



DACON : Ego-Vision 손동작 인식 경진대회

Private 1st, RegNet 솔루션 분석

2기 임원진 장에서

목차

#01 Introduction

#02 Data Preprocessing

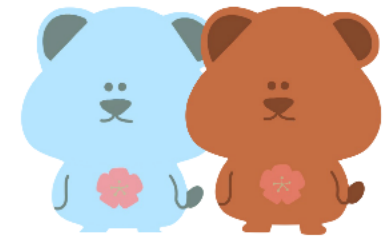
#03 Cross Validation by K-fold

#04. Network Model: RegNet

#05. Conclusion




1. Introduction



#1. 데이터 소개

```
|-- train
|   |-- 0
|   |   |-- 0.png : 연속된 Image 중 첫 번째 이미지 (가장 낮은 숫자)
|   |   |-- 1.png : 연속된 Image 중 두 번째 이미지 (두 번째로 낮은 숫자)
|   |   |-- 2.png : 연속된 Image 중 세 번째 이미지 (세 번째로 낮은 숫자)
|   |   |   ...      : 숫자의 크기 순으로 연속된 이미지들이 저장됨
|   |   |-- 0.json : 폴더명과 동일한 이름의 json 파일
|   |               연속된 Image에 대한 포괄 정보
|   |               actor : 배우에 대한 정보
|   |               image : 이미지에 대한 정보
|   |               annotations : 이미지의 id, label, key_point 정보
|   |-- 648
|-- test
|   |-- 649
|   |   |-- 0.png
|   |   |-- 1.png
|   |   |-- 2.png
|   |   |   ...
|   |   |-- 649.json
|   |-- 865

|-- hand_gesture_pose.csv : 손동작에 대한 정보
|-- sample_submission.csv : 제출 파일 Sample_submission
```



Train 649장
Test 217장 (매우 적음!)
총 데이터는 5888개

#1. 데이터 소개

```
| -- 0
|   |-- 0.png : 연속된 Image 중 첫 번째 이미지 (가장 낮은 숫자)
|   |-- 1.png : 연속된 Image 중 두 번째 이미지 (두 번째로 낮은 숫자)
|   |-- 2.png : 연속된 Image 중 세 번째 이미지 (세 번째로 낮은 숫자)
|   |       ... : 숫자의 크기 순으로 연속된 이미지들이 저장됨
```

- 한 train label 당 하나의 이미지가 있는 것이 아닌, 여러 이미지가 존재함
- 연속된 이미지들이 인덱싱되어 저장되어 있음



#1. 평가 기준

1차 평가: 계량 평가

- **Logloss**
- 손 동작의 Label 예측

2차 평가: 비계량 평가

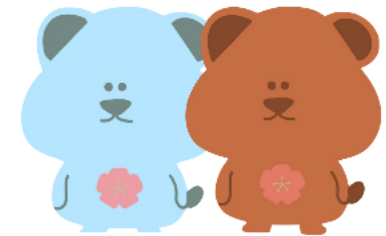
Logloss?

ID	Actual	Predicted Probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.02687
ID1	1	0.9	0.9	-0.04576
ID7	1	0.78	0.78	-0.10791
ID8	0	0.56	0.44	-0.35655
ID2	0	0.51	0.49	-0.3098
ID3	1	0.47	0.47	-0.3279
ID4	1	0.32	0.32	-0.49485
ID5	0	0.1	0.9	-0.04576

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

2. Data Preprocessing



#2. 기본 Preprocessing

Remove noise keypoints

데이터 분석을 통해

Train dataset에 475, 543 폴더는 의도하지 않은
나머지 손에 대해서도 Keypoint가 잡히게 됨

→ Json의 Keypoint를 사용하기 위해 475,543 폴더인 경우 해당 부분 Keypoint 제거

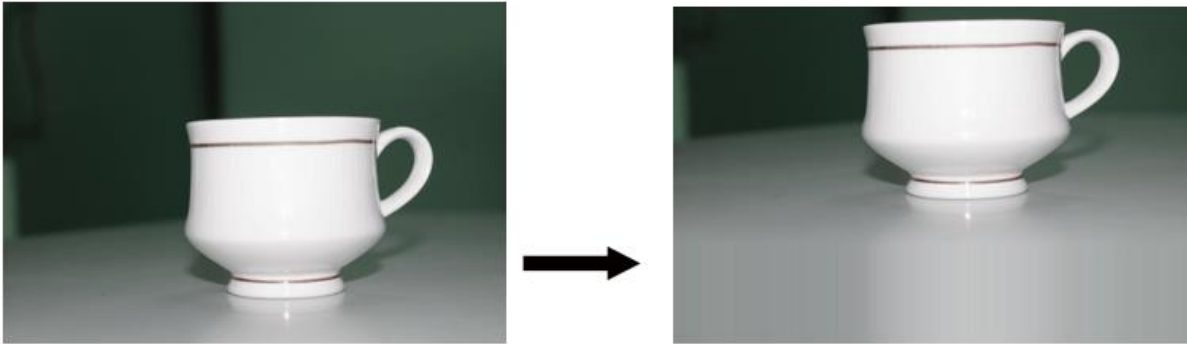
```
def remove_keypoints(folder_num, points):  
    lst = []  
    for x,y,z in points:  
        cond1 = x<250 and y>800  
        cond2 = x>1400 and y<400  
        if not (cond1 or cond2):  
            lst.append([x,y,z])  
    # print('Finished removing {} wrong keypoints....'.format(folder_num))  
    return lst
```

- 범위 밖의 keypoint 제거해 줌

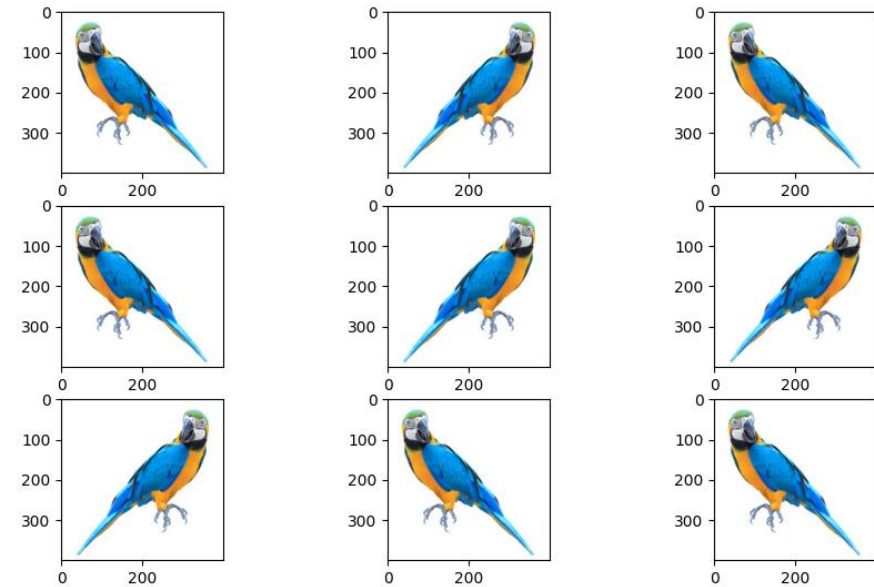
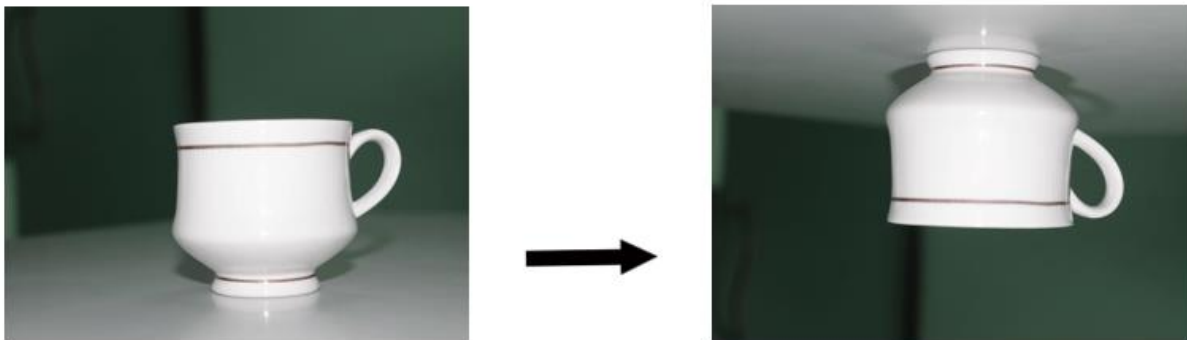
#2. 기본 Preprocessing

