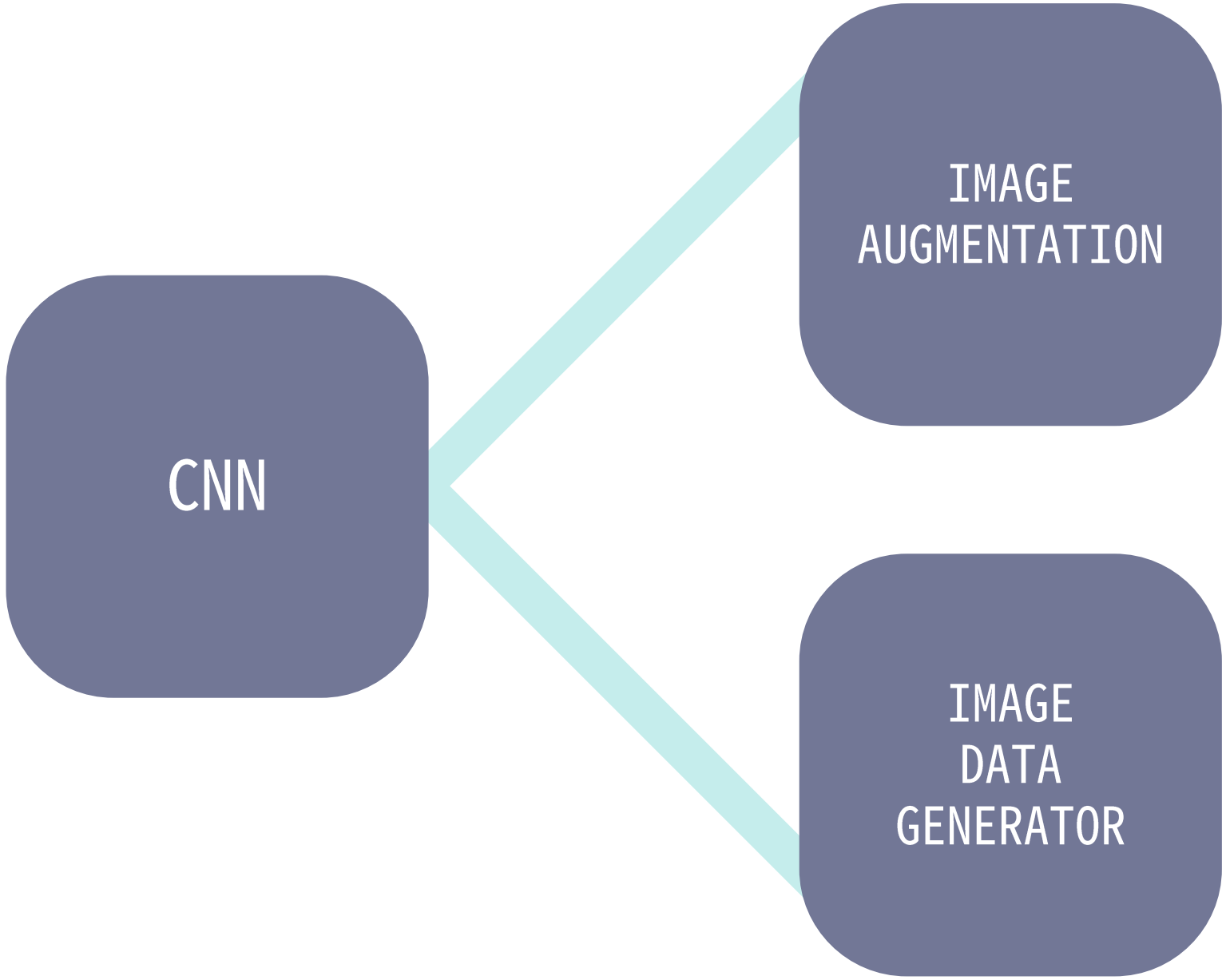


검색 결과에 표시되는 매장 증가를 통한
고객의 검색 편의성 향상

적용 기술



적용 기술



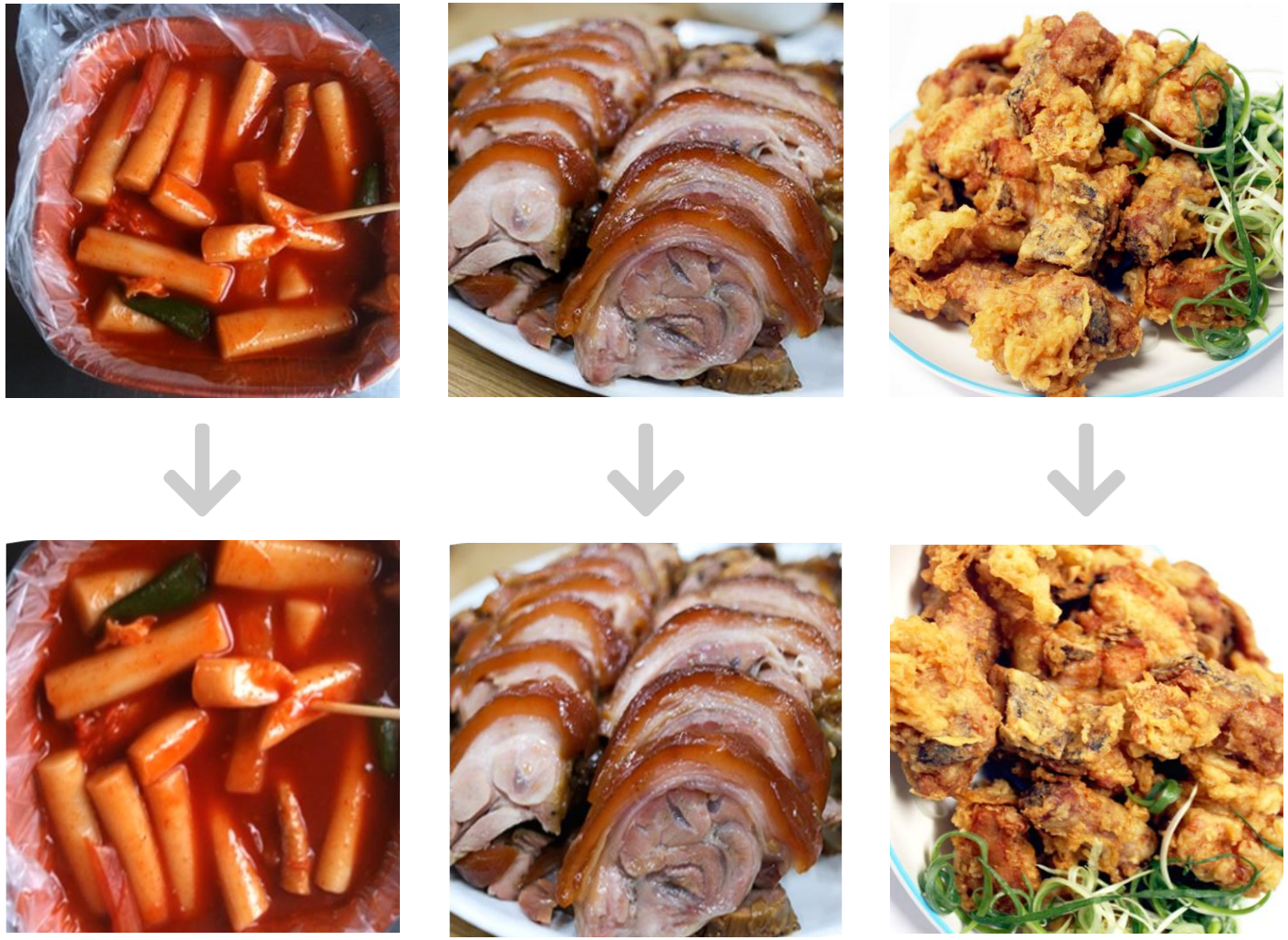
기본 데이터

Aihub에서 제공받은
배달음식 이미지 데이터 사용

5개 Category, 각 300장씩 총
1,500개의 image Group

DATA ENLARGEMENT

- Image augmentation을 통한 Data set 추가 확보



DATA ENLARGEMENT

- Image augmentation을 통한 Data set 추가 확보

