PPT PRESENTATION

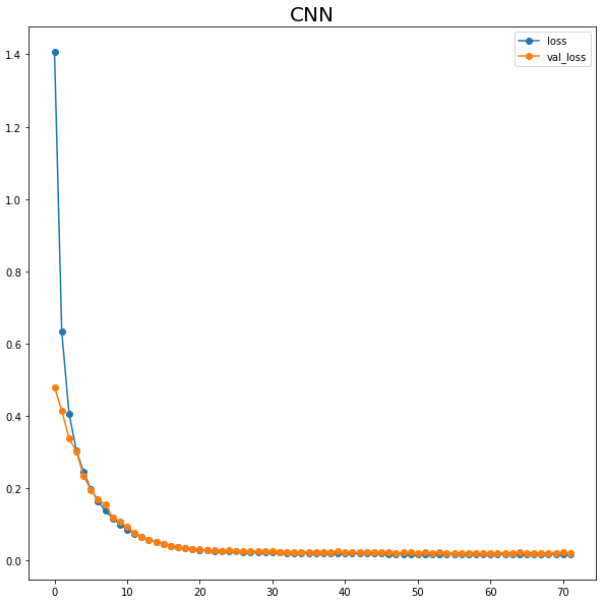
Enjoy your stylish business and campus life with BIZCAM

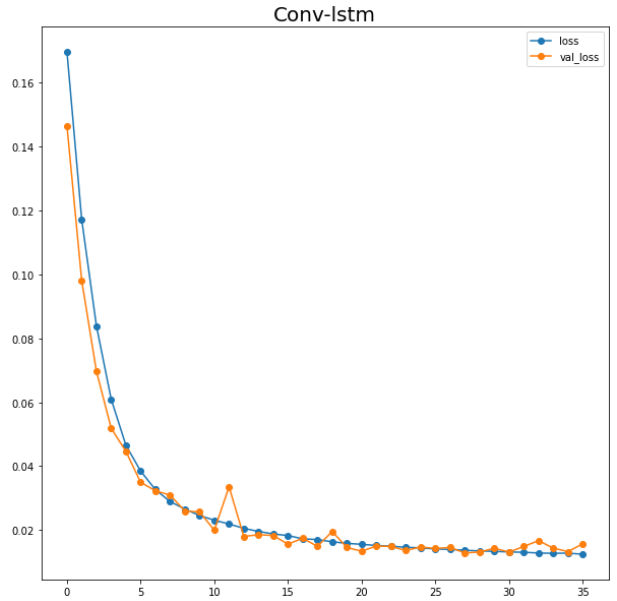
실험 세부 사항

- Conv-lstm에 들어가는 윈도우끼리의 채널 순서는 통일
- Drop-out 확률 조정(0.6 -> 0.3)
- Learning rate step 함수 도입