Flip Augmentation

Horizontal Flip Augmentation



Vertical Flip Augmentation



- 이미지 회전 등 변환하여 다양성 증가시키는 기법

<https://www.charterglobal.com/image-data-augmentation-and-ai/>
<https://www.studytonight.com/post/horizontal-and-vertical-flip-data-augmentation>

#2. 기본 Preprocessing

Flip Augmentation

```
# Flip Augmentation (Change target class)
if df_flip_info is not None:
    self.use_flip = True
    print('Use Flip Augmentation')
    left = label_encoder.transform(df_flip_info['left'])
    right = label_encoder.transform(df_flip_info['right'])
    left_to_right = dict(zip(left, right))
    right_to_left = dict(zip(right, left))

    self.flip_info = left_to_right.copy()
    self.flip_info.update(right_to_left)
    self.flip_possible_class = list(set(np.concatenate([left, right])))
self.flipaug_ratio = flipaug_ratio

print(f'Dataset size:{len(self.id)}')
```

- 중복 데이터 많음 → Overfitting → 방지하기 위해 Flip Augmentation 이용
- 왼손/오른손 class 다르므로 Horizontal Flip 수행 시 Class도 바꾸어 줌
- Flip augmentation 비율은 직접 해 본 결과 0.1~0.5 중 0.30이 가장 적합

#2. 기본 Preprocessing

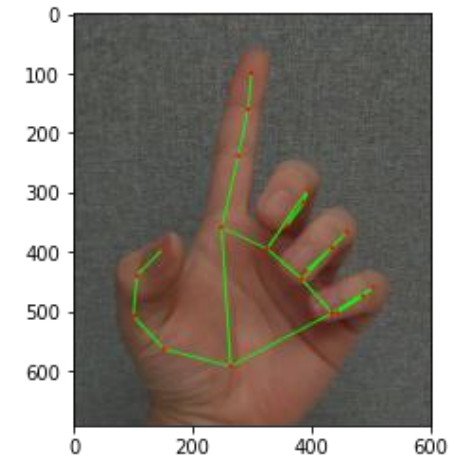
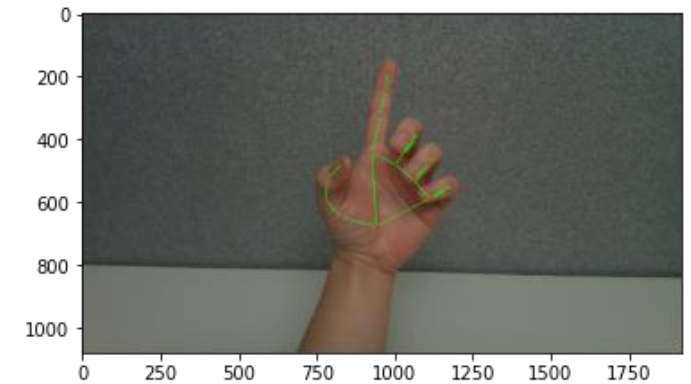
Crop using the Keypoints (Sample Code)

```
# annotation data의 max와 min point를 가져온 후 이미지 잘림 방식을 위해 각각의 크기에 100을 더해주고 빼줌
max_point = np.max(np.array(test_image_annotation), axis=0).astype(int) + 100
min_point = np.min(np.array(test_image_annotation), axis=0).astype(int) - 100
max_point = max_point[:-1] # remove Z order
min_point = min_point[:-1] # remove Z order

max_x, max_y = max_point
min_x, min_y = min_point

# 데이터 포인트의 크기가 원 이미지를 넘어서는 경우를 방지
max_x = max_x if max_x < 1920 else 1920
max_y = max_y if max_y < 1080 else 1080
min_x = min_x if min_x > 0 else 0
min_y = min_y if min_y > 0 else 0
```

- Keypoint가 있는 부분만 crop 해 줌 (object detection BBOX처럼)
- 크기에 100을 +-



#2. 기본 Preprocessing

Crop using the Keypoints (개선)

- Random Margin Crop Image

```
max_x, max_y = max_point
min_x, min_y = min_point
max_y += 100 # 손목부분까지 여유를 주기위해

# 매 에폭마다 Margin이 조금씩 다르게 들어가므로
if self.random_margin:
    if random.random() < 0.5:
        max_x += self.margin
    if random.random() < 0.5:
        max_y += self.margin
    if random.random() < 0.5:
        min_x -= self.margin
    if random.random() < 0.5:
        min_y -= self.margin
else:
    max_x += self.margin
    max_y += self.margin
    min_x -= self.margin
    min_y -= self.margin
```

- 손목 부분까지 여유를 주기 위해서 y에만 “기본적으로” 100을 더함
- 폴더 내 비슷한 이미지들의 overfitting을 방지하기 위해 margin을 1/2의 확률로 랜덤하게 주거나 주지 않음
- 여러 margin값으로 실험!
- 그리고 crop한 이미지에 대해서도 flip augmentation 해 줌

도전

- 양손일 경우 한 손씩 crop하여 이미지로 이용
→ 성능이 별로 안 좋아서 기각

#2. 기본 Preprocessing

Random Affine & Random Perspective Augmentation

Pytorch의 transform이 지원

Original image



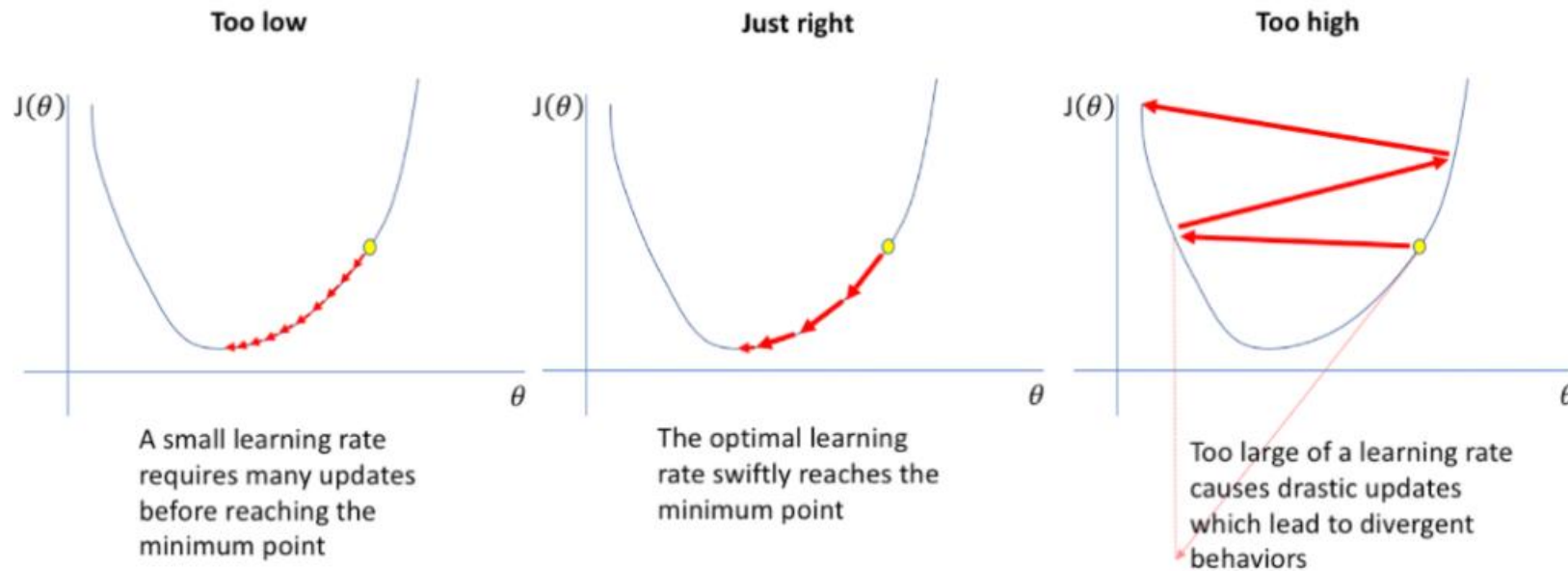
Original image



https://pytorch.org/vision/main/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