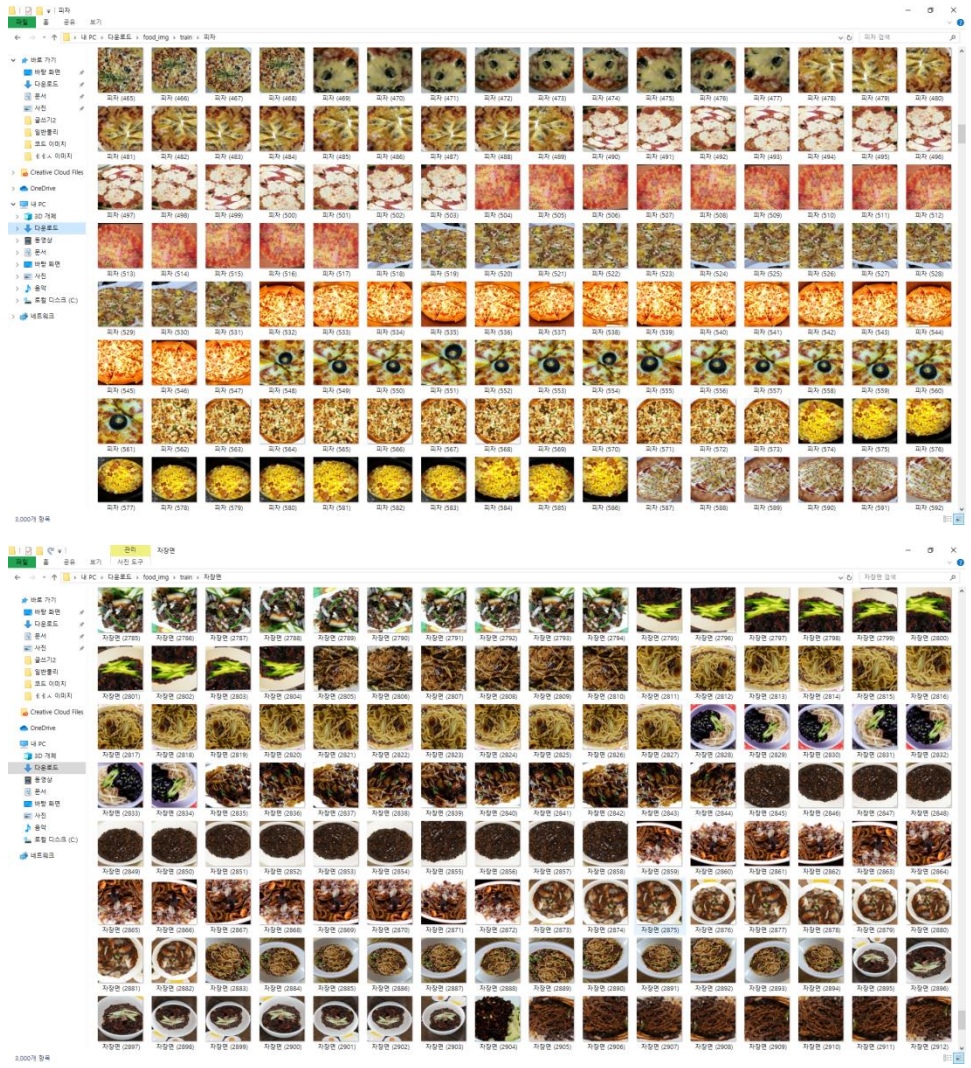


Image generator를 이용해
AI hub에서 선별한
기본 Dataset 1,500개를
15,000개로 Enlargement

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

1. Keras 호출

2. CNN 모델 생성
- Sequential 모델을 사용하여 레이어를 선형으로 연결하여 구성함.

- 64 * 64 size의 input data를 입력할 모델을 생성함.

- 1번째 Convolution에서 16개의 3 * 3 size Kernel을 사용함. 이때 RGB로 진행하므로 3개의 층을 사용함.

- 각 층에서의 활성화 함수의 출력 값 분포가 골고루 되도록 Batch Normalization 함.

- Convolution 활성화 함수로 ReLU function을 사용함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

1. Keras 호출

2. CNN 모델 생성
- Sequential 모델을 사용하여 레이어를 선형으로 연결하여 구성함.

- 64 * 64 size의 input data를 입력할 모델을 생성함.

