LNRNet: A new lightweight and noise-resilient network for on-device machine fault diagnosis

2023/08/03

이성재

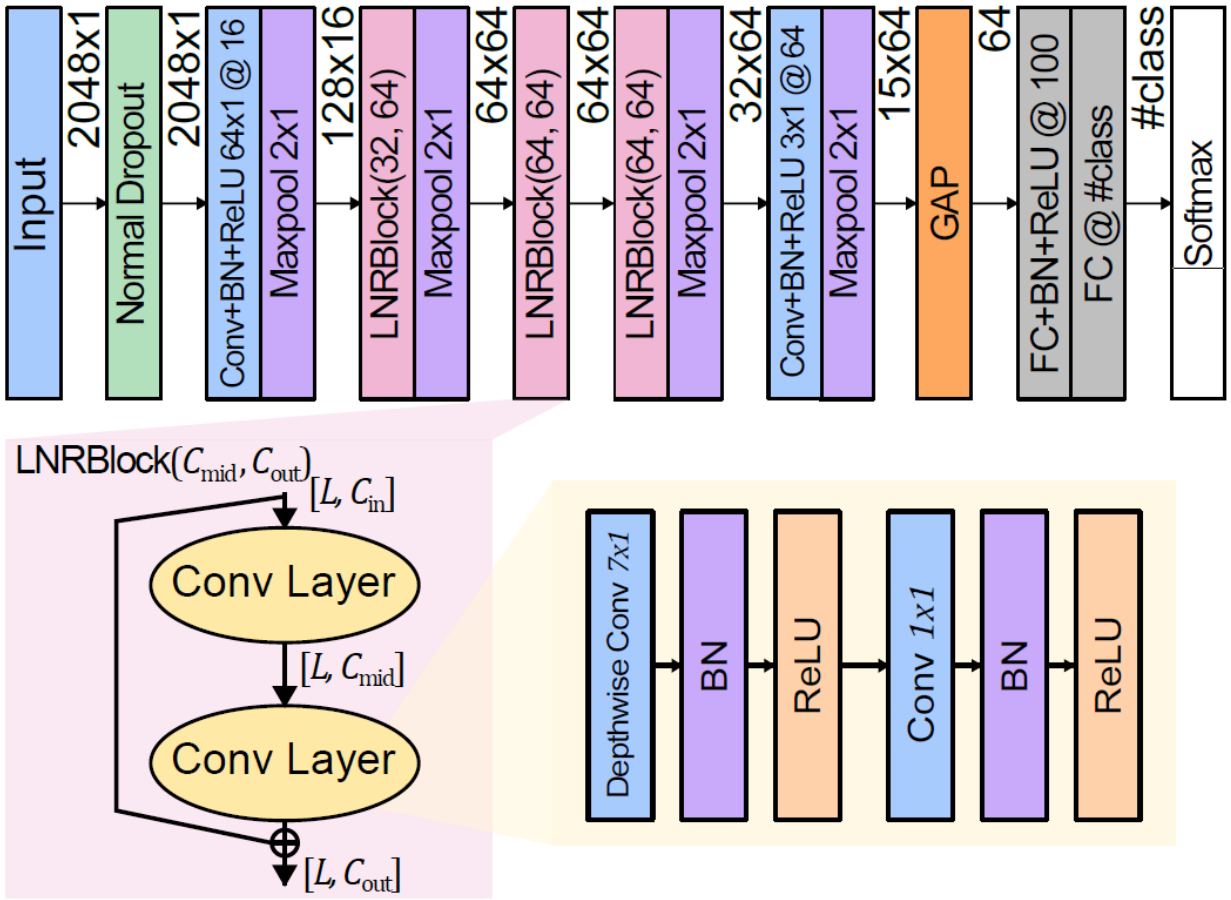
*주의사항: 오늘 발표하는 내용은 현재 제출 준비중인 논문의 내용입니다.
논문이 publish되기 전까지는 자료와 소스코드를 외부로 공유하지 말아주세요.