#2. 기본 Preprocessing

Learning Rate Scheduler

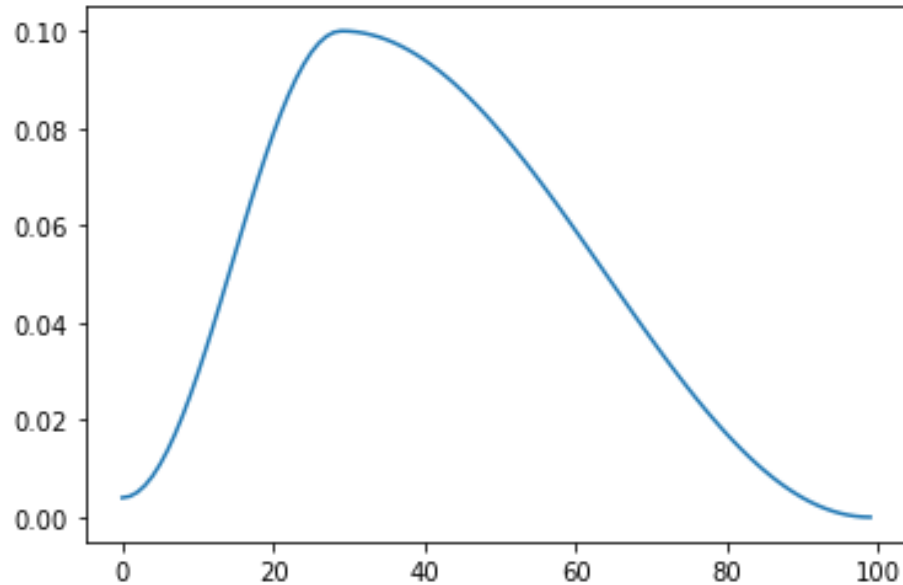


- 학습 때 사용되는 **learning rate**를 **변경**하기 위해 LR Scheduler를 사용함
- 학습이 진행됨에 따라 epoch 또는 iteration 간 learning rate을 조정
- 처음부터 끝까지 같은 **learning rate**를 사용하지 않고 학습 과정에서 **learning rate**를 조절

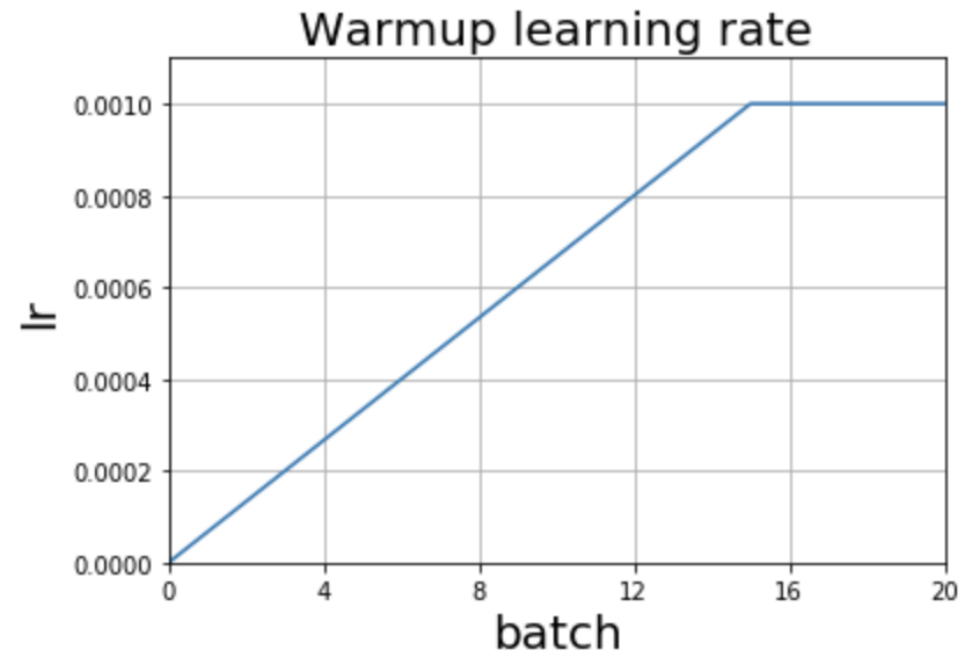
#2. 기본 Preprocessing

OneCycleLR Scheduler & WarmUpLR Scheduler

Pytorch가 제공하는 Learning rate scheduler



- 큰 값의 주기적 학습률(cyclical learning rates)을 적용: 학습 빠르고 정확함



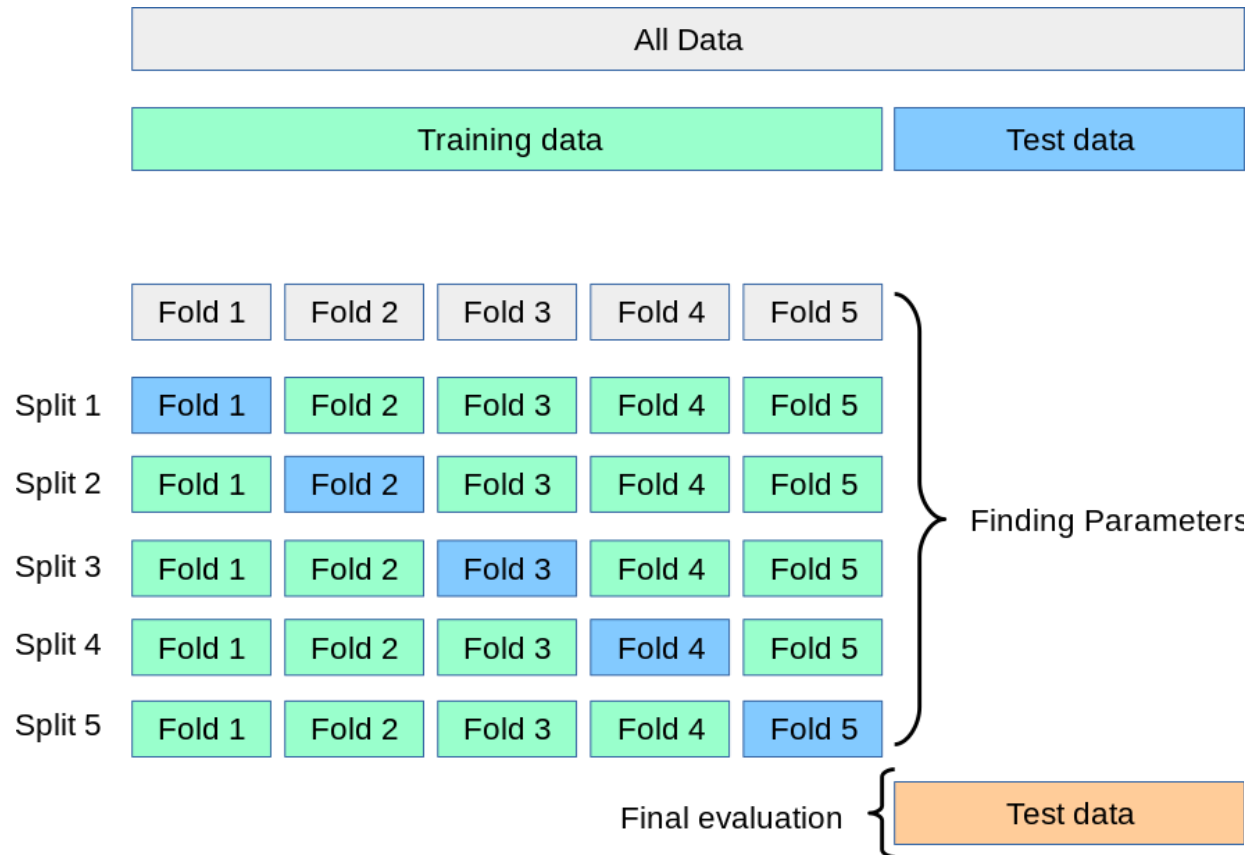
- 학습 초기 단계에서 변동성을 줄임

Cross Validation by K-fold

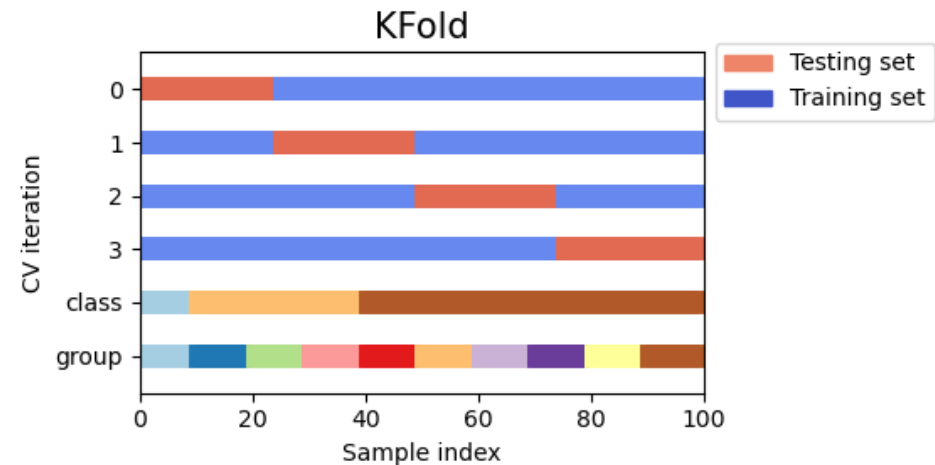


#3. Stratified GroupKFold Split

K-fold



- 과적합(Overfitting) 방지
- 학습 데이터에만 지나치게 최적화되는 것 방지
- 여러 학습 데이터 세트 / 검증 세트에서 학습 평가 수행

