2021 DATA
CREATOR CAMP

2021년 데이터 크리에이터 캠프

-12회차 오합지존-

과학기술정보통신부 | NIA 한국지능정보사회진흥원

01. 데이터 파악 (EDA)

KFOOD DATASET

한국의 다양한 음식 사진을 담은 이미지 데이터셋
이 음식 사진들을 5개 종류로 분류하는 것이 목표

Train

모델을 학습시키기 위한 데이터셋으로,
지도학습을 위해 조림 / 김치 / 면 / 쌀 / 구이 음식으로 분류되어 있다.
각 카테고리 별로 597 / 596 / 766 / 766 / 766 개, 총 3,491개의 이미지를 포함한다.

Val

Training dataset을 통해 생성한 모델의 성능을 가늠하기 위한 데이터셋으로,
마찬가지로 5개 음식으로 분류되어 있다.
각 카테고리 별로 117 / 117 / 150 / 150 / 150 개, 총 684개의 이미지를 포함한다.

Test

사용할 모델이 결정된 후, 최종적으로 모델의 성능을 측정하기 위한 데이터셋으로,
분류되지 않은 958개 이미지를 포함한다.

01. 데이터셋 특징 파악

1. 같은 클래스 내에서도, 이미지들이 **일관적인 색깔을 띄고 있지 않기에** 이미지 속 **색깔을 통해 feature를 도출해내는 데 어려움**이 존재한다.

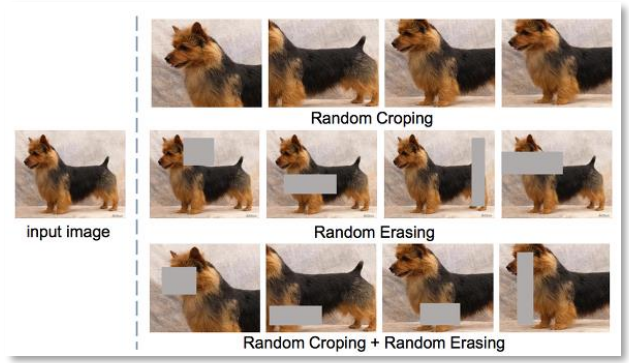
Noodle 클래스의 예시 train 이미지 3장.
각 이미지는 일관되지 않은 색을 띄고 있음을 알 수 있다.

해결책 모색 :
모델이 이미지의 feature를 도출해내는 데 있어 **이미지가 띄는 색에 대한 비중을 낮추었다**.
(augmentation 하이퍼 파라미터에 있어 gray_scale, color_jitter 등의 사용)

01. 데이터셋 특징 파악

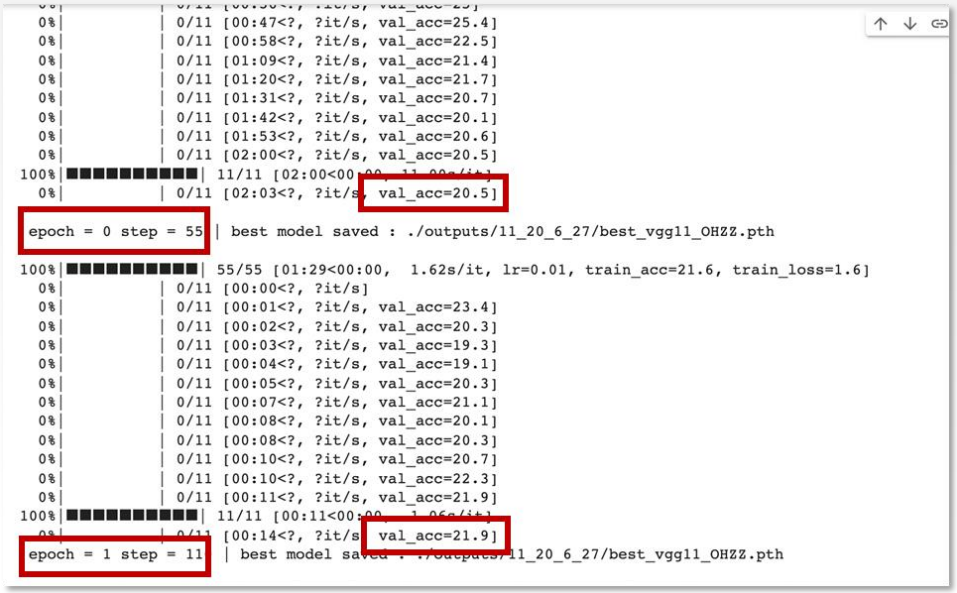
2. 각 이미지가 한 개의 음식만을 포함하고 있지 않아,
한 개의 이미지를 한 개의 클래스로 명확히 분류하기 힘든 경우가 존재한다.