연구 개요

- 최근 기계 결함 진단 분야에서 가장 많이 사용되는 방법은 data-driven 딥러닝 기반 접근 방법임. 딥러닝 모델은 일반적으로 연산량과 메모리 요구량이 커서 클라우드에서 추론을 수행해야 함. 그러나 클라우드와 필드 엣지 디바이스 간 데이터 교환 과정에서 시간 지연과 보안 문제가 발생할 가능성이 있음.
- 이 문제를 해결하기 위해 엣지 디바이스에서 모델 추론을 수행하는 온디바이스 AI가 많은 관심을 받고 있음. 그러나 엣지 디바이스에서 모델 추론을 수행하려면 경량 모델을 사용해야 한다는 제약조건이 있음. 이 제약조건은 산업현장에서 쉽게 발생하는 노이즈 있는 신호에 강건한 모델을 만드는 데 큰 어려움을 발생시킴.
- 이 연구는 진동 신호를 입력으로 받아 엣지 디바이스에서 추론을 수행할 수 있는 경량 구조와 노이즈 강건성을 모두 가진 기계 고장 진단 CNN 모델 lightweight and noise-resilient network (LNRNet)을 제시하고, 그 성능을 평가했음.
- 본 세미나에서는 논문에서 다루는 전체 내용 중 **(1) 모델 설계 아이디어 및 구현; (2) 양자화 아이디어** 두 가지에 대해 집중적으로 다룸.

모델 구조

- LNRNet은 연산량과 파라미터 수를 최소화하고자 전체 10층의 구조를 가지고 있는 1D CNN 모델로 설계되었음.

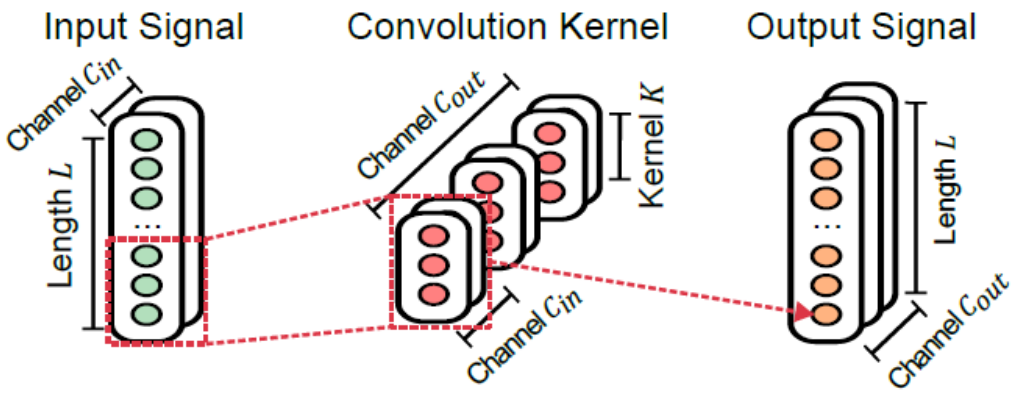


모델 설계 아이디어

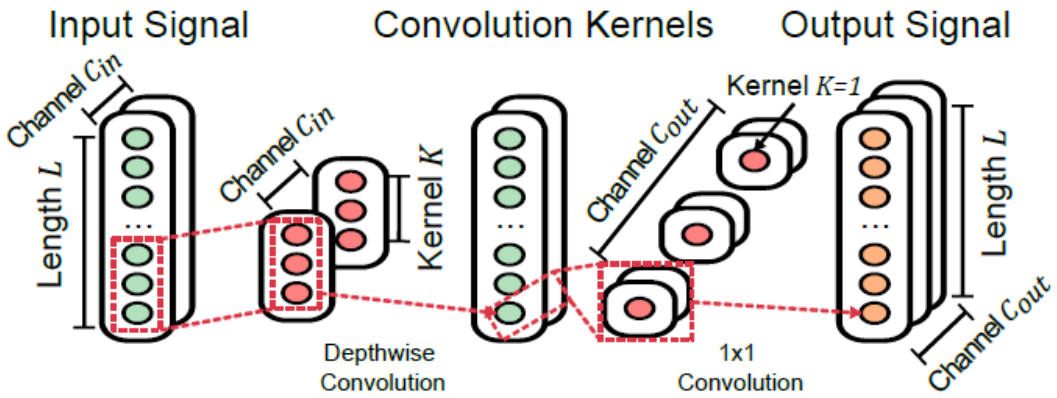
- Wide-kernel depthwise-separable convolution (DSConv)
  - 일반적인 이미지 분류나 고장 진단을 수행하는 CNN의 커널 크기는 3임. WDCNN (Zhang et al. 2017)과 같은 모델은 첫 convolutonal (conv) layer의 커널 크기를 64로 하기도 했으나 그 다음 층 conv layer의 커널 크기는 3으로 고정되어 있음.
  - 그러나 MnasNet (Tan et. al 2019)이나 ConvNext (Liu et al. 2022)와 같은 현대 CNN 모델들은 conv layer의 커널 크기를 키워서 receptive field를 확장하여 모델 성능을 향상시키려는 시도를 하고 있음.
  - 그러나 커널 크기를 아무 조치 없이 키우면 conv layer의 연산량이 커널 크기에 비례해서 증가하므로 모델을 온디바이스 AI에 적용할 수 없다는 문제가 있음.
  - 따라서 LNRNet은 DSConv와 커널 크기 7을 동시에 사용하여 모델 성능을 높이는 동시에 연산량과 파라미터 증가량은 최소화하고자 했음.