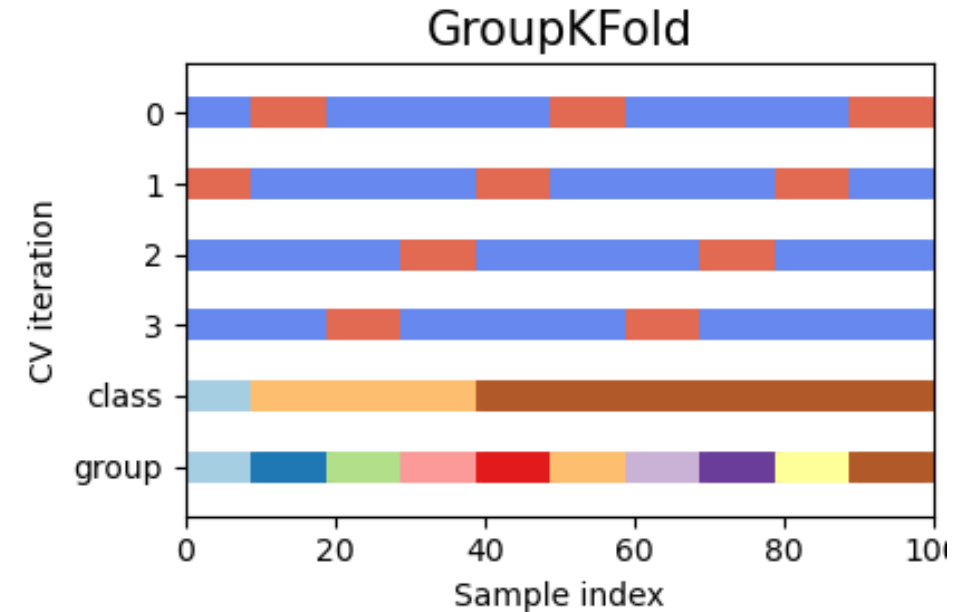
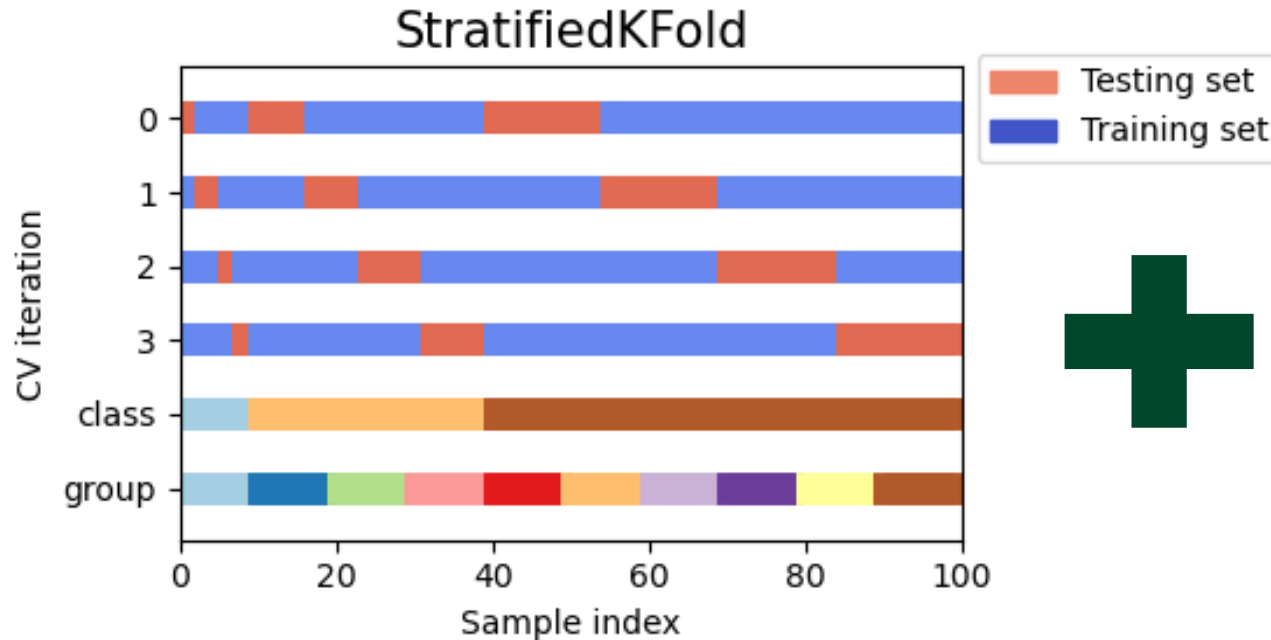


#3. 5Fold Ensemble

Stratified GroupKFold

Stratified GroupKFold

- ▶ 오류 심한 클래스에 대한 분석 수행
- ▶ Validation Loss 성능 안 좋음
- ▶ 클래스 비율 고려하여 섞어 주는 Stratified Kfold 이용

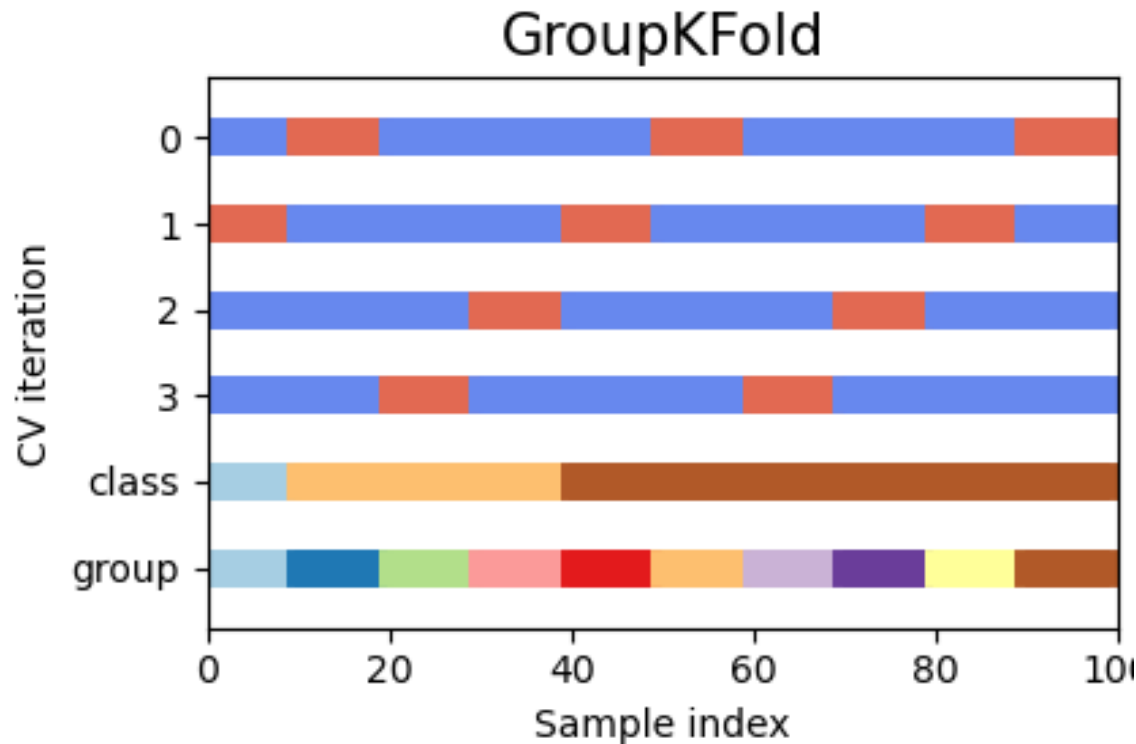


- K-fold의 변형된 방식
- 불균형 클래스의 사용이 추가됨

- Group K-fold는 동일 그룹이 Test & Valid Set에 동시에 포함되지 않도록 함

#3. 5Fold Ensemble

Stratified GroupKFold



- 데이터셋의 특성:
 - 이미지 5888개, 한 폴더 안(같은 index 안)의 이미지 비슷함
 - -> 일반적인 split 방법으로는 validation accuracy 1
- 각 폴더별 Group 부여를 통해 타파: Group 단위로 팀을 나누는 GroupKFold
- Class Label 분포 개선 위해 Stratified 함께 적용