Noodle 클래스에 속하지만,
국수를 제외한 타 음식 또는 물체를
이미지 내에 포함하는 예시 사진 2장.



Random crop을 사용할 경우,
이미지 타겟이 의도와 다르게 설정될 수 있기에
Crop image size를 크게 설정하였다.

02 - 1. 학습 실행 (epoch 증가)




```
4.11s/it, lr=0.01, train_acc=48.1, train_loss=1.24]
```

Epoch을 50으로 설정했을 때 Accuracy가 증가한 모습

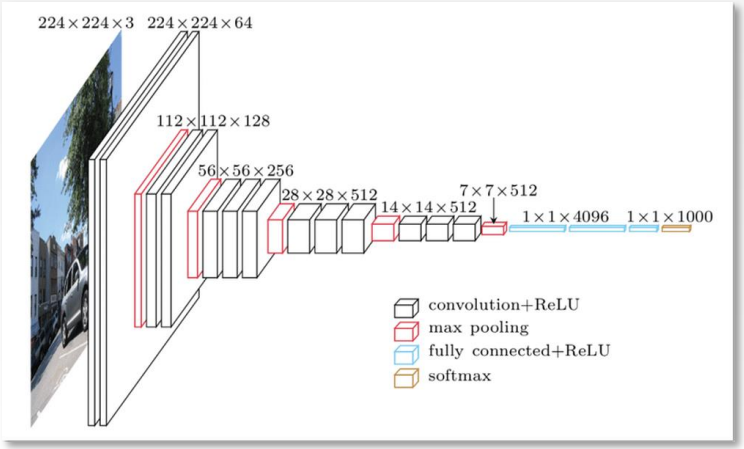
우선 학습 횟수를 증가시키며, 성능 향상을 기대하였다.

베이스 라인 코드의 타 하이퍼 파라미터는 디폴트로 설정 후, **Epoch만 증가시켜보았을 때**

val 데이터셋에 대해 정확도가 **약 1.4% 증가**한 것을 확인하였다.

따라서, 추후 여러 model architecture, optimizer 등을 사용하는 시도에 있어 가능한 높은 epoch에 대해 시도해보고자 하였다.

02 - 2. 학습 실행 (model_arch 변경)



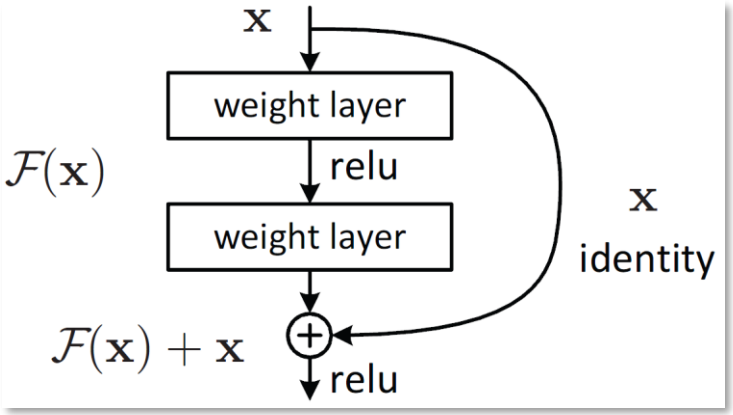
디폴트 모델 구조인 vgg11

```
  0%|          | 0/11 [00:15<?, ?it/s, val_acc=21.9]
100%|██████████| 55/55 [03:02<00:00,  3.33s/it, lr=0.01, train_acc=21.6, train_loss=1.62]
```

베이스 코드를 실행해보았을 때,

21.6%의 val accuracy를 기록했다.