모델 설계 아이디어

- Wide-kernel depthwise-separable convolution (DSConv)



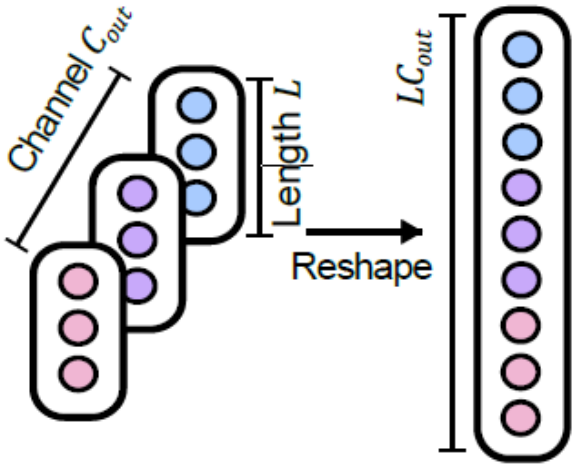
(a) Normal convolution



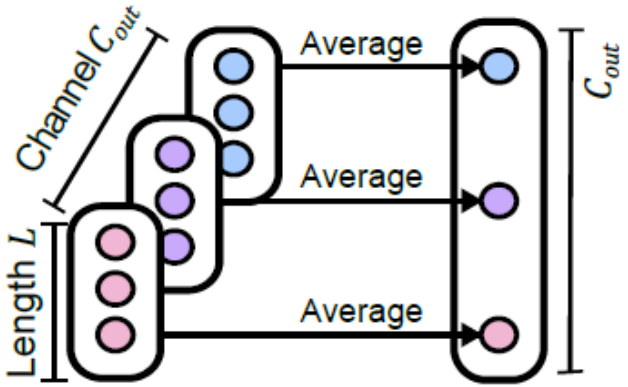
(b) DSConv

모델 설계 아이디어

- Global average pooling (GAP)
  - conv layer의 최종 출력을 fully connected (FC) layer로 전달하는 가장 간단한 방법은 출력을 1차원으로 reshape하는 방법임.
  - 그러나 이 방법을 사용하면 입력 크기에 따라 파라미터 수가 큰 폭으로 변화하는 FC layer의 특성상 모델의 파라미터 수가 크게 증가할 수밖에 없음.
  - 따라서 LNRNet은 GAP (Lin et al. 2013)을 사용하여 모델의 파라미터를 최소화하고자 함.



