한국소음진동공학회 AI혁신위원회 주관
2022년 인공지능 강습회

AI 기계이상진단을 위한 인공지능 학습 기법

제 6강 데이터 증강과 이상치 노출

한국과학기술원 전기및전자공학부

최정우
jwoo@kaist.ac.kr

KAIST EE

목차

- **데이터 전처리 기법**
  - 데이터 증강 기법 (data augmentation)
- **이상진단을 위한 분류 태스크 성능 향상 기법**
  - Outlier Exposure
  - Mixup

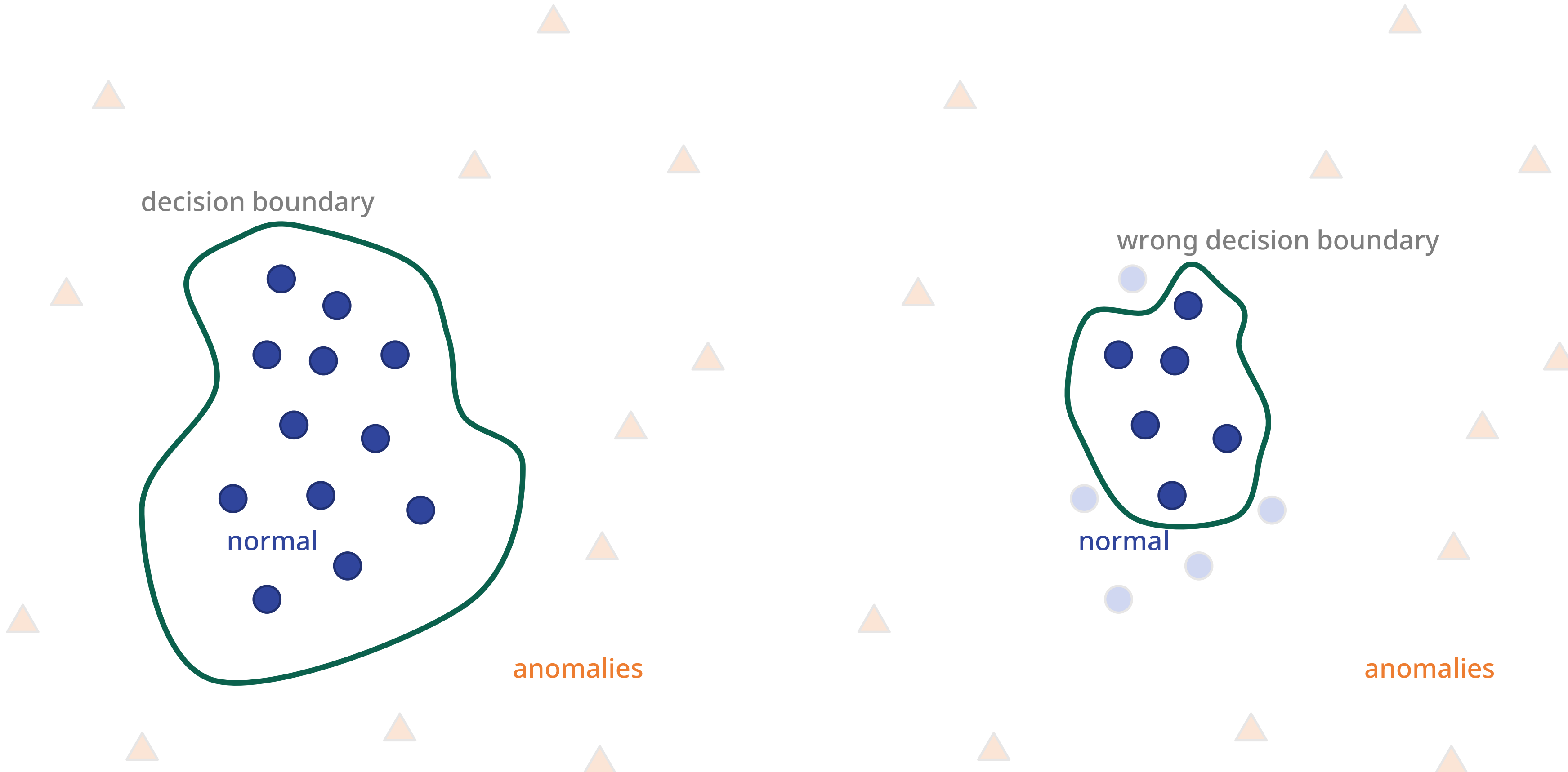
Lack of training data

- New production model
  - Human resources are required for data collection
  - High-variability of normal data depending on the operating condition

- Insufficiency in machine condition measurements
  - One cannot measure enough data before the activation of a new plant