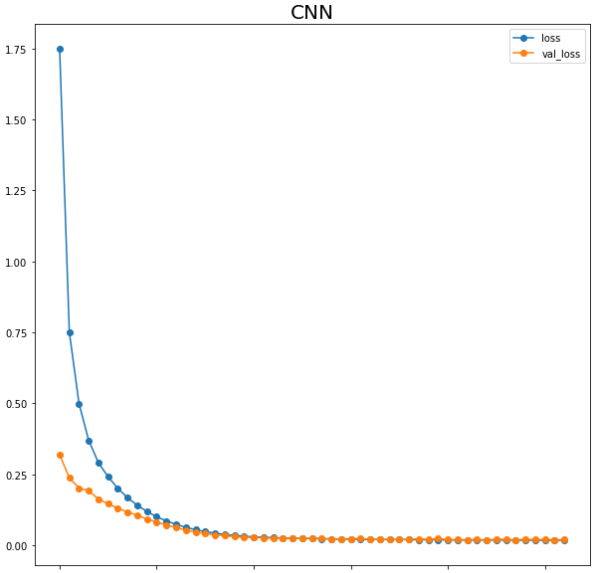
UWB

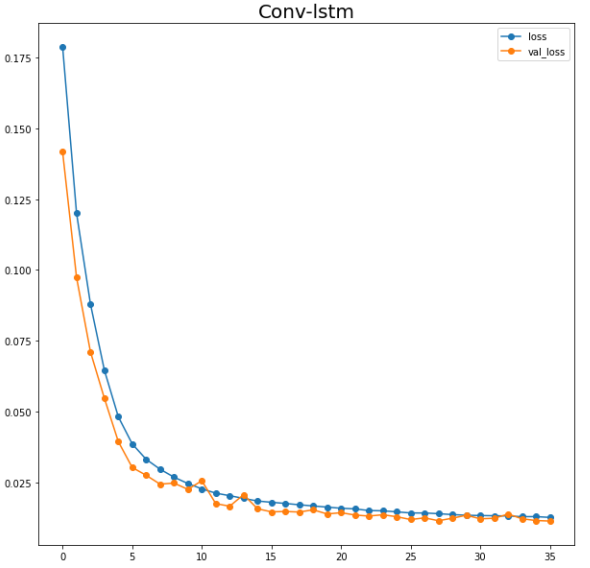
<TITT>





<TPITT>





실험 결과

- 노이즈가 매우 심할 때 상황(아직 중간이랑 없음은 실험X)
- 모든 실험은 10번 반복
- 증강 기법을 활용하면 안정적이게 나오지만(표준 편차 작음) 성능적 평균은 증강 전이 전반적으로 좋은 결과를 보임(오차라 볼 수 있는 거 50% / 차이 나는거 40% / 증강 후가 더 좋음 10%)
- 다른 환경(좁은 지역 넓은 지역) 혹은 운동(회전 지그제그 랜덤)에서도 이와 비슷한 경향을 보임

단위(m) : 평균(표준 편차)	증강 전	증강 후
TITT CNN	1.623(0.049)	1.6145(0.036)
TPITT CNN	1.535(0.058)	1.5462(0.029)
TITT Conv-LSTM	1.23(0.054)	1.2825(0.039)
TPITT Conv-LSTM	1.143(0.056)	1.1774(0.037)

Siamese

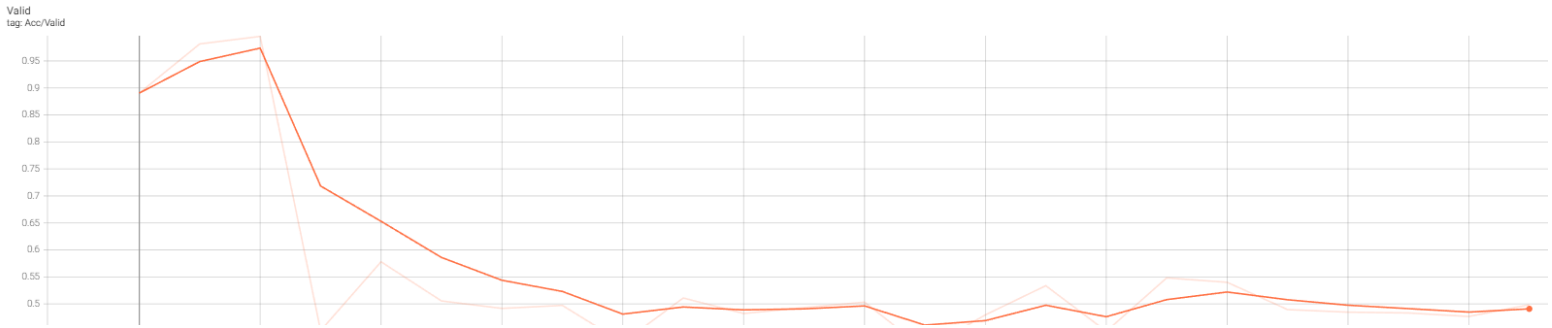
실험 세부 사항

- 학습 15000쌍 / val 4000쌍(5 way 1 shot) / test 4000쌍(5 way 1 shot)
- 데이터 셋 : Omniglot dataset
- Optimizer : Adam(1e-4)
- 모델 구조 : 논문과 같이 동일
- Input data : 105 * 105
- Loss : BCEWithLogitsLoss