BUT

- Split 할 때마다 점수 변동성이 큼
- 원인: 한두 개 있는 특정 클래스에 대해 검증 X
상황 발생: 방지하기 위해 Flip Augmentation

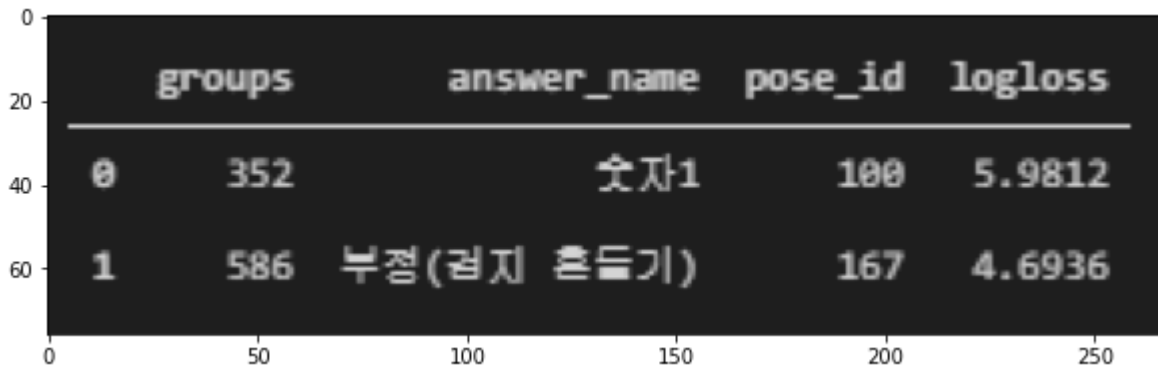
BUT

- 여전히.. Fold별 편차가 심함

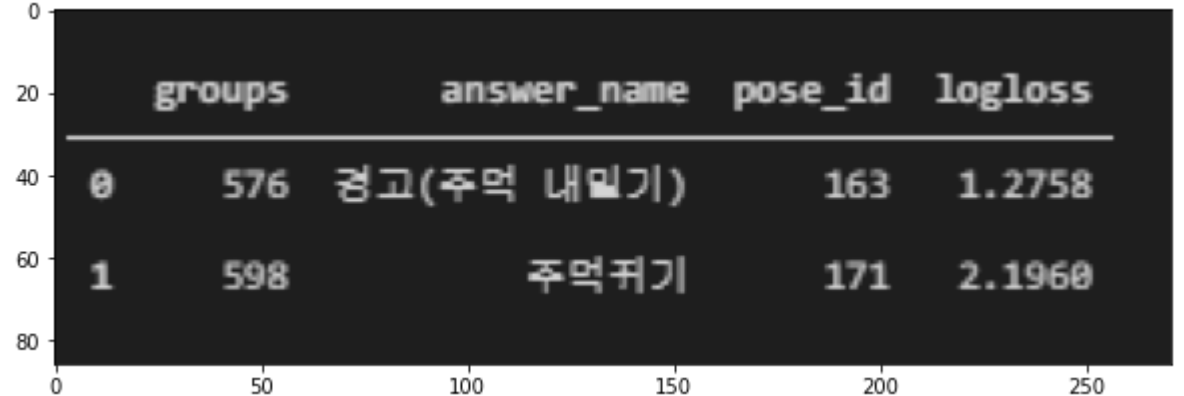
#3. Detail Image Preprocessing

왜 Fold별 점수 편차가 클까?

- 모델이 어떤 클래스를 못 맞추는지 분석하고자 함
- 특정 클래스들에서 LogLoss 매우 크게 발생 → 분석 결과: 숫자1, 검지 흔들기 / 주먹 내밀기, 주먹 쥐기



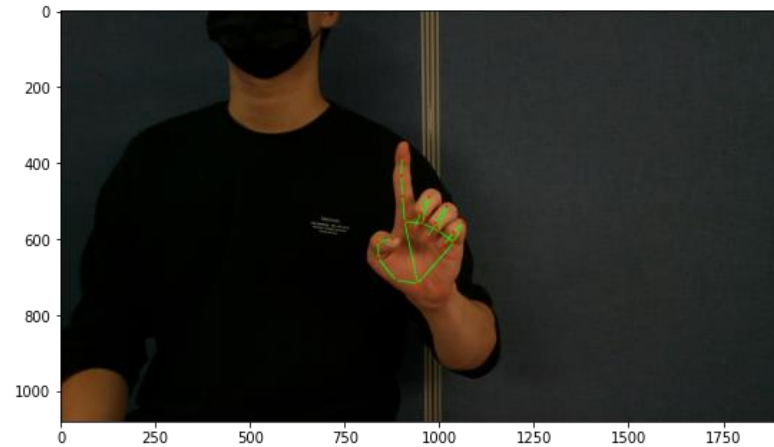
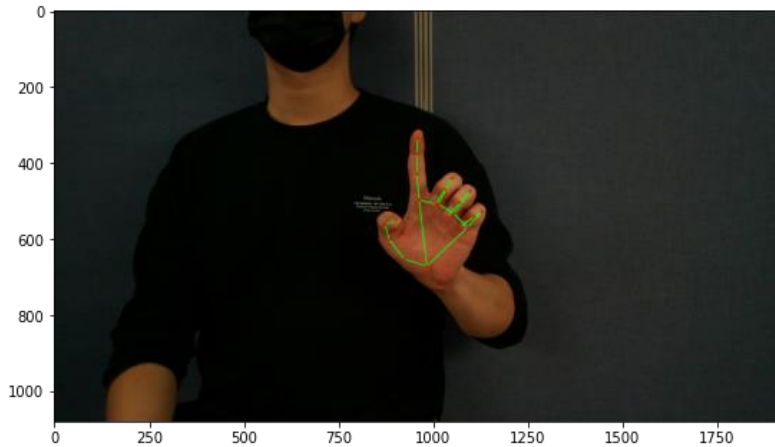
	groups	answer_name	pose_id	logloss
0	352	숫자1	100	5.9812
1	586	부정(검지 흔들기)	167	4.6936



	groups	answer_name	pose_id	logloss
0	576	경고(주먹 내밀기)	163	1.2758
1	598	주먹 쥐기	171	2.1960

#3. Detail Image Preprocessing

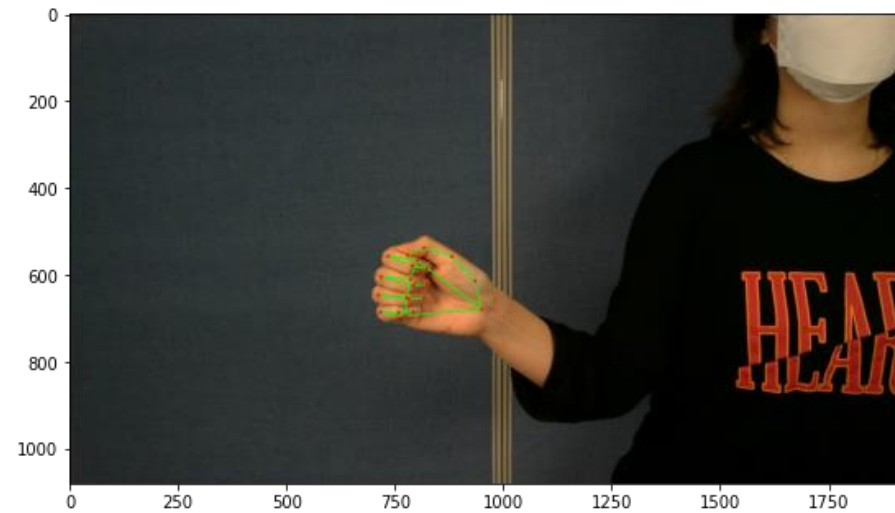
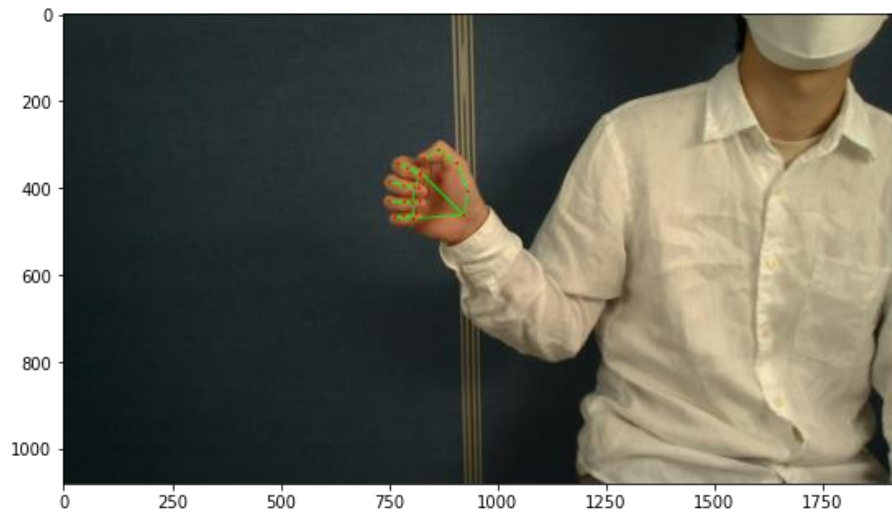
LogLoss > 1.2: 숫자 1, 검지 흔들기