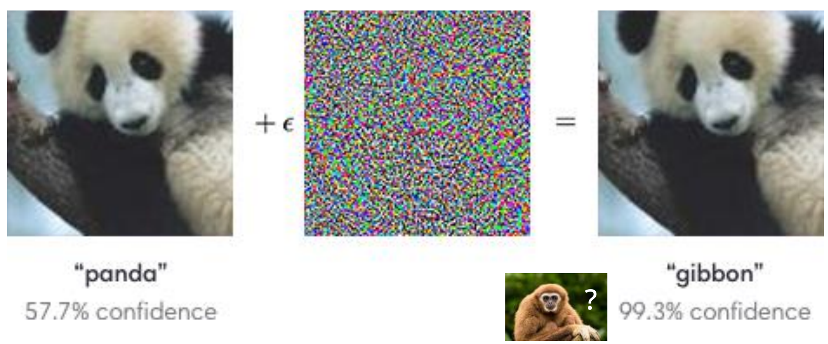
Lack of training data

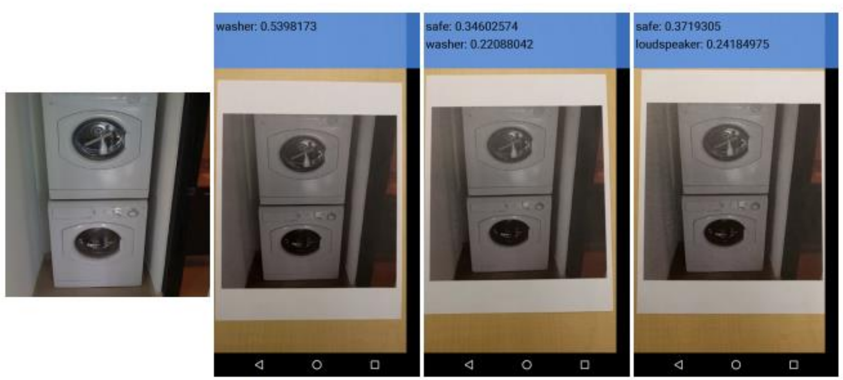
- In self-supervised training



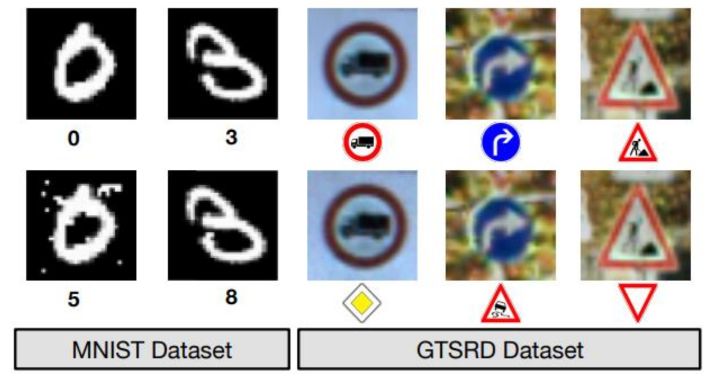
Example of overfitted model

- Adversarial attack examples
  - Adding small noise





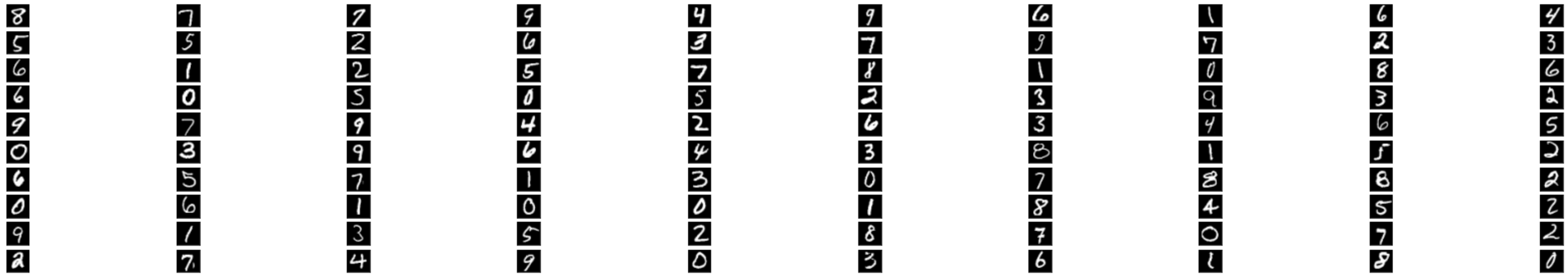
(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$



Images from
https://arxiv.org/pdf/1602.02697.pdf

MNIST example

- Training dataset: 600 samples (out of 24,000 samples)
  - Normal: digits 0 – 4
  - Anomalies: digits 7,8,9



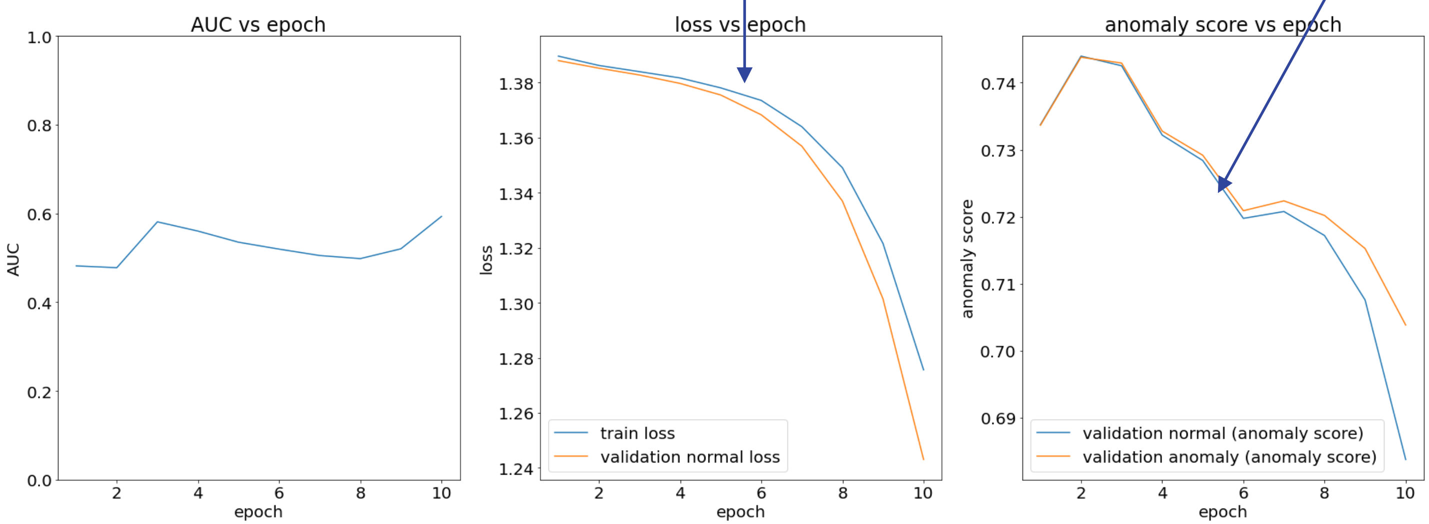
- Model
  - Pretext: classification task
  - Conv 2D classification model (3 conv 2D layers + one linear)
  - Anomaly score: maximum softmax probability
  - Label smoothing: 0.2
  - Test over 3584 samples

Problems with small dataset

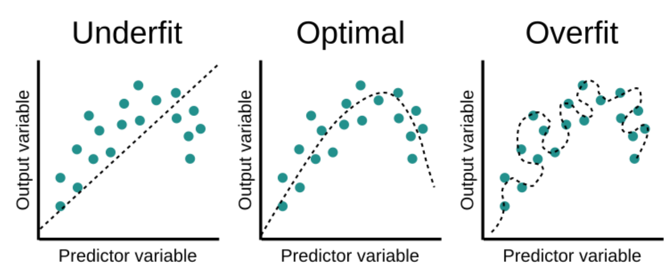
- Overfitting

trained model does not work well with validation data

no big difference in anomaly scores

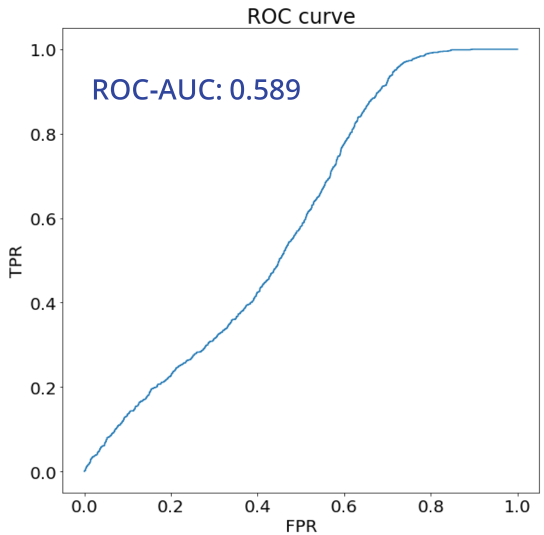


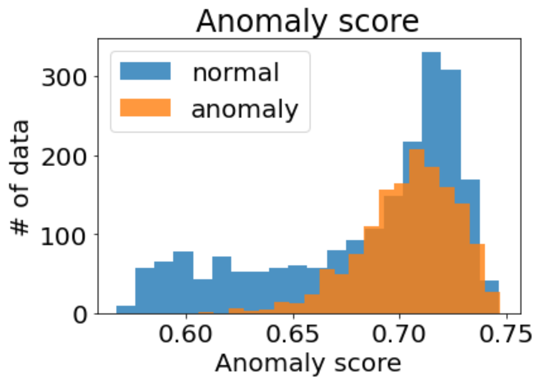
  - Big gap between performances for the training & validation datasets
  - Model only works well for the training data