(a) Flattening



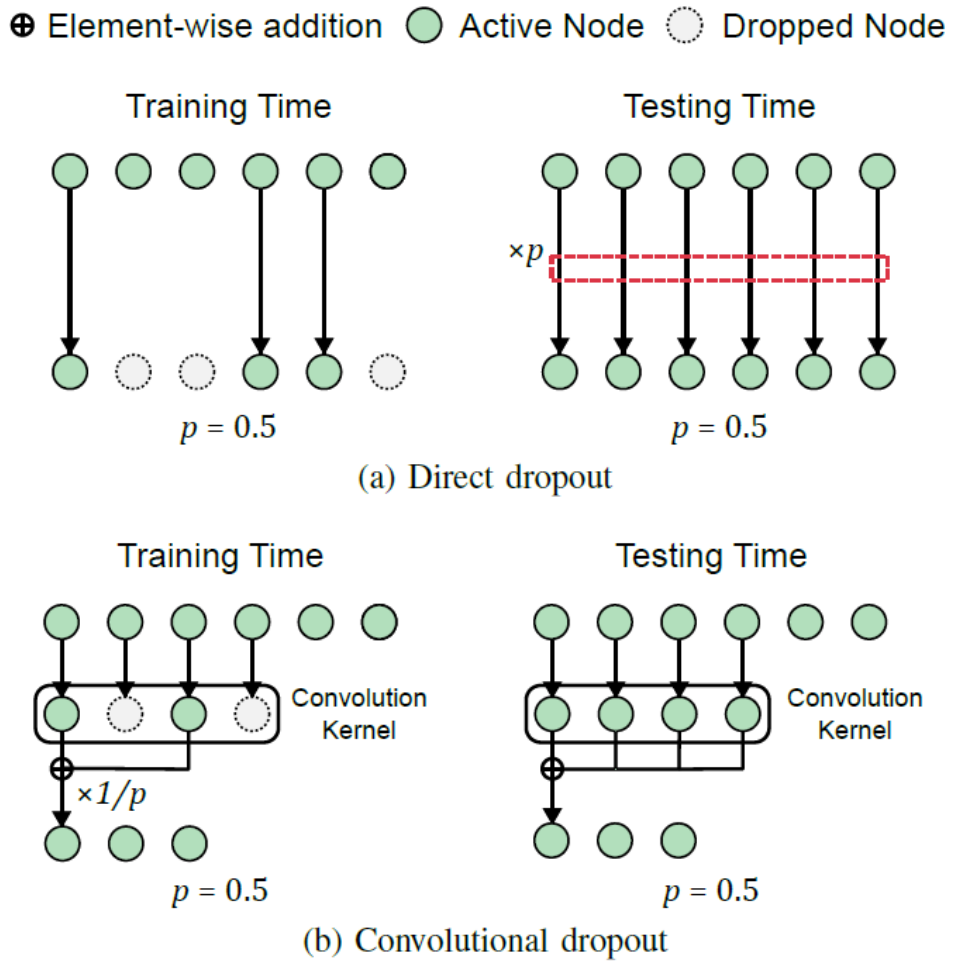
(b) GAP

모델 설계 아이디어

- A direct dropout strategy
  - Dropout (Srivastava et al. 2014)은 원래 artificial neural network (ANN) 모델의 훈련 과정에서 뉴런의 일부를 비활성화하여 오버피팅을 방지하고자 하는 목적으로 제시되었음.
  - LNRNet은 dropout에서 “뉴런의 일부를 무작위로 비활성화한다”는 점에서 착안하여 input data에 직접적으로 dropout을 적용해서 input data에 masking을 가하는 방법을 사용했음. 이렇게 훈련을 진행하면 모델이 학습 과정에서 조작된 입력 데이터 (즉, 노이즈 있는 데이터라고도 볼 수 있음)에 대한 학습을 잘 할 수 있게 되서 모델의 노이즈 강건성이 향상될 것이라고 봄.
  - 고장 진단 분야에서 (Zhang et al. 2018)는 훈련 과정에서 첫 conv kernel의 일부를 비활성화하여 모델의 잡음 강건성을 높이고자 하는 시도를 했음. 그러나 이 방법은 간접적으로 입력에 간섭을 주는 방법으로 direct dropout보다는 모델의 노이즈 강건성을 덜 올릴 것이라고 판단했음.