- 1번째 Convolution에서 16개의 3 * 3 size Kernel을 사용함. 이때 RGB로 진행하므로 3개의 층을 사용함.

- 각 층에서의 활성화 함수의 출력 값 분포가 골고루 되도록 Batch Normalization 함.

- Convolution 활성화 함수로 ReLU function을 사용함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

1. Keras 호출

2. CNN 모델 생성
- Sequential 모델을 사용하여 레이어를 선형으로 연결하여 구성함.

- 64 * 64 size의 input data를 입력할 모델을 생성함.

- 1번째 Convolution에서 16개의 3 * 3 size Kernel을 사용함. 이때 RGB로 진행하므로 3개의 층을 사용함.

- 각 층에서의 활성화 함수의 출력 값 분포가 골고루 되도록 Batch Normalization 함.

- Convolution 활성화 함수로 ReLU function을 사용함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

1. Keras 호출

2. CNN 모델 생성
- Sequential 모델을 사용하여 레이어를 선형으로 연결하여 구성함.

- 64 * 64 size의 input data를 입력할 모델을 생성함.

- 1번째 Convolution에서 16개의 3 * 3 size Kernel을 사용함. 이때 RGB로 진행하므로 3개의 층을 사용함.

- 각 층에서의 활성화 함수의 출력 값 분포가 골고루 되도록 Batch Normalization 함.

- Convolution 활성화 함수로 ReLU function을 사용함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

1. Keras 호출

2. CNN 모델 생성
- Sequential 모델을 사용하여 레이어를 선형으로 연결하여 구성함.

- 64 * 64 size의 input data를 입력할 모델을 생성함.

- 1번째 Convolution에서 16개의 3 * 3 size Kernel을 사용함. 이때 RGB로 진행하므로 3개의 층을 사용함.

- 각 층에서의 활성화 함수의 출력 값 분포가 골고루 되도록 Batch Normalization 함.

- Convolution 활성화 함수로 ReLU function을 사용함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

1. Keras 호출

2. CNN 모델 생성
- Sequential 모델을 사용하여 레이어를 선형으로 연결하여 구성함.

- 64 * 64 size의 input data를 입력할 모델을 생성함.

- 1번째 Convolution에서 16개의 3 * 3 size Kernel을 사용함. 이때 RGB로 진행하므로 3개의 층을 사용함.

- 각 층에서의 활성화 함수의 출력 값 분포가 골고루 되도록 Batch Normalization 함.

- Convolution 활성화 함수로 ReLU function을 사용함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```python
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
import tensorflow as tf
from tensorflow.keras import layers, utils
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras.layers.normalization import BatchNormalization
import numpy as np


model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), padding='same', use_bias=False, input_shape=(64, 64, 3)))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(32, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(64, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
model.add(layers.Dropout(0.2))

model.add(layers.Conv2D(128, (3, 3), padding='same', use_bias=False))
model.add(layers.BatchNormalization(axis=3, scale=False))
model.add(layers.Activation("relu"))
model.add(layers.Flatten())
model.add(layers.Dropout(0.2))

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
model.summary()


model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

[학습모델 만들기]

- Kernel size는 2 * 2 로, stride는 2로 MaxPooling 함. 출력 크기를 입력 크기와 동일하게 padding 함.
- 2, 3, 4번째 Convolution에서 각각 32, 64, 128개의 3 * 3 size Kernel을 사용함.
- 과적합을 방지하기 위해 Dropout과 Flatten을 함.
- Flatten된 데이터를 512개의 Hidden unit과 Fully Connect 함.
- Multi-class Classification이므로 출력 노드의 활성화 함수는 softmax function을 사용함.
- 모델 optimizer는 adam을, loss function은 categorical cross-entropy를 사용하고, accuracy를 측정함.