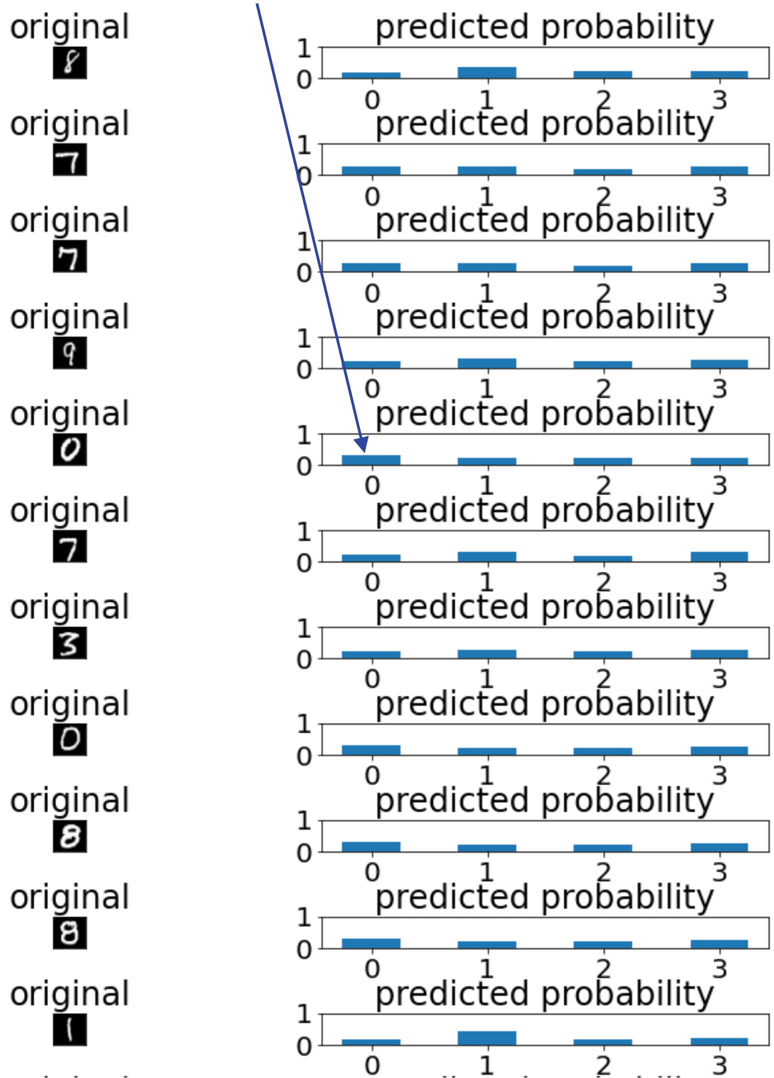
Problems with small dataset

- Low AD performance



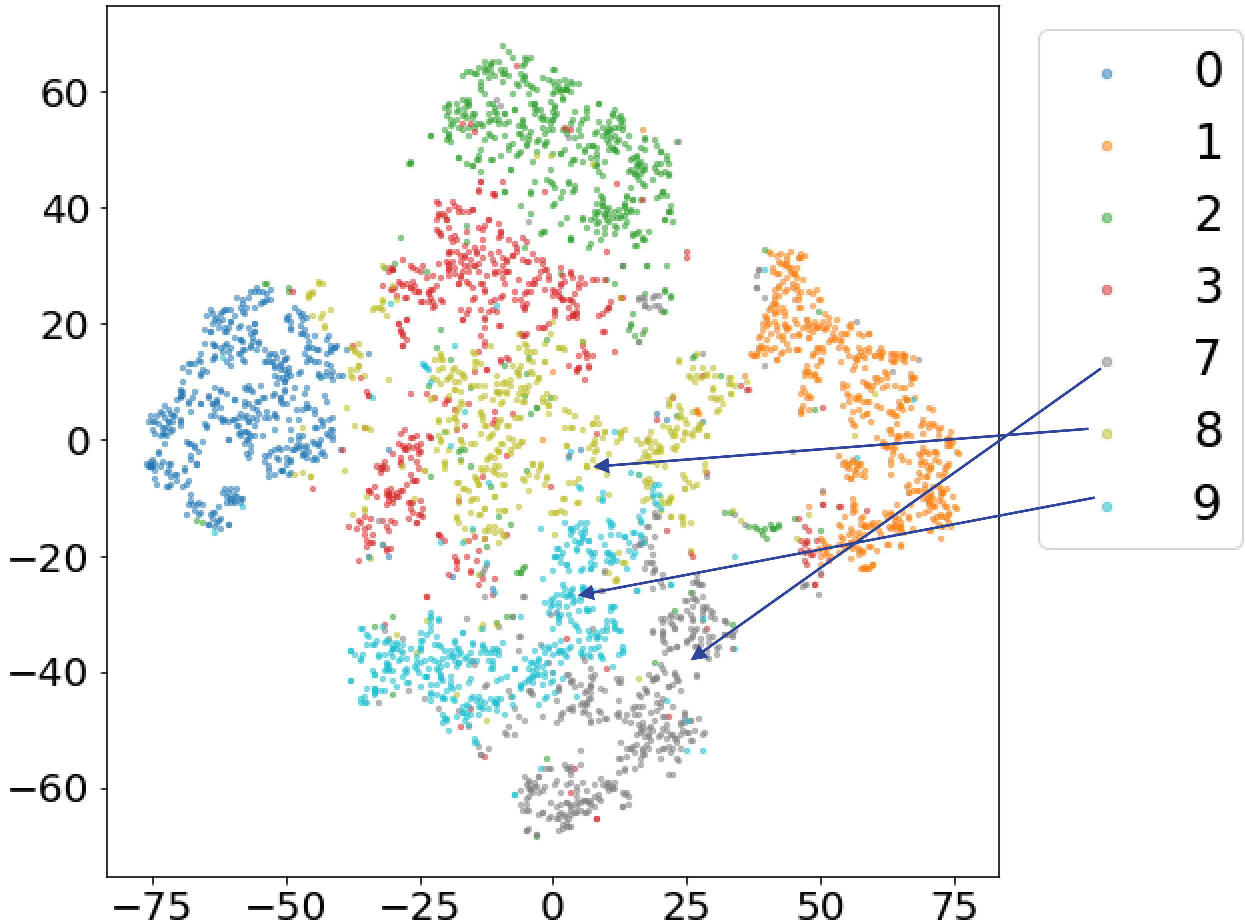


Even cannot recognize (unseen) normal data



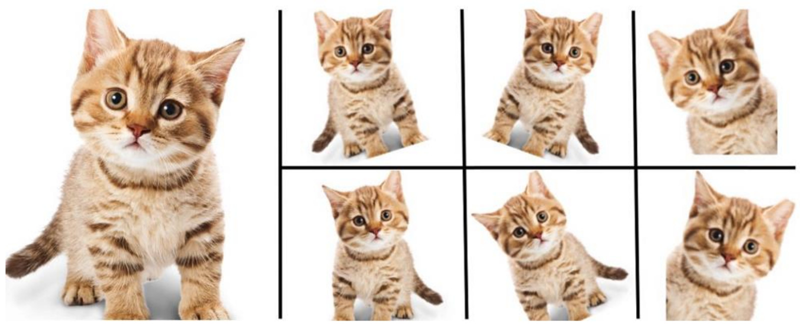
Problems with small dataset

- Scattered samples in feature space



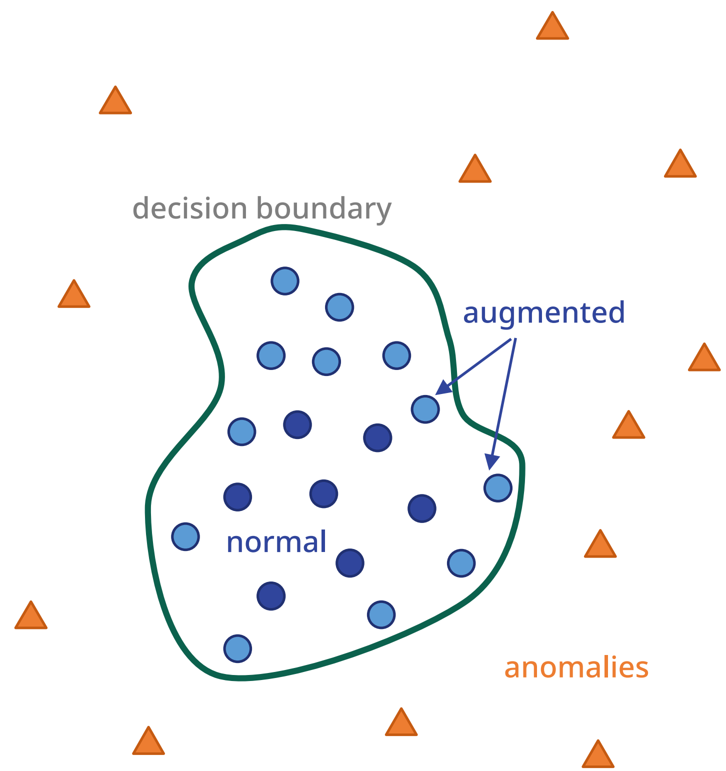
Augmentation

- Increasing the size of training data