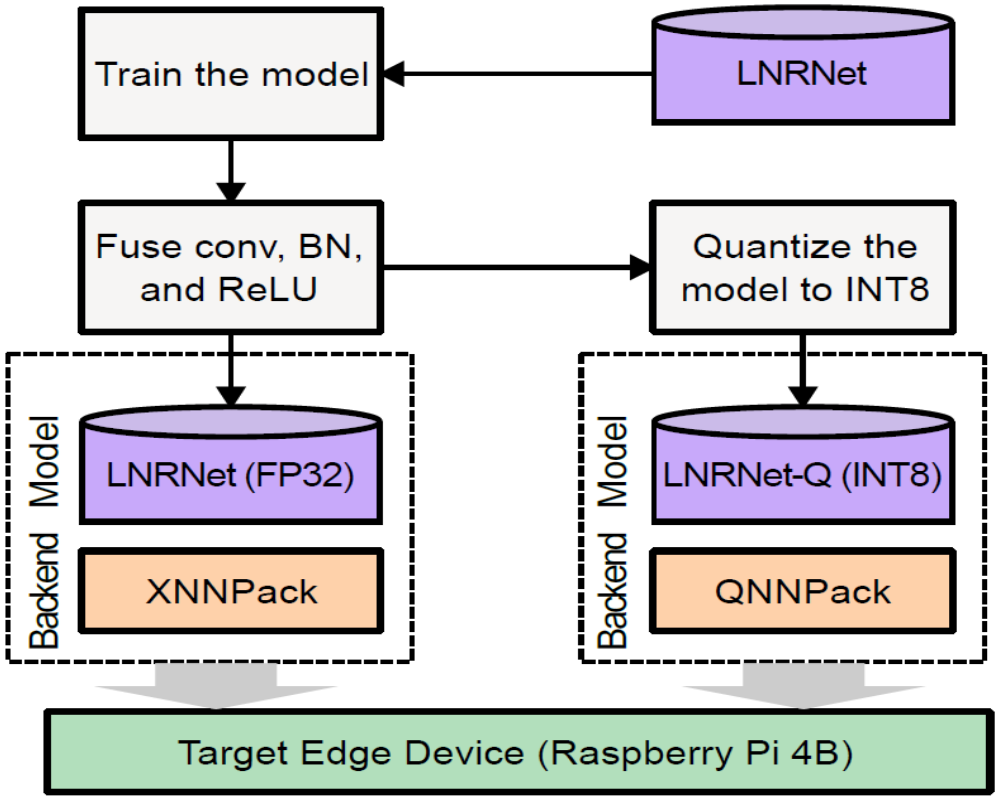
모델 설계 아이디어

- A direct dropout strategy



양자화 아이디어

- 모델 최적화와 양자화
  - 본 연구에서는 설계된 모델을 추가적으로 최적화하고 양자화했음.
  - 양자화에 대한 이해의 시작점으로 논문 (Wu et al. 2020)을 읽어보는 것을 추천함.



양자화 아이디어

- 모델 최적화
  - 모델을 양자화하기 전 Conv-batch normalization (BN)-rectified linear unit (ReLU) 연산 3개를 1개로 통합함.
  - BN은 affine 변환이므로 conv layer의 weight와 bias를 다음과 같이 바꿔서 BN을 conv 연산에 합칠 수 있음.

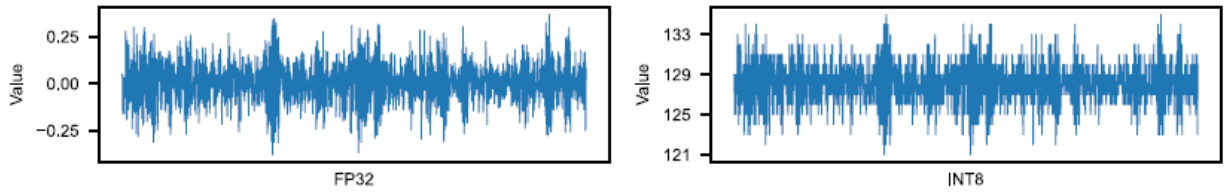
$$W'_{m,n,j} = \frac{\gamma_j}{\sqrt{\sigma_j^2 + \epsilon}} W_{m,n,j} \tag{12}$$

$$b'_j = \frac{\gamma_j}{\sqrt{\sigma_j^2 + \epsilon}} (b_j - \mu_j) + \beta_j \tag{13}$$

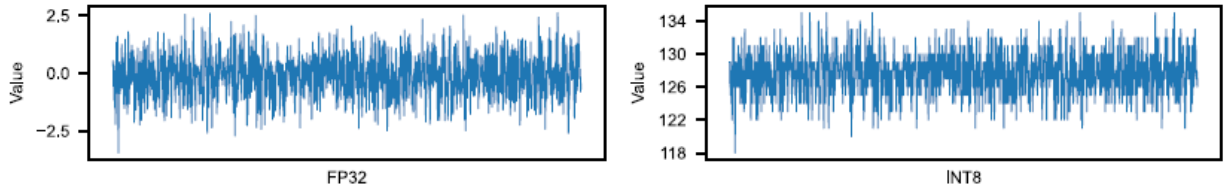
  - 그리고 conv 연산과 ReLU 함수는 PyTorch 프레임워크에서 합칠 수 있는 기능을 제공함.