- 이미지 분석 결과 이산적 사진으로 구분 불가능
- 폴더 내부에 있는 여러 사진들 모아 연속적으로 분석 -> 차이점 발견!
- 즉, 이미지 각각이 아닌 폴더 내 이미지 변화(손가락 움직임)으로 클래스 구별 가능

#3. Detail Image Preprocessing

LogLoss > 4.6: 주먹 내밀기, 주먹 쥐기

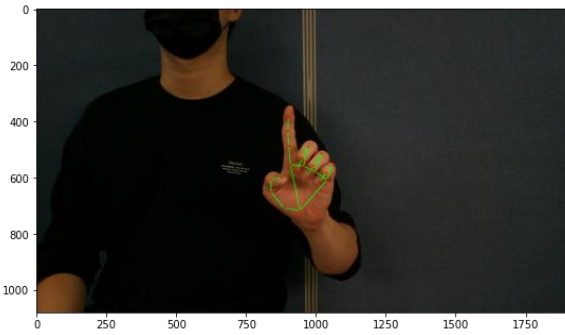
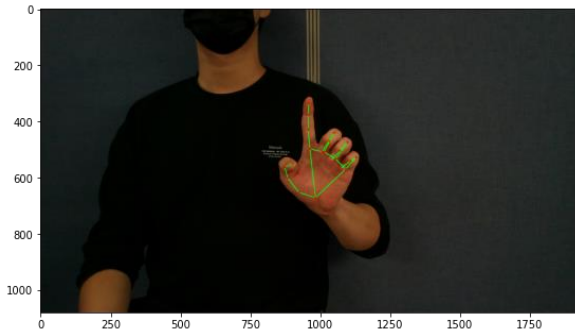


- 마찬가지로 이미지 분석 결과 이산적 사진으로 구분 불가능
- 폴더 내부에 있는 여러 사진들 모아 연속적으로 분석 -> 차이점 발견!
- 즉, 이미지 각각이 아닌 폴더 내 이미지 변화(손가락 움직임)으로 클래스 구별 가능
- ▶ 각 클래스를 구별하기 위해 추가적인 처리 필요함!

#3. Detail Image Preprocessing

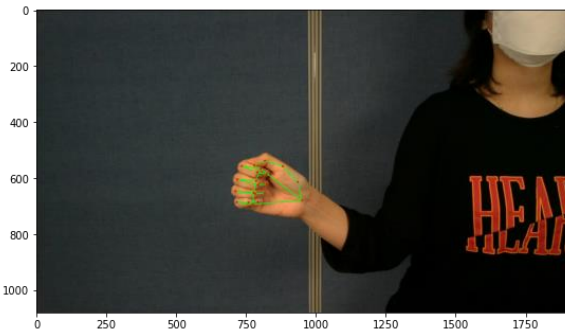
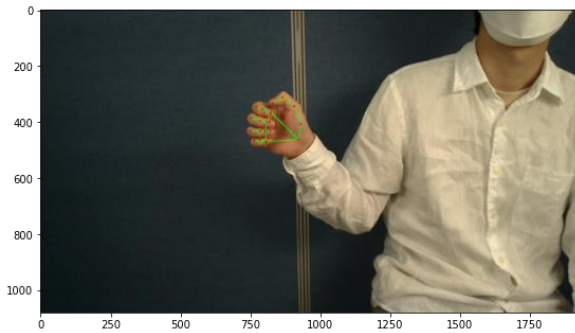
연속적 행동을 찾기 위한 Keypoint 알고리즘 구축

- 앞선 4가지 case의 모델로 예측이 된 경우 다시 구분하기 위해 keypoint 알고리즘을 구축함



숫자1과 검지 흔들기

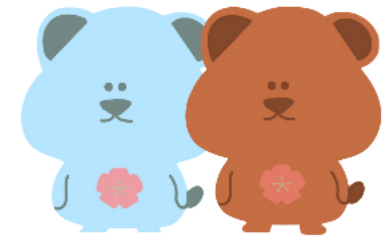
검지 흔들기: 숫자 1보다 검지 흔들는 것 검출됨
→ 검지 Keypoint X 좌표의 움직임 탐지
ex) 연속된 이미지 사이의 x 좌표들에 대한 max-min 계산. 적절한 threshold값 설정 + margin 줌



주먹 쥐기와 주먹 내밀기

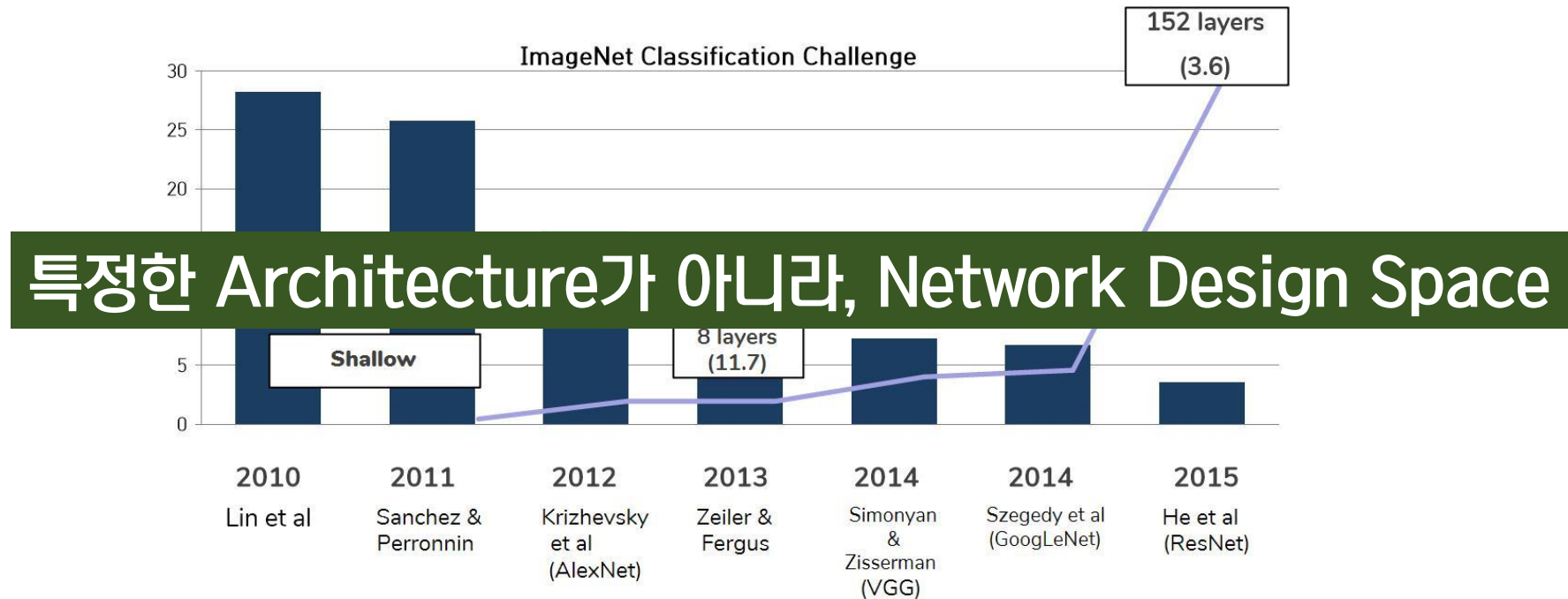
손의 움직임을 통해 구분 가능
ex) keypoint의 가장 오른쪽 x 좌표의 변화량 통해 max-min 계산.