  - Using proper transforms



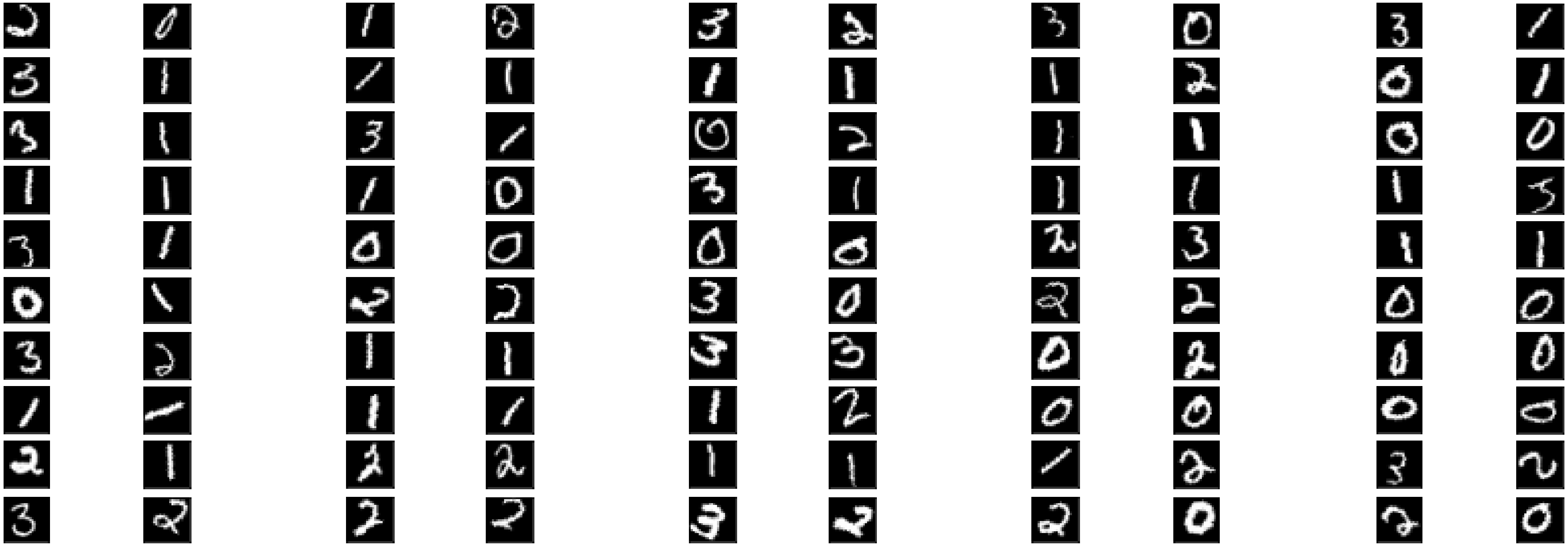
  - Image: translation, rotation, flip, zoom, …

- What is a ‘proper transform’?
  - Relies on the human’s prior knowledge
  - T(normal data) ∈ normal data



Data Augmentation

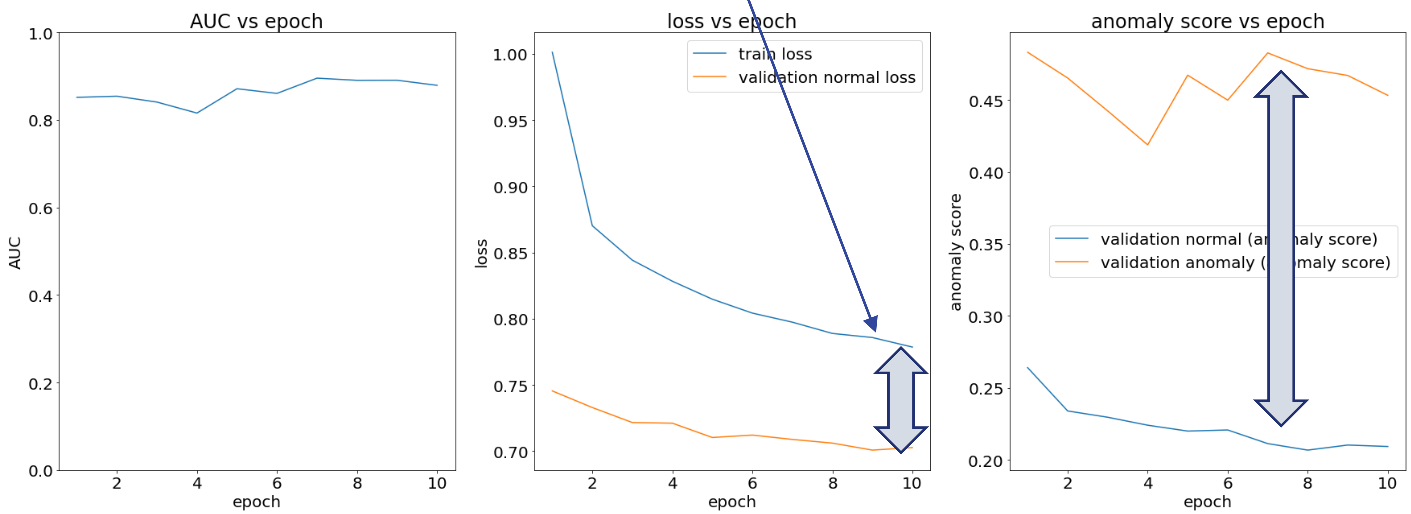
- Examples of augmented data (rotation <$\pm 30°$, pixel shift < ± 2 px)