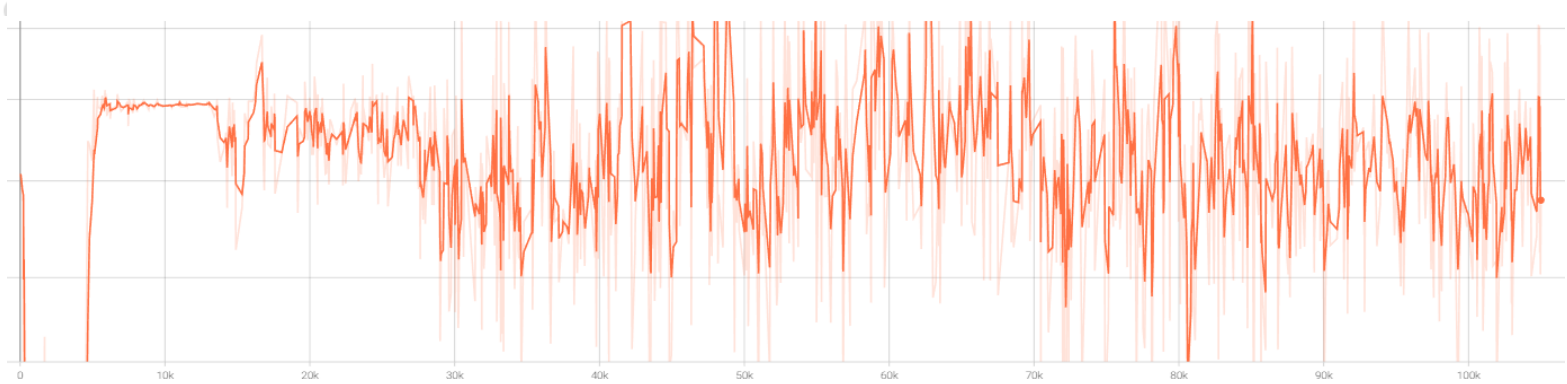
```
def __init__(self):
    super(SiameseNet, self).__init__()

    self.conv = nn.Sequential(
        nn.Conv2d(1, 64, 10),  # 64@96*96
        nn.ReLU(inplace=True),
        nn.MaxPool2d(2),  # 64@48*48
        nn.Conv2d(64, 128, 7),
        nn.ReLU(inplace=True),  # 128@42*42
        nn.MaxPool2d(2),  # 128@21*21
        nn.Conv2d(128, 128, 4),
        nn.ReLU(inplace=True),  # 128@18*18
        nn.MaxPool2d(2),  # 128@9*9
        nn.Conv2d(128, 256, 4),
        nn.ReLU(inplace=True),  # 256@6*6
    )
    self.liner = nn.Sequential(nn.Linear(9216, 4096), nn.Sigmoid())
    self.out = nn.Linear(4096, 1)
```