처리 과정

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.7,
                                   zoom_range=[0.9, 2.2],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest',
                                   validation_split=0.20)

test_set = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="training")
validation_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,qh
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="validation")

test_set = test_set.flow_from_directory('food_img_test')


hist = model.fit_generator(training_set,
                           steps_per_epoch = 50,
                           epochs = 50,
                           validation_data = validation_set,
                           validation_steps = 10,
                           callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])


from keras.models import load_model

model.save('model.woong.h5')


print("-- Evaluate --")

scores = model.evaluate_generator(
            validation_set,
            steps = 10)

print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

[학습모델 만들기]

3-1. CNN 모델 학습

- 학습 데이터 set을 불러옴.
- ImageGenerator 함수를 통해 지정된 파라미터로 원본 이미지 데이터 부풀리기를 수행한 후, 결과를 특정 경로에 저장함.
- RGB scale을 0과 255에서 0과1 사이로 변환함.
- 지정된 각도/수평 · 수직방향 /밀림 강도/확대 · 축소 범위 내에서 임의로 원본 이미지를 변형시킴.
- 수평방향으로 이미지 반전을 수행함.

처리 과정

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.7,
                                   zoom_range=[0.9, 2.2],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest',
                                   validation_split=0.20)
test_set = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="training")
validation_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,qh
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="validation")

test_set = test_set.flow_from_directory('food_img_test')


hist = model.fit_generator(training_set,
                           steps_per_epoch = 50,
                           epochs = 50,
                           validation_data = validation_set,
                           validation_steps = 10,
                           callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])


from keras.models import load_model

model.save('model.woong.h5')


print("-- Evaluate --")

scores = model.evaluate_generator(
            validation_set,
            steps = 10)

print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

[학습모델 만들기]

3-1. CNN 모델 학습

- 학습 데이터 set을 불러옴.
- ImageGenerator 함수를 통해 지정된 파라미터로 원본 이미지 데이터 부풀리기를 수행한 후, 결과를 특정 경로에 저장함.
- RGB scale을 0과 255에서 0과1 사이로 변환함.
- 지정된 각도/수평 · 수직방향 /밀림 강도/확대 · 축소 범위 내에서 임의로 원본 이미지를 변형시킴.
- 수평방향으로 이미지 반전을 수행함.

처리 과정

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.7,
                                   zoom_range=[0.9, 2.2],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest',
                                   validation_split=0.20)

test_set = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="training")
validation_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,qh
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="validation")

test_set = test_set.flow_from_directory('food_img_test')


hist = model.fit_generator(training_set,
                           steps_per_epoch = 50,
                           epochs = 50,
                           validation_data = validation_set,
                           validation_steps = 10,
                           callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])


from keras.models import load_model

model.save('model.woong.h5')


print("-- Evaluate --")

scores = model.evaluate_generator(
            validation_set,
            steps = 10)

print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

[학습모델 만들기]

3-2. CNN 모델 학습

- Training Set, Validation Set, Test Set의 이미지 데이터를 불러옴.
- {"constant", "nearest", "reflect"/"wrap"} 중 택일하여 입력 경계 외부 점을 채움.
- 각 Set의 이미지 데이터를 랜덤하게 섞은 뒤, size를 64 * 64로 변환, Batch size = 50, Classification mode는 Categorical로 설정함.
- Training data와 Validation data로 모델을 학습시킴. Epoch = 50으로 설정 후 callback을 사용함.

처리 과정

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.7,
                                   zoom_range=[0.9, 2.2],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest',
                                   validation_split=0.20)

test_set = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="training")
validation_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,qh
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="validation")

test_set = test_set.flow_from_directory('food_img_test')


hist = model.fit_generator(training_set,
                           steps_per_epoch = 50,
                           epochs = 50,
                           validation_data = validation_set,
                           validation_steps = 10,
                           callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])


from keras.models import load_model

model.save('model.woong.h5')


print("-- Evaluate --")

scores = model.evaluate_generator(
            validation_set,
            steps = 10)

print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

[학습모델 만들기]

3-2. CNN 모델 학습

- Training Set, Validation Set, Test Set의 이미지 데이터를 불러옴.
- {"constant", "nearest", "reflect"/"wrap"} 중 택일하여 입력 경계 외부 점을 채움.
- 각 Set의 이미지 데이터를 랜덤하게 섞은 뒤, size를 64 * 64로 변환, Batch size = 50, Classification mode는 Categorical로 설정함.
- Training data와 Validation data로 모델을 학습시킴. Epoch = 50으로 설정 후 callback을 사용함.