Training with augmented dataset

- Training performance

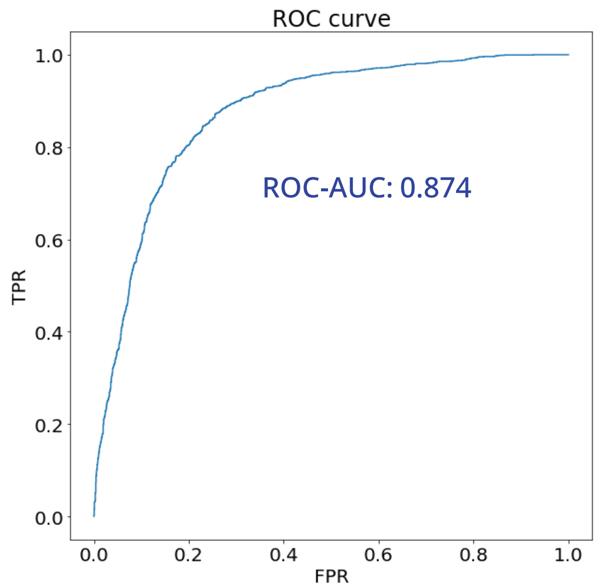
smaller gap between train & validation losses

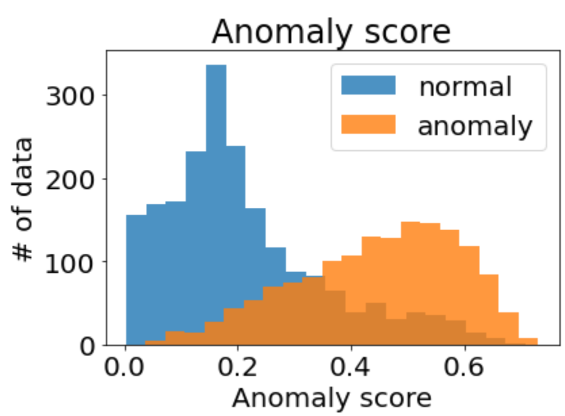


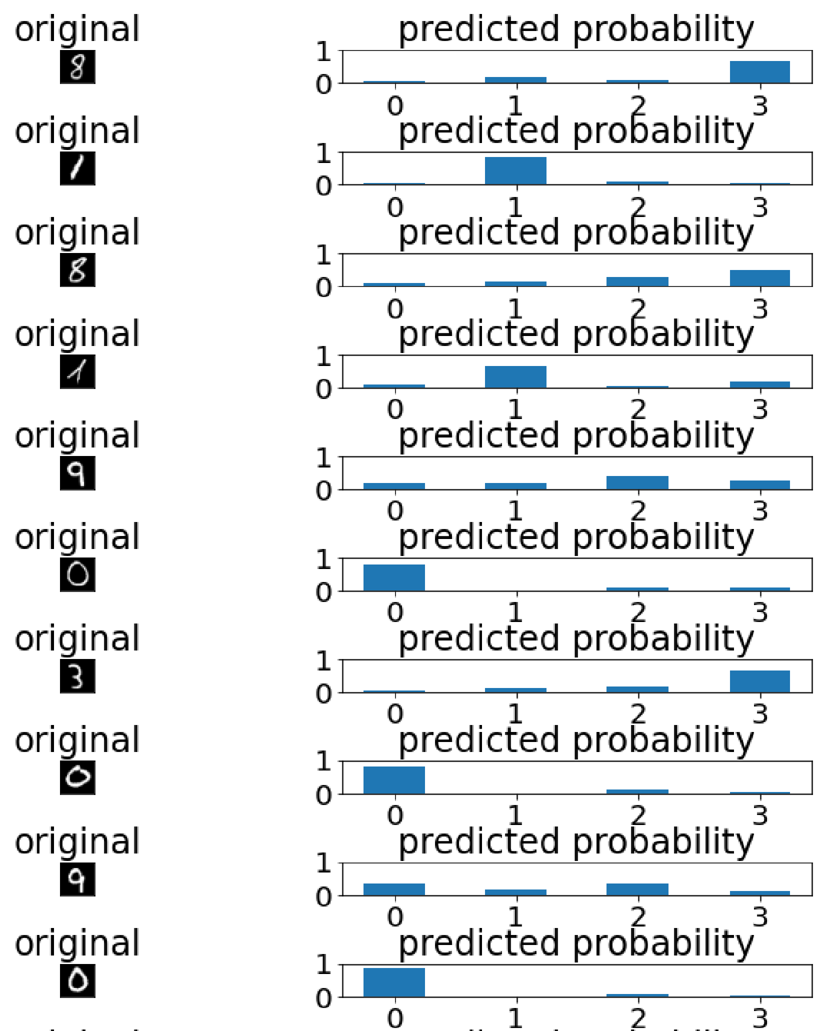
  - Smaller gap between training & validation losses
  - Anomaly scores for normal & abnormal data are more distinguishable