이에, 모델의 성능을 향상시키기 위해 **타 모델 구조를 사용**해보았다.

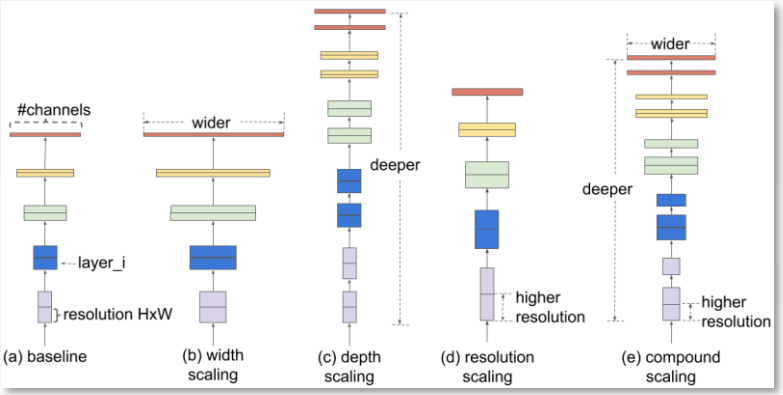


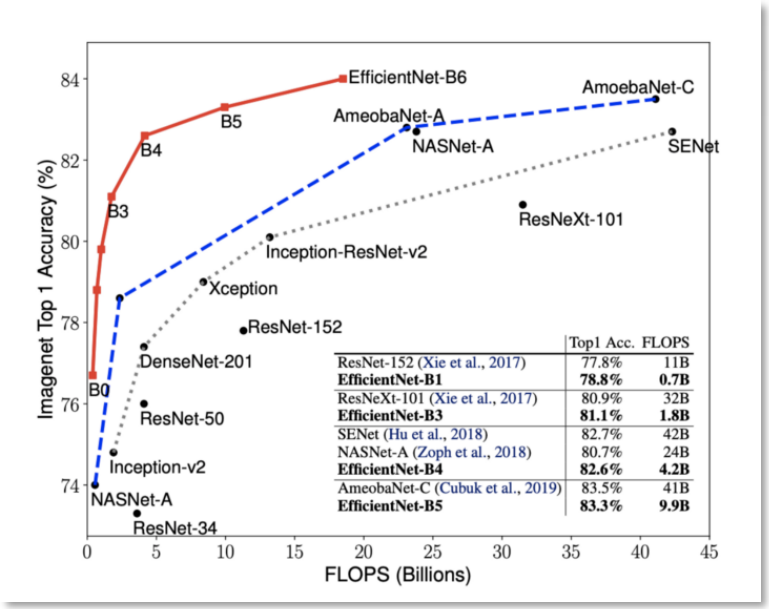
두 번째로 사용한 모델이 resnet18이다.

```
  0%|          | 0/11 [00:15<?, ?it/s, val_acc=21.9]
100%|██████████| 55/55 [03:02<00:00,  3.33s/it, lr=0.01, train_acc=21.6, train_loss=1.62]
```

Resnet을 사용해보았을 때도,
성능적 관점에서 vgg와 큰 차이를 도출해낼 수 없었다.