Network Model: RegNet



#2. RegNet

Designing Network Design Spaces, FAIR, CVPR 2020

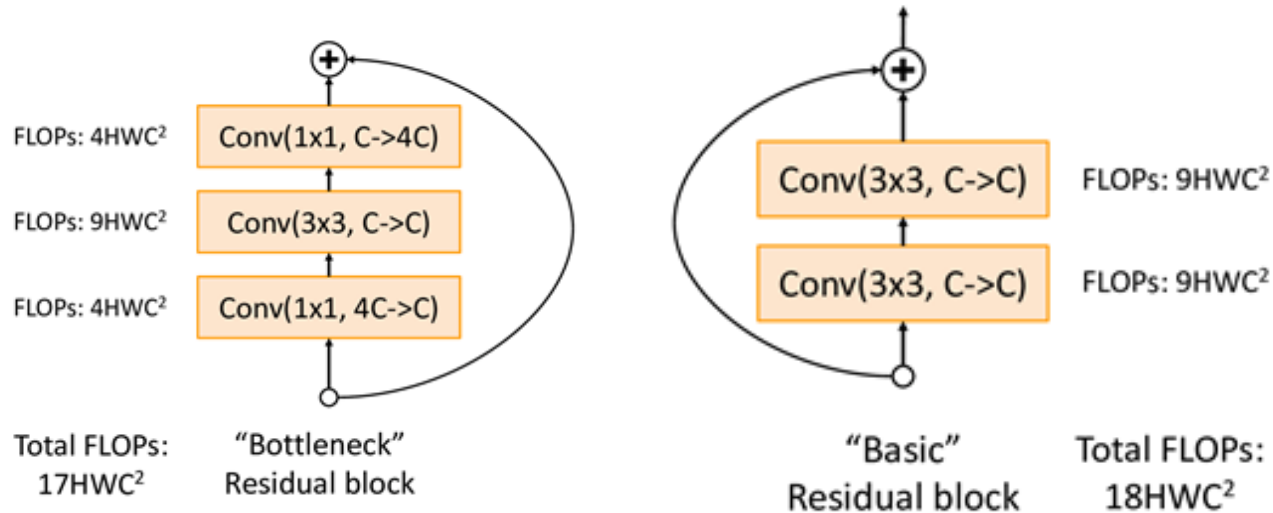


- 매우 유연한 network architecture 탐색 & 설계 가능
- Controlled by **setting the right parameters** in a quantized linear function **to determine width and depth of the network.**
- Network Design Spaces

#2. ResNet

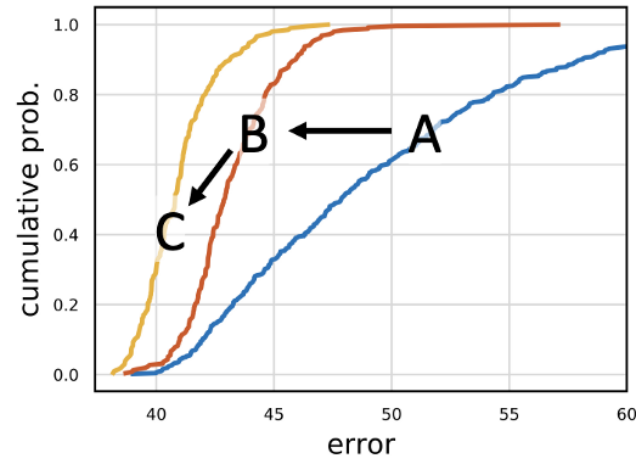
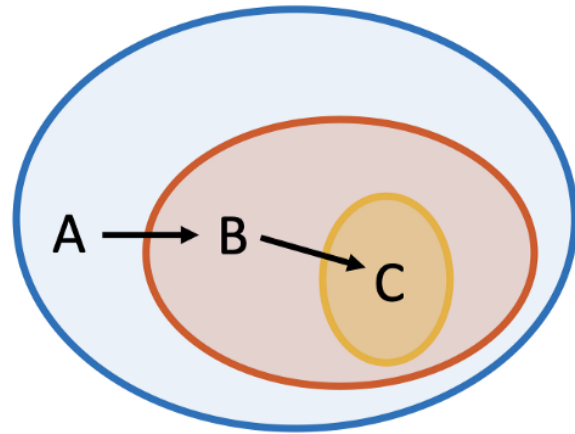
Network Design Space

- 다양한 architecture로 구성됨
- Network의 다양한 매개변수로 구성됨(width, depth, groups..)
- 한 가지 fixed된 network block만 선택해서 사용함 (e.g. the bottleneck block)



#2. RegNet

Network Design Space

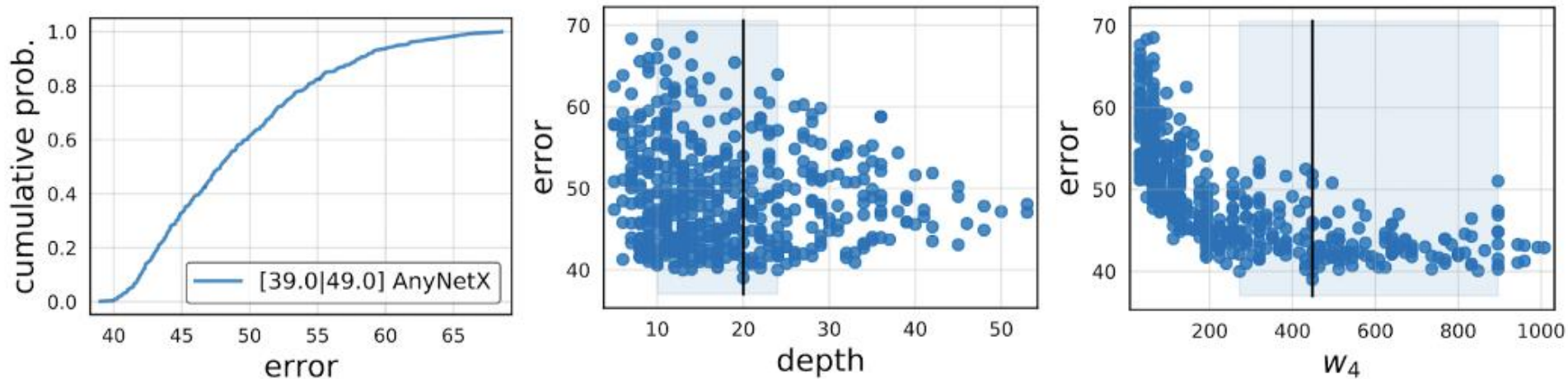


The space is constantly optimized to arrive at a smaller design space with the best models.

- 좋은 모델로 좁히기 위해 다양한 매개변수 실험
- 최종적으로 좋은 모델만 포함하는 RegNet Design Space 도출

#2. RegNet

AnyNet: a space of all possible models

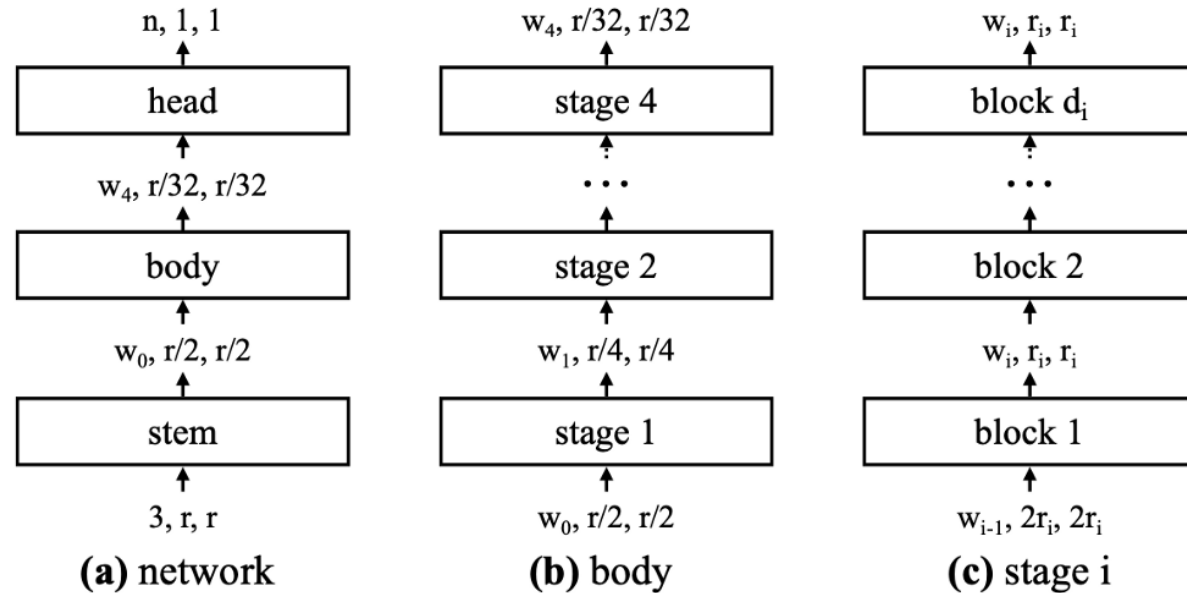


- 서로 다른 parameters 간의 가능한 모든 조합으로 이루어진 모든 종류의 모델을 포함
- 이 모든 모델들은 ImageNet dataset을 이용해 같은 환경에서(epochs, optimizer, weight decay, learning rate scheduler) trained & evaluated됨.