Training with augmented dataset

- High AD performance

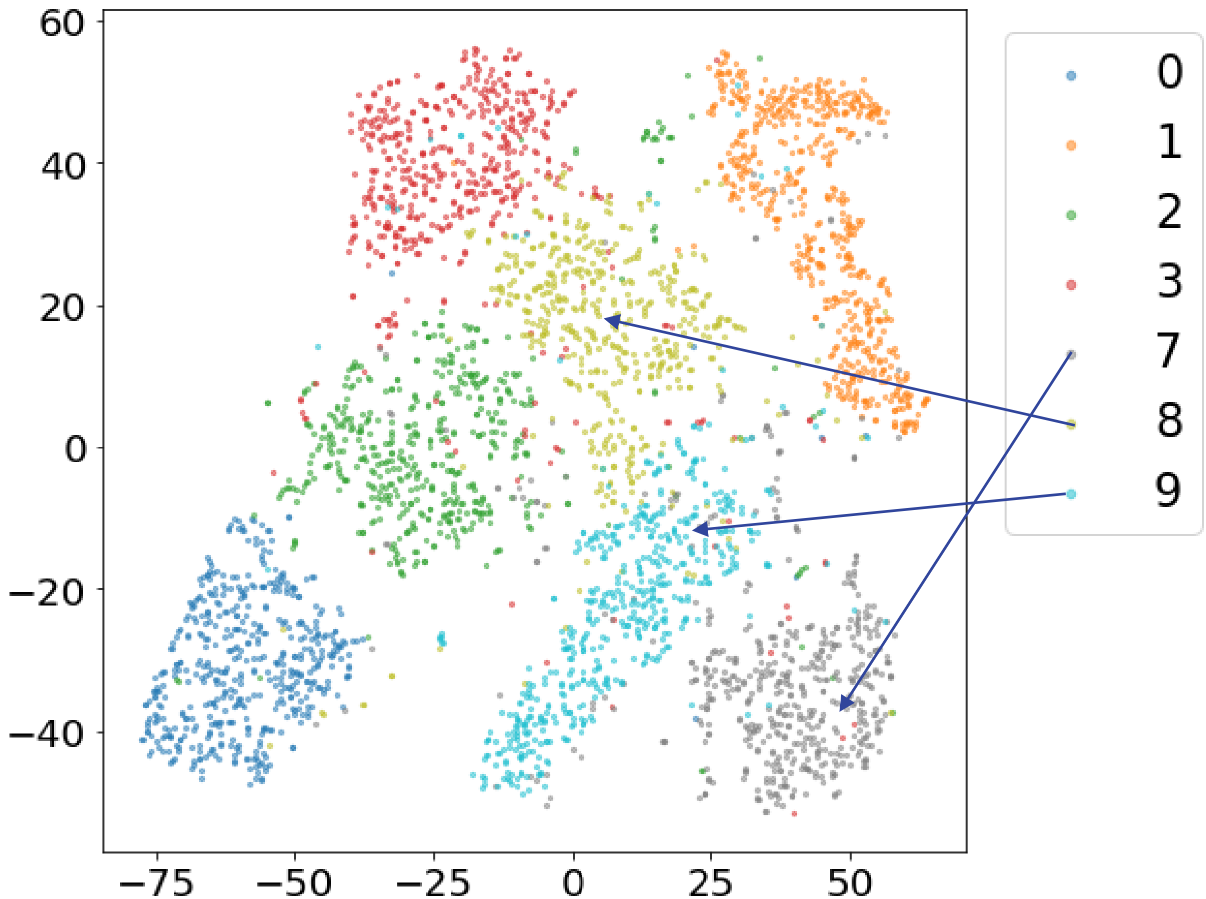






Training with augmented dataset

- More separation in feature space



Code review

- Augmentation part

```
# define transforms for augmentation
tfs = TF.Compose([
                    TF.RandomAffine(degrees=0, translate=(0.1,0.1)),
                    TF.RandomRotation(degrees=(-30, 30))
                    ])

# get small sized dataset
# Sample the data in same proportion of labels
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, train_size=train_dataset_num, random_state=0)

indices = list(range(len(mnist_train)))   # gen list 0, ... ,len(mnist_train)
labels  = [y for _, y in mnist_train]      # get labels from mnist
train_idx = [idx for idx, _ in sss.split(indices,labels)] # separate indices
train_subset  = Subset(mnist_train, train_idx[0]) # get subset
```

declare augmentation transform object

Sample 600 data from MNIST train dataset

Code review

- Augmentation part

```
# Begin augmentation
# init dataset list with original dataset
aug_dataset = [getSubset(train_subset)]

if not NAUG==0:
  # repeat random augmentation NAUG-1 times
  for ii in range(NAUG-1):
    aug_dataset.append(getSubset(train_subset, transform=tfs ))

# concatenate datasets using ConcatDataset function
train_dataset = torch.utils.data.ConcatDataset(aug_dataset)

train_dataloader = DataLoader(train_dataset, batch_size=BATCH, shuffle=True)
```

Apply augmentation and attach to list

Unify datasets using ConcatDataset

Code review

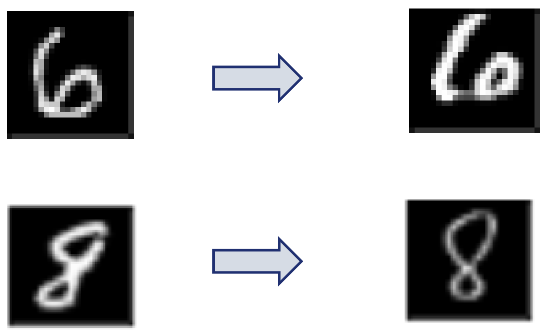
- getSubset: customized class for building sub-dataset with transforms

```python
class getSubset(TensorDataset):
    def __init__(self, subset, transform=None, target_transform=None):
        self.subset = subset
        self.transform = transform
        self.target_transform = target_transform

    def __getitem__(self, index):
        x, y = self.subset[index]
        if self.transform:
            x = self.transform(x)
        if self.target_transform:
            y = self.target_transform(y)
        return x, y

    def __len__(self):
        return len(self.subset)
```

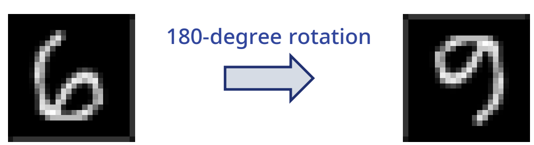
- get the transform for data
- get the transform for labels
- extract subset
- Apply transform to data
- Apply transform to labels

Limitation of Data Augmentation

- Augmentation is not a fundamental solution (rather a sidekick)
  - Transformed normal data are still normal
  - Cannot make complex modifications: possible expressions are limited