양자화 아이디어

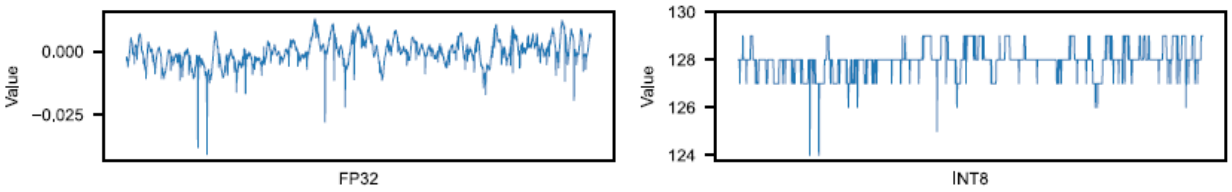
- 모델 양자화와 해결해야 할 문제점
  - 기계 고장 진단 태스크에서 양자화된 모델은 stable signal (변화가 거의 없는 신호)을 판별하는 데 가장 큰 어려움을 겪음 (그림의 (c)에서 발생하는 양자화 오차를 참고)



(a) CWRU dataset



(b) MFPT dataset



(c) Ottawa dataset

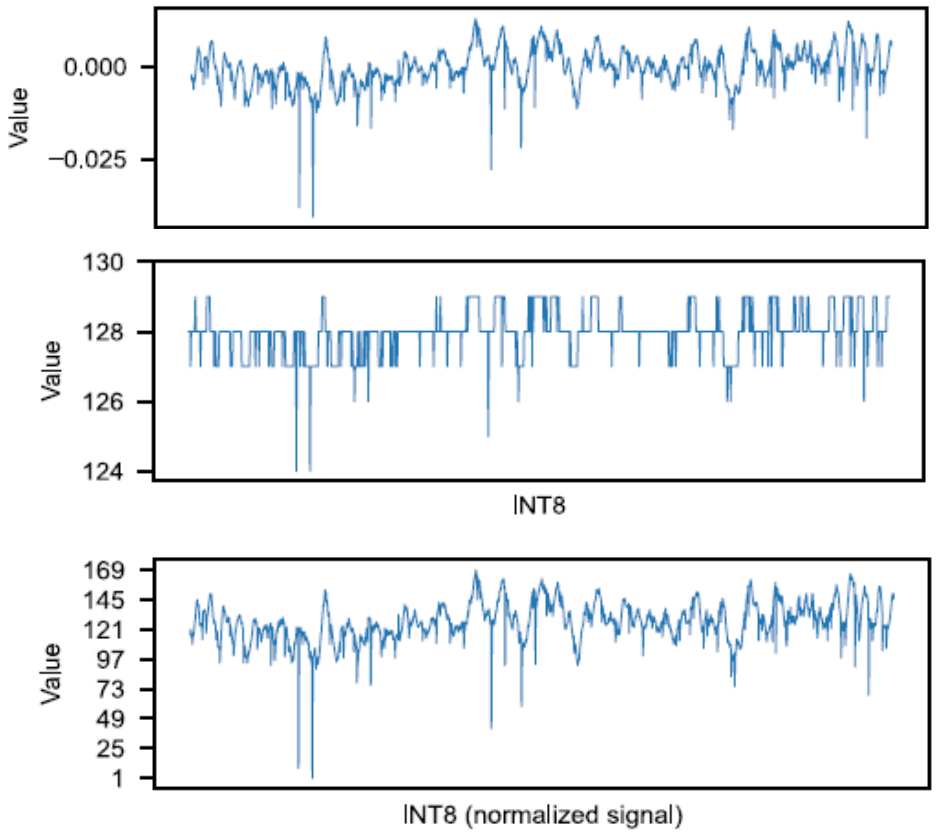
양자화 아이디어

- 모델 양자화와 해결해야 할 문제점
  - 이 문제를 해결하기 위해서 모든 샘플을 샘플의 절댓값의 최댓값으로 나눠 범위를 [-1, 1]로 스케일링하는 L-infinity normalization을 사용함.