02 - 2. 학습 실행 (model_arch 변경)





마지막으로 사용해본 모델 구조인 efficient

efficient_b4로 모델 구조를 설정하고, epoch 등을 조정했을 때

Val accuracy가 64.3%를 기록하며

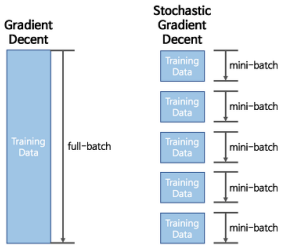
상당히 발전된 성능을 확인했다.

02 - 3. 학습 실행 (optimizer 변경)

<Stochastic gradient decent(SGD)>

full-batch가 아닌 mini batch로 학습을 진행하는 것

(* batch로 학습하는 이유 : full-batch로 epoch마다 weight를 수정하지 않고 빠르게 mini-batch로 weight를 수정하면서 학습하기 위해)



GD vs SGD (출처 : https://seamless.tistory.com/38)

디폴트로 주어진 optimizer인 SGD

초반 베이스 코드로 val 데이터를 분류해보았을 때 얻은

낮은 accuracy를 극복하기 위해,

성능이 개선된 optimizer를 대신 사용해보았다.

Momentum과 Adagrad는 각각 v와 h가 처음에 0으로 초기화되면 W가 학습 초반에 0으로 biased되는 문제가 있다.

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

Momentum notation1

$$h_i \leftarrow \rho h_{i-1} + (1-\rho)\frac{\partial L_i}{\partial W} \odot \frac{\partial L_i}{\partial W}$$

RMSProp notation

그리하여 사용한 optimizer가 **Adam**이다.

Adam은 momentum + RMSprop 이라고 생각될 수 있는데,

그 원리를 간단히 요약하면

Momentum과 최신 grad 결과를 더 반영해 update하는 것이다.

02 - 4. 학습 실행 (augmentation 추가)

RandomRotation

The `RandomRotation` transform (see also `rotate()`) rotates an image with random angle.

```
rotater = T.RandomRotation(degrees=(0, 180))
rotated_imgs = [rotater(orig_img) for _ in range(4)]
plot(rotated_imgs)
```

Original image

```
def get_transform(size=224, random_crop=True, use_flip=True, use_color_jitter=True, use_gray_scale=True, use_normalize=True):
    if random_crop:
        resize_crop = transforms.RandomResizedCrop(size=size)
    else:
        resize_crop = transforms.Compose([transforms.Resize(size), transforms.CenterCrop(size)])
    random_flip = transforms.RandomHorizontalFlip(p=0.5)
    v_flip = transforms.RandomVerticalFlip(p=0.5)
    color_jitter = transforms.RandomApply([
        transforms.ColorJitter(0.8, 0.8, 0.8, 0.2)
    ], p=0.8)
    gray_scale = transforms.RandomGrayscale(p=0.2)
    normalize = transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
    rotation = transforms.RandomRotation(degrees=(0,180))
    to_tensor = transforms.ToTensor()

    transforms_array = np.array([resize_crop, v_flip, random_flip, color_jitter, gray_scale, to_tensor, normalize, rotation])
    transforms_mask = np.array([True, True, use_flip, use_color_jitter, use_gray_scale, True, use_normalize, True])

    transform = transforms.Compose(transforms_array[transforms_mask])

    return transform
```

RandomVerticalFlip

The `RandomVerticalFlip` transform (see also `vflip()`) performs vertical flip of an image, with a given probability.

```
vflipper = T.RandomVerticalFlip(p=0.5)
transformed_imgs = [vflipper(orig_img) for _ in range(4)]
plot(transformed_imgs)
```

Original image

클래스에 대한 이미지의 **feature를 명확히 추출**해내기 위해,
Random flip/rotation Augmentation을 추가해보았다.

```
100%|██████████| 55/55 [01:32<00:00,  1.68s/it, lr=0.01 train_acc=22.1, train_loss=1.6]
```

기존 resnet만을 사용했을 때 기록했던 21.6%에 비해,

Accuracy가 향상된 모습을 보여

해당 augmentation 또한 최종 모델에 추가하였다.