  - Some transforms can yield confusion between labels

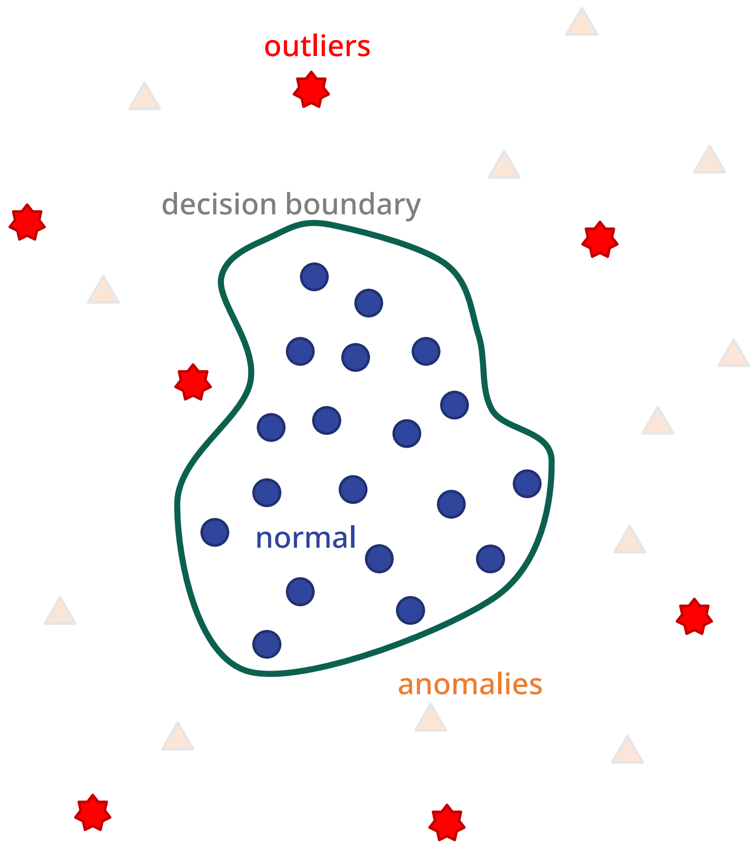


Outlier exposure

Injection of outlier examples to model

Outlier exposure

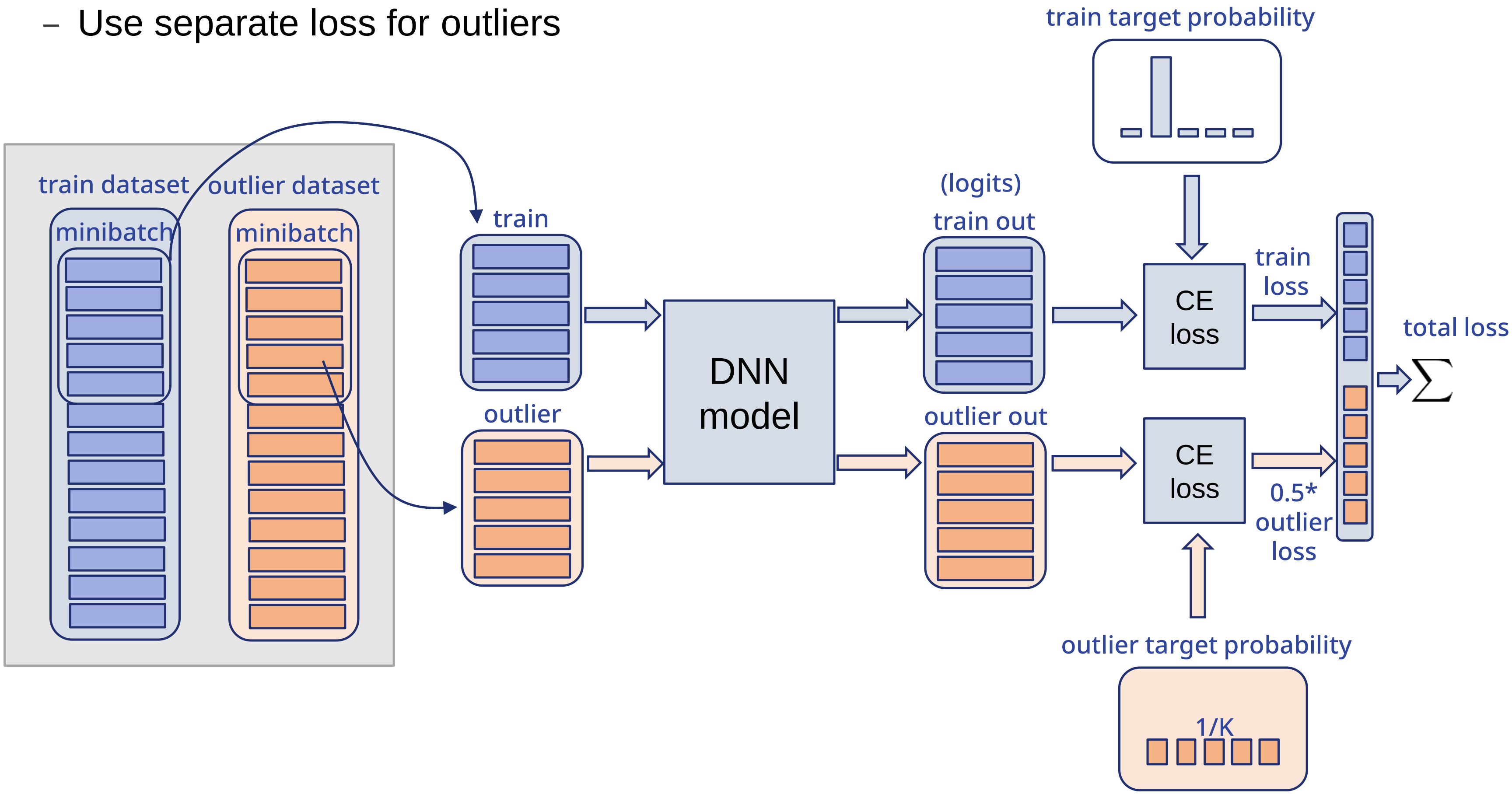
- Prior knowledge of anomalies
  - Some data can never be normal
  - Exposing a model to out-of-distribution (OOD) examples entirely disjoint from normal data (auxiliary data)
  - Learning cues for whether an input is modeled / unmodeled



- Benefits
  - Learn effective heuristics for detecting OOD samples
  - Enabling the detection of novel forms of anomalies

Outlier exposure

- How to
  - Train normal & outlier data together in the same batch
  - Use separate loss for outliers



Outlier exposure

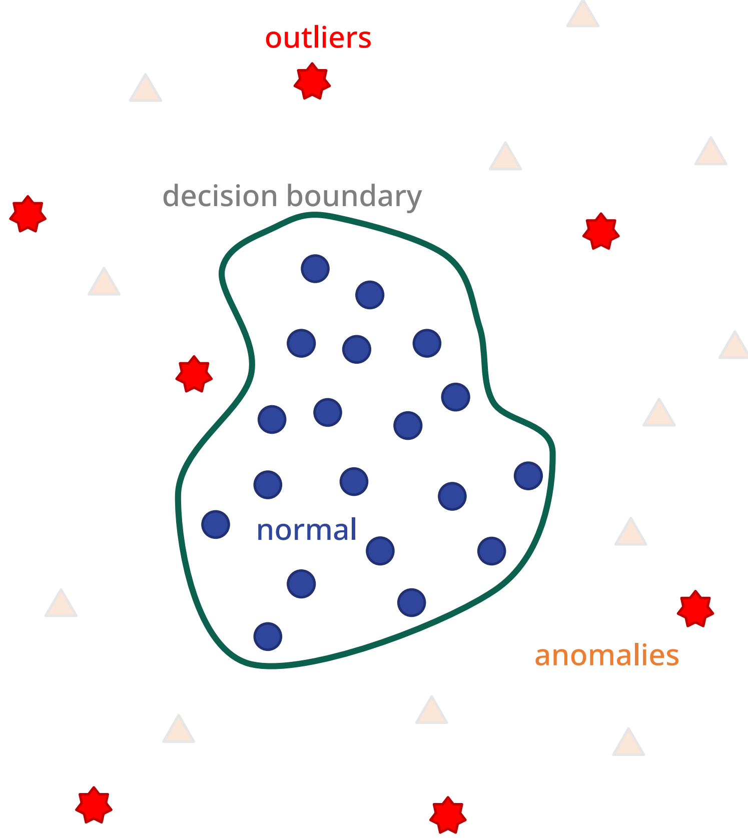
- Loss function

$$\mathbb{E}_{(x)\sim\mathcal{D}_{\text{in}}}\left[\mathcal{L}(f(x))\right] + \lambda\mathbb{E}_{x'\sim\mathcal{D}_{\text{out}}^{\text{OE}}}\left[\mathcal{L}_{\text{OE}}\left(f\left(x'\right)\right)\right]$$

in-distribution
(normal data)

out-of-distribution (OOD)
(outliers)

  - λ: controls relative weight between two losses



- Example: MNIST
  - Convolutional Classifier: same model
  - Label smoothing: 0.2
  - No augmentation, full training set

Outlier exposure: code review

- Hyperparameters

```
EPOCHS = 5          # Number of epochs to train
BATCH = 32          # Minibatch size
ORGCLASS_NUM = 10   # Num of original classes (10: 0 to 9)
ANOMALY_NUM = [7,8,9]   # (list) Digits will be used as anomalous data
NORMAL_NUM =  [0,1,2,3]   # (list) Digits used as normal data
OUTLIER_NUM = [4,5,6] # (list) Digits used as ouliter

TEMPSC = 1.0    # temperature parameter (for temperature scaling)
LBSMOOTH = 0.2  # label smoothing parameter

SCORE_MODE = 0 # Anomaly score : 0 for 1-MSP, 1 for H(u;.)
```

Outlier exposure: code review

- Dataloader

```
train_dataloader = DataLoader(train_dataset, batch_size=BATCH, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=BATCH)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH)

# outlier dataset
outlier_idx = [i for i,v in enumerate(mnist_train) if v[1] in OUTLIER_NUM]
outlier_dataset = Subset(mnist_train,outlier_idx)
outlier_dataloader = DataLoader(outlier_dataset, batch_size=BATCH, shuffle=True)
```