$$||\boldsymbol{x}||_\infty = \max_i(|x_i|)$$

$$y_i = \frac{x_i}{||\boldsymbol{x}||_\infty}$$

L-infinity normalization



모델 설계 아이디어 검증

- CWRU (Case Western Reserve University), MFPT (Society of Machine Failure Prevention Technology), Ottawa (Huang and Baddour 2018) 데이터셋에서 노이즈 유/무 시 모델 설계 아이디어가 성능에 영향을 어떻게 미치는지 검증
  - 회색 음영이 LNRNet의 옵션
  - # Params: 파라미터 수, # MAdds: multiply-add operations 수

Model Configuration	Value	Accuracy(Noise-Free, %)			Accuracy (0dB Noise, %)			#Params (K)	# MAdds (M)
		CWRU	MFPT	Ottawa	CWRU	MFPT	Ottawa		
Kernel Size	3	99.73±0.09	99.99±0.01	99.09±0.08	89.79±0.75	81.24±0.67	92.68±1.21	43.09	2.31
	5	99.79±0.12	99.98±0.02	99.37±0.05	90.53±0.84	83.36±0.95	94.25±1.04	43.69	2.36
	7	99.80±0.05	99.99±0.01	99.50±0.03	92.61±0.72	90.47±1.08	95.42±0.60	44.30	2.40
Convolution	Normal	99.81±0.07	100±0.0	99.62±0.03	89.91±0.71	82.78±0.50	94.29±0.88	154.93	10.41
	DSConv	99.80±0.05	99.99±0.01	99.50±0.03	92.61±0.72	90.47±1.08	95.42±0.60	44.30	2.40
Vectorization	Flattening	99.50±0.18	99.96±0.03	99.37±0.05	91.56±1.11	91.33±1.21	95.32±0.44	133.90	2.49
	GAP	99.80±0.05	99.99±0.01	99.50±0.03	92.61±0.72	90.47±1.08	95.42±0.60	44.30	2.40
Dropout	No Dropout	99.76±0.10	99.99±0.01	99.77±0.02	82.22±1.70	84.16±1.74	67.05±3.37	44.30	2.40
	Conv Dropout	99.61±0.11	99.98±0.03	99.58±0.04	89.46±0.98	89.36±1.32	90.90±1.26		
	Direct Dropout	99.80±0.05	99.99±0.01	99.50±0.03	92.61±0.72	90.47±1.08	95.42±0.60		

양자화 성능 검증

- Normalization 사용에 따라 양자화 시 발생하는 성능 변화 차이 확인
  - 회색 음영이 최종 옵션
  - LNRNet: 32bit float 모델, LNRNet-Q: 8bit int 모델

Normalization	Model	Accuacy(Noise-Free, %)			Accuracy (0dB Noise, %)		
		CWRU	MFPT	Ottawa	CWRU	MFPT	Ottawa
✘	LNRNet	99.17± 0.11	99.93± 0.04	98.90± 0.13	90.10± 2.14	89.34± 2.18	94.01± 0.92
✘	LNRNet-Q	98.26± 0.45	97.93± 0.66	80.85± 3.47	89.76± 2.18	89.31± 2.21	76.93± 2.88
✓	LNRNet	99.80± 0.05	99.99± 0.01	99.50± 0.03	92.61± 0.72	90.47± 1.08	95.42± 0.60
✓	LNRNet-Q	**99.24± 0.24**	**99.7± 0.34**	**99.41± 0.06**	**92.11± 1.02**	**90.51± 1.51**	**95.02± 0.85**

LNRNet 소스코드 분석

소스코드 분석

- 포인트 1: 모듈화
  - 모듈화할 수 있는 부분은 별도 클래스로 모듈화하기

```
class ConvBnActivation(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, activation, depthwise):
        super(ConvBnActivation, self).__init__()
        self.activation = activation
        self.bn = nn.BatchNorm1d(out_channels)
        self.depthwise = depthwise
        padding_size = int((kernel_size-1)/(2))
        if depthwise:
            self.conv = torch.nn.Sequential(
                torch.nn.Conv1d(in_channels, in_channels, kernel_size, stride=1,
                                padding=padding_size, groups=in_channels),
                torch.nn.BatchNorm1d(in_channels),
                self.activation,
                torch.nn.Conv1d(in_channels, out_channels, 1, stride=1, padding=0)
            )
        else:
            self.conv = torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1,
padding=padding_size)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.activation(x)

        return x
```