Siamese

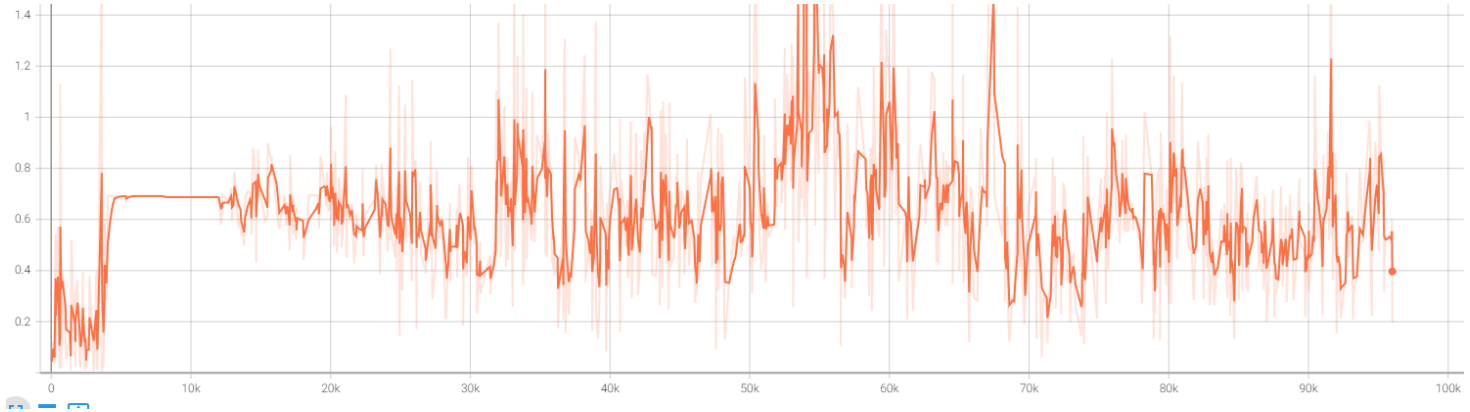
<Val acc(model)>



<Train Loss>



<Val Loss>



실험 결과

- 실험 결과 5way 1shot은 99.1% 성능을 보임
- 이는 반도체 class가 9개 이기에 다음과 같이 5way로 선정
- 본 논문에서는 20way 1 shot
- 마찬가지로 돌려본 결과 논문 결과보다 2% 정도 차이 보임 (81%)

Protonet

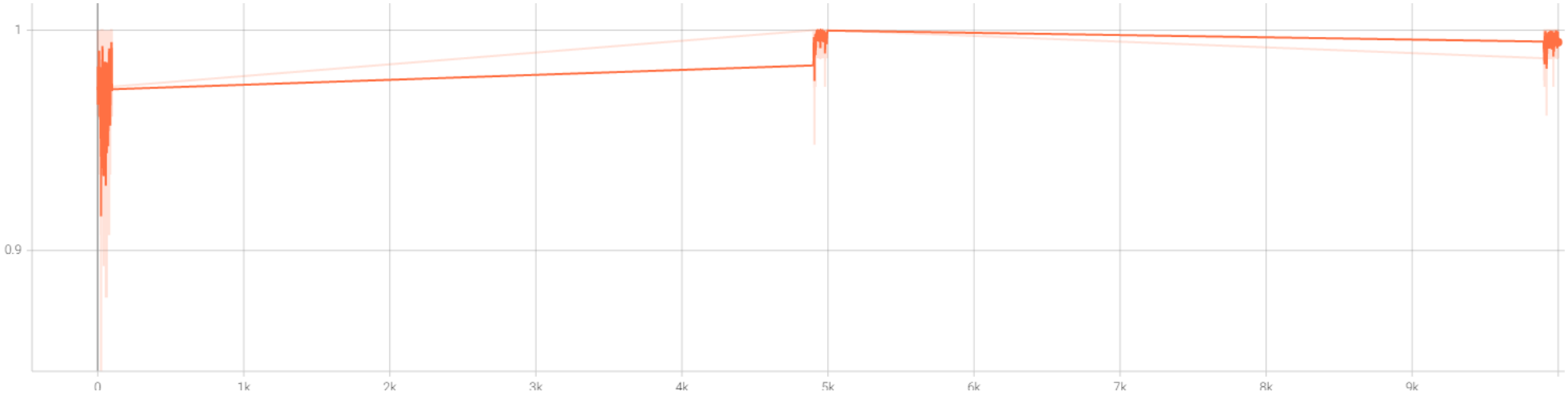
실험 세부 사항

- 학습 : 1 episode당 60개의 class와 15개의 query, support point를 사용
- 데이터 셋 : Omniglot dataset
- Optimizer : Adam(1e-4)
- 모델 구조 : 논문과 같이 동일
- Input data : 105 * 105
- Loss : prototypical loss