03. 최종 사용 모델

앞선 과정들을 거쳐, **성능적 관점에서 가장 뛰어난 모델**은
다음 augmentation과 하이퍼 파라미터를 포함하였다.

model_arch : efficientnet_b4

epoch : 20 (정해진 시간 내에 수행할 수 있는 가장 높은 수치의 epoch으로 설정)

augmentation : pretrained, color_jitter, gray_scale, random_crop, normalize 등
성능 향상에 있어 긍정적 영향을 주었던 모든 augmentation을 사용하였다.

optimizer : adam

batch_size : 64

03. 최종 사용 모델

```python
config = ConfigTree()

config.DATASET.ROOT = "./kfood" # 데이터 위치
config.DATASET.NUM_CLASSES = 5 # 분류해야 하는 클래스 종류의 수

config.BASE_SAVE_DIR = './outputs' # 모델 파라미터가 저장되는 위치

config.SYSTEM.GPU = 0 # GPU 번호
config.SYSTEM.PRINT_FREQ = 5 # 로그를 프린트하는 주기

# 실험 요소
config.TRAIN.EPOCH = 50 # 총 학습 에폭
config.TRAIN.BATCH_SIZE = 64 # 배치 사이즈
config.TRAIN.BASE_LR = 0.01 # 러닝 레이트
config.TRAIN.AUGMENTATION = {'size' : 224, 'random_crop' : True, 'use_flip' : True, 'use_color_jitter' : True, 'use_gray_scale' : False, 'use_normalize' : True}
config.TRAIN.WEIGHT_DECAY = 1e-2 # 모델 파라미터 제약

# config.MODEL.ARCH = 'resnet' # 모델 구조
config.MODEL.ARCH = 'efficientnet_b4' # 모델 구조
# config.MODEL.OPTIM = 'SGD' # 파라미터 업데이터(optimizer)
config.MODEL.OPTIM = 'Adam' # 파라미터 업데이터(optimizer)
config.MODEL.PRETRAIN = True

config.TEAM_NAME = 'OHZZ' # 소속 팀 이름(띄어쓰기 없이 영어 알파벳으로만 구성되게 작성)
```

```
100%|| 55/55 [22:07<00:00, 24.13s/it, lr=0.01, train_acc=61.5, train_loss=1.11]
  0%|                                                                          | 0/11 [00:00<?, ?it/s]
  0%|                                                                | 0/11 [00:23<?, ?it/s, val_acc=67.2]
  0%|                                                                | 0/11 [00:48<?, ?it/s, val_acc=68.8]
  0%|                                                                | 0/11 [01:13<?, ?it/s, val_acc=67.7]
  0%|                                                                | 0/11 [01:37<?, ?it/s, val_acc=68.4]
  0%|                                                                | 0/11 [02:02<?, ?it/s, val_acc=67.8]
  0%|                                                                 | 0/11 [02:26<?, ?it/s, val_acc=68]
  0%|                                                                | 0/11 [02:51<?, ?it/s, val_acc=67.2]
  0%|                                                                | 0/11 [03:15<?, ?it/s, val_acc=66.8]
  0%|                                                                | 0/11 [03:39<?, ?it/s, val_acc=67.4]
  0%|                                                                | 0/11 [04:04<?, ?it/s, val_acc=67.7]
  0%|                                                                | 0/11 [04:20<?, ?it/s, val_acc=68.6]
100%|                                                               | 11/11 [04:20<00:00, 23.71s/it]
  0%|                                                                | 0/11 [04:21<?, ?it/s, val_acc=68.6]

epoch = 16 step = 935 | best model saved : ./outputs/11_20_16_4/best_efficientnet_b4_OHZZ.pth
```

2	OHZZ	0.68292

최종 사용 모델로 데이터를 분류해보았을 때,

약 69%의 Val_accuracy와 Test_accuracy를 얻어낼 수 있었다.

감사합니다

2021 DATA CREATORCAMP