소스코드 분석

- 포인트 2: 모델의 동적 생성
  - 빠른 성능 평가를 위하여 모델의 속성을 인자로 받아 단일 개체에서 다양한 모델을 만들어 실험할 수 있도록 하기

```
class LNRNetBase(nn.Module):
    def __init__(self, n_classes: int=10, device: str=None, kernel_size=3, activation=nn.ReLU(),
                 depthwise=False, residual_connection=False, gap=False, first_layer="nodrop"):
        super(LNRNetBase, self).__init__()

        self.dropout_rate = 0.5
        self.cdrop = False

        if first_layer == "cdrop":
            self.first_conv = Conv1dDropout(1, 16, 64, stride=8, padding=28, device=device)
            self.cdrop = True
        elif first_layer == "drop":
            self.first_conv = torch.nn.Sequential(
                torch.nn.Dropout(0.5),
                torch.nn.Conv1d(1, 16, 64, stride=8, padding=28)
            )
        else:
            self.first_conv = torch.nn.Conv1d(1, 16, 64, stride=8, padding=28)
...
...
...
```

소스코드 분석

- 포인트 3: 상속
  - PyTorch의 특정한 모듈 (예: nn.Conv1d)를 기반으로 하지만, 일부 기능만 다르게 만들고 싶은 경우 nn.Conv1d를 상속받아서 필요한 부분만 고치기

```python
class Conv1dDropout(nn.Conv1d):
    def __init__(
        self,
        in_channels: int,
        out_channels: int,
        kernel_size: int,
        stride: int,
        padding: Union[str, Tuple[int, ...]],
        device=None,
        dtype=None,
    ) -> None:
        super(Conv1dDropout, self).__init__(
            in_channels,
            out_channels,
            kernel_size,
            stride,
            padding,
            bias=False,
            device=device,
            dtype=dtype,
        )
        self._factory_kwargs = {"device": device, "dtype": dtype}
        self._mask = torch.zeros(
            (out_channels, in_channels, kernel_size), **self._factory_kwargs
        )
```

참고문헌

(Huang and Baddour 2018) H. Huang and N. Baddour, “Bearing vibration data collected under timevarying rotational speed conditions,” Data Br., vol. 21, pp. 1745–1749, Dec. 2018.

(Lin et al. 2013) M. Lin, Q. Chen, and S Yan, “Network in network,” Dec. 2013, arXiv:1312.4400.

(Liu et al. 2022) Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” in Proc. IEEE/CVF Conf. Comput. Vis. Patterm Recognit., New Orleans, LA, USA, 2022, pp. 11976–11986.

(Srivastava et al. 2014) N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” J. Mach. Learn. Res., vol. 15, no. 56, pp. 1929–1958, Jun. 2014.

(Tan et. al 2019) M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “MnasNet: Platform-aware neural architecture search for mobile,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Long Beach, CA, USA, 2019, pp. 2820–2828.

(Wu et al. 2020) H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, “Integer quantization for deep learning inference: Principles and empirical evaluation,” Apr. 2020, arXiv:2004.09602v1.

(Zhang et al. 2017) W. Zhang, G. Peng, C. Li, Y. Chen, and Z. Zhang, “A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals,” Sensors, vol. 17, no. 2, p. 425, Feb. 2017.

(Zhang et al. 2018) W. Zhang, C. Li, G. Peng, Y. Chen, and Z. Zhang, “A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load,” Mech. Syst. Signal Process., vol. 100, pp. 439–453, Feb. 2018.

(Case Western Reserve University) Bearing Data Center. Accessed: Jul. 28, 2023. [Online]. Available: https://engineering.case.edu/bearingdatacenter/.

(Society of Machine Failure Prevention Technology) Condition Based Maintenance Fault Database for Testing of Diagnostic and Prognostics Algorithms. Accessed: Jul. 28, 2023. [Online]. Available: https://www.mfpt.org/fault-data-sets/.