처리 과정

```python
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.7,
                                   zoom_range=[0.9, 2.2],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest',
                                   validation_split=0.20)

test_set = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="training")

validation_set = train_datagen.flow_from_directory('food_img2',
                                                   shuffle=True,
                                                   seed=13,qh
                                                   target_size = (64, 64),
                                                   batch_size = 50,
                                                   class_mode = 'categorical',
                                                   subset="validation")

test_set = test_set.flow_from_directory('food_img_test')


hist = model.fit_generator(training_set,
                           steps_per_epoch = 50,
                           epochs = 50,
                           validation_data = validation_set,
                           validation_steps = 10,
                           callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])


from keras.models import load_model

model.save('model.woong.h5')


print("-- Evaluate --")

scores = model.evaluate_generator(
            validation_set,
            steps = 10)

print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

[학습모델 만들기]

3-2. CNN 모델 학습

- Training Set, Validation Set, Test Set의 이미지 데이터를 불러옴.
- {"constant", "nearest", "reflect"/"wrap"} 중 택일하여 입력 경계 외부 점을 채움.
- 각 Set의 이미지 데이터를 랜덤하게 섞은 뒤, size를 64 * 64로 변환, Batch size = 50, Classification mode는 Categorical로 설정함.
- Training data와 Validation data로 모델을 학습시킴. Epoch = 50으로 설정 후 callback을 사용함.

처리 과정

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.7,
                                   zoom_range=[0.9, 2.2],
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest',
                                   validation_split=0.20)

test_set = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="training")
validation_set = train_datagen.flow_from_directory('food_img2',
                                                 shuffle=True,
                                                 seed=13,qh
                                                 target_size = (64, 64),
                                                 batch_size = 50,
                                                 class_mode = 'categorical',
                                                 subset="validation")

test_set = test_set.flow_from_directory('food_img_test')


hist = model.fit_generator(training_set,
                           steps_per_epoch = 50,
                           epochs = 50,
                           validation_data = validation_set,
                           validation_steps = 10,
                           callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])


from keras.models import load_model

model.save('model.woong.h5')


print("-- Evaluate --")

scores = model.evaluate_generator(
            validation_set,
            steps = 10)

print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))
```

[학습모델 만들기]

3-2. CNN 모델 학습

- Training Set, Validation Set, Test Set의 이미지 데이터를 불러옴.
- {"constant", "nearest", "reflect"/"wrap"} 중 택일하여 입력 경계 외부 점을 채움.
- 각 Set의 이미지 데이터를 랜덤하게 섞은 뒤, size를 64 * 64로 변환, Batch size = 50, Classification mode는 Categorical로 설정함.
- Training data와 Validation data로 모델을 학습시킴. Epoch = 50으로 설정 후 callback을 사용함.