▶ 최고 성능 모델의 parameter 분석 → 초기 모델 생성 → 더 좋은, 좁은 모델로 좁혀 나감!

#2. RegNet

AnyNet Design Space



Net 구성: (a) stem, body, head

- 여러 단계가 정의되어 있고, 각 단계는 블록으로 구성됨 (블록 유형은 1가지)
- 모델의 설계는 width, depth 같은 고정 parameter에 의해 정의되지 않고, 선택된 parameter의 linear function으로 정의됨

#2. RegNet

	flops (B)	params (M)	acts (M)	infer (ms)	train (hr)	error (top-1)
RESNET-50	4.1	22.6	11.1	53	12.2	35.0 \pm 0.20
REGNETX-3.2GF	3.2	15.3	11.4	57	14.3	33.6 \pm 0.25
RESNEXT-50	4.2	25.0	14.4	78	18.0	33.5 \pm 0.10
RESNET-101	7.8	44.6	16.2	90	20.4	33.2 \pm 0.24
REGNETX-6.4GF	6.5	26.2	16.4	92	23.5	32.6 \pm 0.15
RESNEXT-101	8.0	44.2	21.2	137	31.8	32.1 \pm 0.30
RESNET-152	11.5	60.2	22.6	130	29.2	32.2 \pm 0.22
REGNETX-12GF	12.1	46.1	21.4	137	32.9	32.0 \pm 0.27

(a) Comparisons grouped by **activations**.

RESNET-50	4.1	22.6	11.1	53	12.2	35.0 \pm 0.20
RESNEXT-50	4.2	25.0	14.4	78	18.0	33.5 \pm 0.10
REGNETX-4.0GF	4.0	22.1	12.2	69	17.1	33.2 \pm 0.20
RESNET-101	7.8	44.6	16.2	90	20.4	33.2 \pm 0.24
RESNEXT-101	8.0	44.2	21.2	137	31.8	32.1 \pm 0.30
REGNETX-8.0GF	8.0	39.6	14.1	94	22.6	32.5 \pm 0.18
RESNET-152	11.5	60.2	22.6	130	29.2	32.2 \pm 0.22
RESNEXT-152	11.7	60.0	29.7	197	45.7	31.5 \pm 0.26
REGNETX-12GF	12.1	46.1	21.4	137	32.9	32.0 \pm 0.27

(b) Comparisons grouped by **flops**.

Table 5. **RESNE(X)T comparisons on ImageNetV2.**

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	top-1 error ours \pm std [orig]
EFFICIENTNET-B0	0.4	5.3	6.7	256	34	11.7	24.9 \pm 0.03 [23.7]
REGNETY-400MF	0.4	4.3	3.9	1024	19	5.1	25.9 \pm 0.16
EFFICIENTNET-B1	0.7	7.8	10.9	256	52	15.6	24.1 \pm 0.16 [21.2]
REGNETY-600MF	0.6	6.1	4.3	1024	19	5.2	24.5 \pm 0.07
EFFICIENTNET-B2	1.0	9.2	13.8	256	68	18.4	23.4 \pm 0.06 [20.2]
REGNETY-800MF	0.8	6.3	5.2	1024	22	6.0	23.7 \pm 0.03
EFFICIENTNET-B3	1.8	12.0	23.8	256	114	32.1	22.5 \pm 0.05 [18.9]
REGNETY-1.6GF	1.6	11.2	8.0	1024	39	10.1	22.0 \pm 0.08
EFFICIENTNET-B4	4.2	19.0	48.5	128	240	65.1	21.2 \pm 0.06 [17.4]
REGNETY-4.0GF	4.0	20.6	12.3	512	68	16.8	20.6 \pm 0.08
EFFICIENTNET-B5	9.9	30.0	98.9	64	504	135.1	21.5 \pm 0.11 [16.7]
REGNETY-8.0GF	8.0	39.2	18.0	512	113	28.1	20.1 \pm 0.09

Table 4. **EFFICIENTNET comparisons** using our standard train-

#2. Pytorch for RegNet

```
else: # Single Best Model (Using the pretrained weight)
    model_path = './results/single_best_model.pth'
    single_best = Pose_Network(args).to(device)
    single_best.load_state_dict(torch.load(model_path)['state_dict'])
    single_best.eval()
    model_list = [single_best]
```

```
class Pose_Network(nn.Module):
    def __init__(self, args):
        super().__init__()
        self.encoder = timm.create_model(args.encoder_name, pretrained=True,
                                         drop_path_rate=args.drop_path_rate,
                                         )
        num_head = self.encoder.head.fc.in_features
        self.encoder.head.fc = nn.Linear(num_head, 157)

    def forward(self, x):
        return self.encoder(x)
```

```
args = easydict.EasyDict({'encoder_name':'regnety_040',
                          'drop_path_rate':0,
                          })
```

#2. Pytorch for RegNet

torchvision.models

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet
- EfficientNet
- RegNet
- VisionTransformer
- ConvNeXt

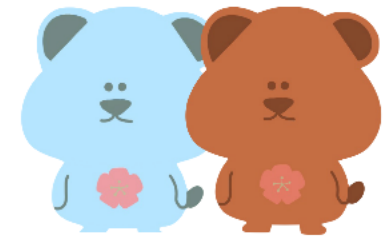
torchvision.models.regnet

```
__all__ = [  
    "RegNet",  
    "regnet_y_400mf",  
    "regnet_y_800mf",  
    "regnet_y_1_6gf",  
    "regnet_y_3_2gf",  
    "regnet_y_8gf",  
    "regnet_y_16gf",  
    "regnet_y_32gf",  
    "regnet_y_128gf",  
    "regnet_x_400mf",  
    "regnet_x_800mf",  
    "regnet_x_1_6gf",  
    "regnet_x_3_2gf",  
    "regnet_x_8gf",  
    "regnet_x_16gf",  
    "regnet_x_32gf",  
]
```











Model	Params (M)	GFLOPs	Batch size	Top-1 acc (%) (our impl.)	Top-1 acc (%) (official)
RegNetX-200M	2.685	0.199	1024	68.210	68.9
RegNetX-400M	5.158	0.398	1024	72.278	72.7
RegNetX-600M	6.196	0.601	1024	73.862	74.1
RegNetX-800M	7.260	0.800	1024	74.940	75.2
RegNetX-1.6G	9.190	1.603	1024	76.706	77.0
RegNetX-3.2G	15.296	3.177	512	78.188	78.3
RegNetX-4.0G	22.118	3.965	512	78.690	78.6
RegNetX-6.4G	26.209	6.460	512	79.152	79.2
RegNetX-8.0G	39.573	7.995	512	79.380	79.3
RegNetX-12G	46.106	12.087	256	79.998	79.7
RegNetX-16G	54.279	15.941	256	80.118	80.0
RegNetX-32G	107.812	31.736	256	80.516	80.5

https://pytorch.org/vision/main/_modules/torchvision/models/regnet.html

Conclusion



차별점

1	FDAI	   	0.00578	67	5달 전
2		  	0.09171	57	5달 전
3			0.09505	48	5달 전
4			0.10767	25	5달 전
5			0.12653	23	5달 전

차별점

- 데이터 Preprocessing

- 데이터를 직접 보고 분석하고 살펴보는 노력
- Crop Using the Keypoints: 475, 543

- 피드백 & 결함 보충

- Cross Validation 이후 끊임없는 피드백 노력

- 다양한 method 사용

- Learning Rate Scheduler
- keypoint 탐지 알고리즘 직접 구현 (데이터 분석을 곁들인)

- 모델: RegNet 사용

임의로 특정 task에서 유명한 모델을 사용해 보며 모델을 고르고 발전시키는 기존의 방식과 달리, RegNet을 이용하여 자체적으로 해당 dataset에 적합한 모델 architecture를 찾음

THANK YOU