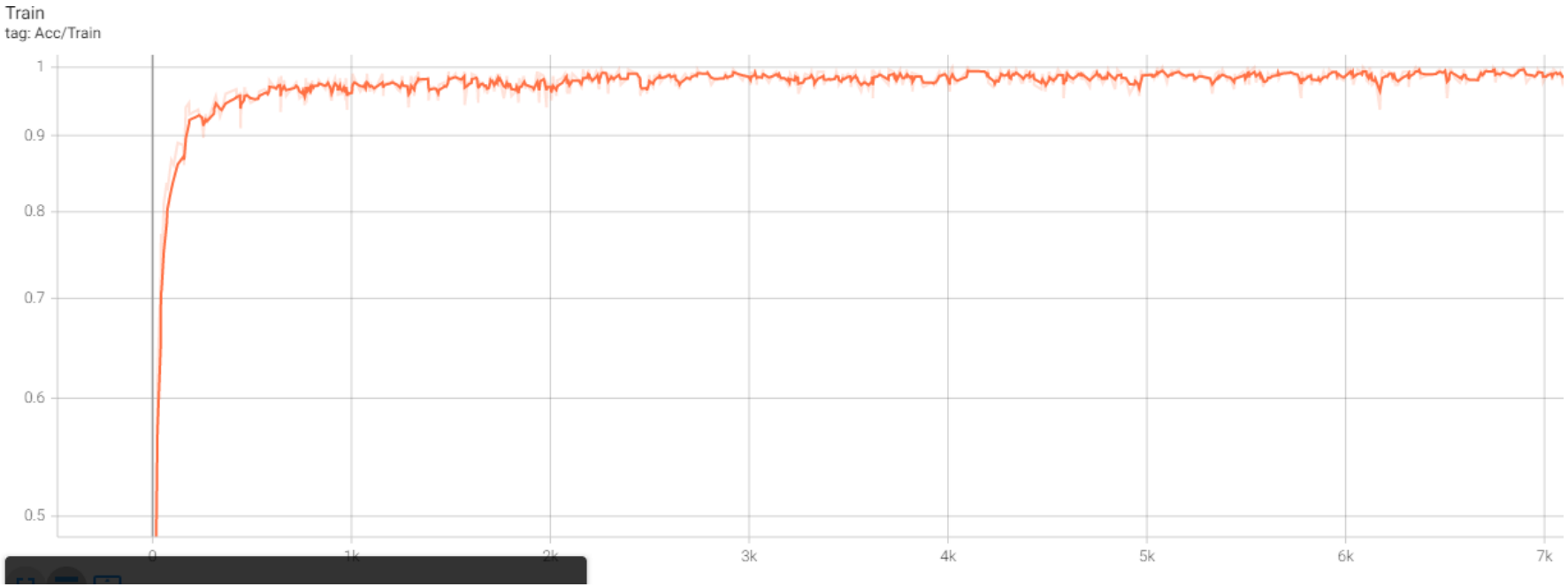
```
def prototypical_loss(input, target, n_support, device):
    classes = torch.unique(target)
    n_classes = len(classes)

    # Make prototypes
    support_idxs = torch.stack(list(map(lambda c: target.eq(c).nonzero()[:n_support].squeeze(1), classes)))
    prototypes = torch.stack([input[idx_list].mean(0) for idx_list in support_idxs])

    # Make query samples
    n_query = target.eq(classes[0].item()).sum().item() - n_support
    query_idxs = torch.stack(list(map(lambda c: target.eq(c).nonzero()[n_support:], classes))).view(-1)
    query_samples = input[query_idxs]

    dists = euclidean_dist(query_samples, prototypes)

    log_p_y = F.log_softmax(-dists, dim=1)
    y_hat = log_p_y.argmax(1)
    target_label = torch.LongTensor([x for x in range(n_classes) for _ in range(n_query)]).to(device)

    loss_val = torch.nn.NLLLoss()(log_p_y, target_label)
    acc_val = y_hat.eq(target_label.squeeze()).float().mean()

    return loss_val, acc_val
```

Protonic

<Train Acc>



<Val Acc>



Siamese

실험 결과

- 실험 : 5 way 1 shot로 진행
- 결과 : loss = 0.014+- 0.005 / Accuracy = 0.953
- 논문 결과

Model	Dist.	Fine Tune	5-way Acc.		20-way Acc.	
			1-shot	5-shot	1-shot	5-shot
MATCHING NETWORKS [29]	Cosine	N	98.1%	98.9%	93.8%	98.5%
MATCHING NETWORKS [29]	Cosine	Y	97.9%	98.7%	93.5%	98.7%
NEURAL STATISTICIAN [6]	-	N	98.1%	99.5%	93.2%	98.1%
PROTOTYPICAL NETWORKS (OURS)	Euclid.	N	**98.8%**	**99.7%**	**96.0%**	**98.9%**