처리 과정

```
Epoch 17/50
50/50 [==============================] - 43s 858ms/step - loss: 0.5769 - acc: 0.7840 - val_loss: 0.8116 - val_acc: 0.6920
Epoch 18/50
50/50 [==============================] - 43s 857ms/step - loss: 0.5181 - acc: 0.8084 - val_loss: 1.2957 - val_acc: 0.6080
Epoch 19/50
50/50 [==============================] - 41s 823ms/step - loss: 0.5504 - acc: 0.8002 - val_loss: 0.9588 - val_acc: 0.6700
Epoch 20/50
50/50 [==============================] - 44s 881ms/step - loss: 0.5297 - acc: 0.8116 - val_loss: 0.8005 - val_acc: 0.6740
Epoch 21/50
50/50 [==============================] - 45s 891ms/step - loss: 0.5160 - acc: 0.8092 - val_loss: 1.6549 - val_acc: 0.6080
Epoch 22/50
50/50 [==============================] - 44s 874ms/step - loss: 0.5044 - acc: 0.8156 - val_loss: 1.0252 - val_acc: 0.6640
Epoch 23/50
50/50 [==============================] - 44s 874ms/step - loss: 0.5201 - acc: 0.8064 - val_loss: 1.1168 - val_acc: 0.6280
Epoch 24/50
50/50 [==============================] - 39s 790ms/step - loss: 0.5207 - acc: 0.8133 - val_loss: 0.9518 - val_acc: 0.6820
Epoch 25/50
50/50 [==============================] - 46s 927ms/step - loss: 0.5212 - acc: 0.8092 - val_loss: 0.9114 - val_acc: 0.6580
Epoch 26/50
50/50 [==============================] - 39s 776ms/step - loss: 0.4652 - acc: 0.8261 - val_loss: 0.8064 - val_acc: 0.7220
Epoch 27/50
50/50 [==============================] - 40s 807ms/step - loss: 0.4810 - acc: 0.8278 - val_loss: 1.3113 - val_acc: 0.6080
Epoch 28/50
50/50 [==============================] - 44s 887ms/step - loss: 0.4844 - acc: 0.8140 - val_loss: 0.7960 - val_acc: 0.7400
Epoch 29/50
50/50 [==============================] - 48s 970ms/step - loss: 0.4489 - acc: 0.8334 - val_loss: 1.0944 - val_acc: 0.6540
Epoch 30/50
50/50 [==============================] - 48s 953ms/step - loss: 0.4561 - acc: 0.8296 - val_loss: 0.8835 - val_acc: 0.7280
Epoch 31/50
50/50 [==============================] - 40s 807ms/step - loss: 0.4204 - acc: 0.8412 - val_loss: 0.7107 - val_acc: 0.7560
Epoch 32/50
50/50 [==============================] - 39s 777ms/step - loss: 0.4573 - acc: 0.8275 - val_loss: 1.4767 - val_acc: 0.5700
Epoch 33/50
50/50 [==============================] - 38s 764ms/step - loss: 0.4426 - acc: 0.8336 - val_loss: 0.8830 - val_acc: 0.6820
Epoch 34/50
50/50 [==============================] - 40s 792ms/step - loss: 0.4424 - acc: 0.8376 - val_loss: 0.9954 - val_acc: 0.6980
Epoch 35/50
50/50 [==============================] - 42s 842ms/step - loss: 0.4200 - acc: 0.8447 - val_loss: 0.9044 - val_acc: 0.7080
Epoch 36/50
50/50 [==============================] - 45s 907ms/step - loss: 0.4181 - acc: 0.8459 - val_loss: 0.7300 - val_acc: 0.7540
Epoch 37/50
50/50 [==============================] - 40s 803ms/step - loss: 0.4136 - acc: 0.8504 - val_loss: 1.3821 - val_acc: 0.6460
Epoch 38/50
50/50 [==============================] - 42s 842ms/step - loss: 0.4105 - acc: 0.8544 - val_loss: 0.8102 - val_acc: 0.7440
Epoch 39/50
50/50 [==============================] - 52s 1s/step - loss: 0.3972 - acc: 0.8496 - val_loss: 0.9699 - val_acc: 0.6940
Epoch 40/50
50/50 [==============================] - 41s 829ms/step - loss: 0.3818 - acc: 0.8645 - val_loss: 0.9863 - val_acc: 0.7180
Epoch 41/50
50/50 [==============================] - 43s 862ms/step - loss: 0.3655 - acc: 0.8724 - val_loss: 1.0009 - val_acc: 0.6880
-- Evaluate --
acc: 67.40%
```

[학습모델 만들기]

3-3. CNN 모델 학습

- 모델을 학습시킨 결과
- Epoch 41/50 에서 Early Stopping 발생함.
- Training data의 Accuracy 87.24%,
- Validation data의 Accuracy 68.80%,
- Test data의 Accuracy 67.40%의 결과를 도출함.

처리 과정

```
test_set = ImageDataGenerator(rescale = 1./255)


test_set = test_set.flow_from_directory('food_img_test',target_size = (64, 64))


print("-- Predict --")

output = model.predict_generator(
            test_set)
print(test_set.class_indices)
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})

print(output)
print(test_set.filenames)
```

[예측모델 만들기]

1. 모델을 h5 파일 형태로 저장함

2. Validation data로 학습시킨 모델의 Loss값과 Accuracy값을 구하여 평가함.

3. Training data 3879개, Validation data 966개, Test data 5개를 (0,1,2,3,4) class로 나눔.

4. 모델을 출력함.

처리 과정

```
Found 5 images belonging to 5 classes.
-- Predict --
{'떡볶이': 0, '자장면': 1, '족발': 2, '피자': 3, '후라이드치킨': 4}
[[0.002 0.000 0.001 0.685 0.312]
 [0.000 0.999 0.001 0.000 0.000]
 [0.006 0.011 0.664 0.311 0.008]
 [0.000 0.000 0.008 0.992 0.000]
 [0.995 0.000 0.000 0.004 0.001]]
['떡볶이₩₩떡볶이.png', '자장면₩₩자장면.jpg', '족발₩₩족발.jpg', '피자₩₩피자.jpg', '후라이드치킨₩₩치킨.jpg']
```

```
Found 6 images belonging to 6 classes.
-- Predict --
{'배추김치': 0, '삼계탕': 1, '순대': 2, '식혜': 3, '육회': 4, '후라이드치킨': 5}
[[0.999 0.000 0.000 0.000 0.001 0.000]
 [0.011 0.000 0.000 0.000 0.000 0.989]
 [0.000 0.066 0.000 0.922 0.001 0.011]
 [0.000 0.002 0.001 0.995 0.000 0.001]
 [0.029 0.000 0.000 0.000 0.967 0.004]
 [0.000 0.000 0.981 0.002 0.017 0.000]]
['배추김치₩₩Img_033_0008.png', '삼계탕₩₩Img_138_0100.jpg', '순대₩₩Img_130_0028.jpg', '식혜₩₩Img_092_0045.jpg', '육회₩₩Img_14
9_0084.jpg', '후라이드치킨₩₩Img_028_0050.jpg']
```

```
Found 5 images belonging to 5 classes.
-- Predict --
{'떡볶이': 0, '자장면': 1, '족발': 2, '피자': 3, '후라이드치킨': 4}
[[0.000 0.001 0.016 0.981 0.001]
 [0.000 1.000 0.000 0.000 0.000]
 [0.000 0.036 0.963 0.000 0.000]
 [0.279 0.000 0.000 0.719 0.002]
 [0.000 0.000 0.000 0.000 1.000]]
```

[예측모델 만들기]

- 총 10가지의 선정 리스트들을 변경하며 모델을 실행시킴.
- 학습 결과가 가장 좋은 메뉴 리스트를 분류 / 선정.
- 이번 모델의 학습결과로는 떡볶이, 자장면, 족발, 피자, 후라이드치킨이 가장 예측률이 높았음.

결과 및 평가

김치 - 떡볶이

피자 - 떡볶이

김치 – 떡볶이, 피자 – 떡볶이 등 몇몇 이미지를 분류하지 못하는 경우가 발생하였음.

결과 및 평가



떡볶이

족발


자장면

Dataset 중에서 육안으로 보기에도
인식하기 어려운 이미지들은
마찬가지로 인식하기 어려웠음.

결과 및 평가

음식 이미지 분류를 학습시키는 과정에서 적은 양의 데이터셋을 보완하기 위해 이미지 augmentation을 적용함.

모델의 epoch를 충분히 늘리지 못한 상태로 진행해 불안정한 accuracy 값을 보임.

결과 및 평가

음식 이미지 분류를 학습시키는 과정에서 적은 양의 데이터셋을 보완하기 위해 이미지 augmentation을 적용함.
모델의 epoch를 충분히 늘리지 못한 상태로 진행해 불안정한 accuracy 값을 보임.

음식 사진의 색상이 조명에 의해 일정하지 않거나 다른 음식과 함께 있을 경우, accuracy가 현저히 낮아지기 때문에 한 가지 메뉴만 있는 사진 데이터의 classification에 적합한 모델임.

결과 및 평가

음식 이미지 분류를 학습시키는 과정에서 적은 양의 데이터셋을 보완하기 위해 이미지 augmentation을 적용함.
모델의 epoch를 충분히 늘리지 못한 상태로 진행해 불안정한 accuracy 값을 보임.

음식 사진의 색상이 조명에 의해 일정하지 않거나 다른 음식과 함께 있을 경우, accuracy가 현저히 낮아지기 때문에 한 가지 메뉴만 있는 사진 데이터의 classification에 적합한 모델임.

개별 메뉴에 대한 리뷰사진 구분에 있어서는 어느 정도 학습된 모습을 보임.
object recognition에 대한 학습을 추가로 한다면 시중의 리뷰 사진도 잘 분류해 낼 것임.

감사합니다.