Outlier exposure: code review

- Anomaly score

```python
loss_fn = nn.CrossEntropyLoss(label_smoothing=LBSMOOTH) # CCE
def anomaly_score(logits, mode=0): # mode = 0 :
    if mode == 0:
      # Anomaly score = 1 - MSP
      softmaxprob = torch.softmax(logits, dim=1)
      MSP = torch.max(softmaxprob, dim=1).values
      return torch.tensor(1) - MSP
    if mode == 1:
      # Anomaly score = -H(u,.)
      uni = torch.ones(logits.shape[0],logits.shape[1]).float().to(device)/ ₩
        torch.tensor(len(NORMAL_NUM)).float().to(device) # make uniform distribution
      score = -F.cross_entropy(logits,uni,reduction='none') # -CCE (negative since anomaly
      #print(score)
      return score
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3) # Adam as optimizer
```

Outlier exposure: code review

- Trainer 1

```
# Here, outlier_dataloader is added. More GPU memory is required for Outlier exposure.
def train(dataloader, outlier_dataloader, model, loss_fn, optimizer):
    model.train()
    size = len(dataloader.dataset)
    losses = []
    in_losses = []
    oe_losses = []
    batch = 0
    for indist, oedist in zip(dataloader, outlier_dataloader):                # each dist is tuple of (img, label)

        # unzip tuple
        in_image, in_idx = indist
        oe_image, oe_idx = oedist

        # in distribution dataset (normal data)
        logits, _ = model(in_image.to(device))                  # logit (batch, class)
        p = F.one_hot(in_idx,ORGCLASS_NUM)[:,NORMAL_NUM]
        p = p.float().to(device)
        in_dist_loss = loss_fn(logits, p)
        in_losses.append(in_dist_loss.cpu().detach())
```

make a tuple of two minibatches from train, outlier dataset

get (image, label) of training data

get (image, label) of outlier data

same as before

Outlier exposure: code review

- Trainer 2

outlier training

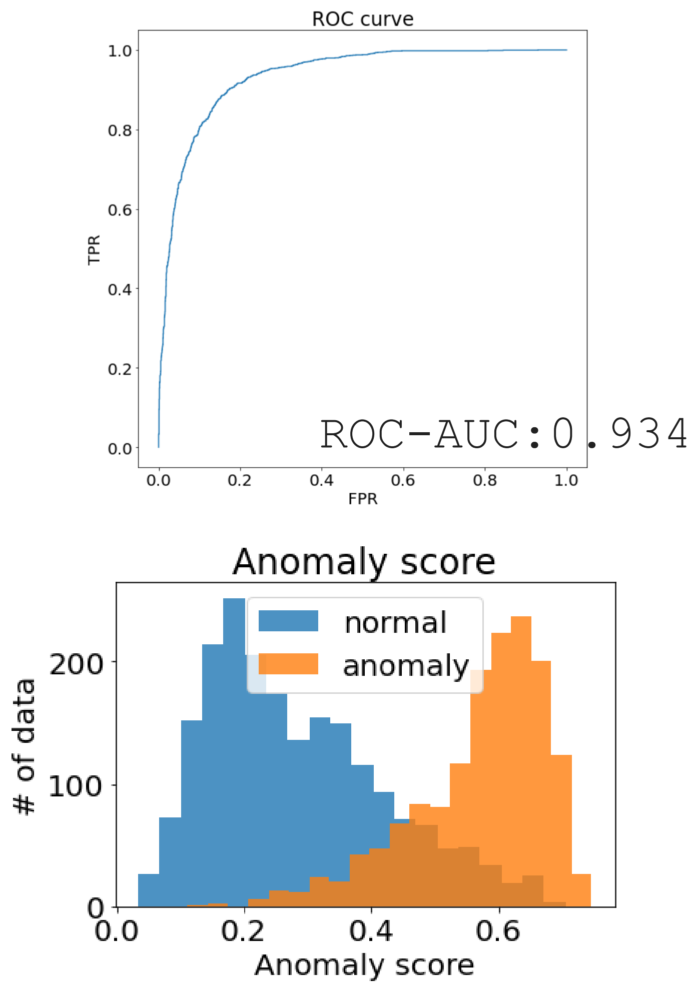
target distribution: uniform

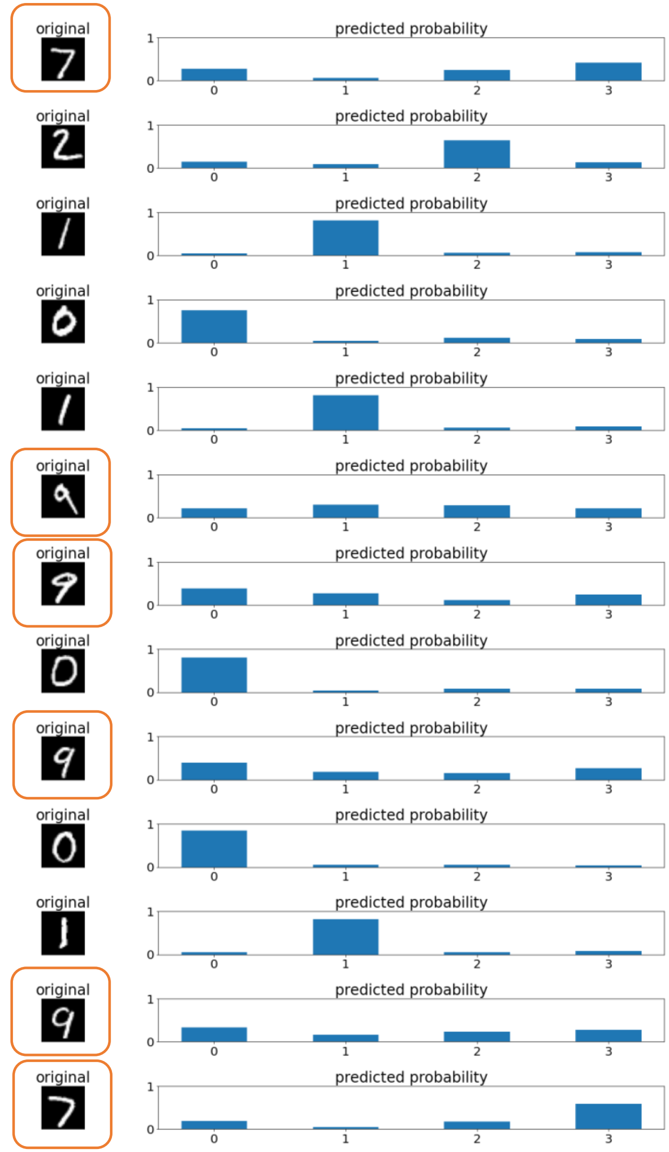
outlier loss

weighted sum of two losses

```
# OE
outlier_logits, _ = model(oe_image.to(device))
p_oe = torch.ones(oe_image.shape[0],len(NORMAL_NUM)) / torch.tensor(len(NORMAL_NUM)) # uniform dist.
p_oe = p_oe.float().to(device)
oe_loss = loss_fn(outlier_logits, p_oe) # CCE
oe_losses.append(oe_loss.cpu().detach())

# joint training

loss = in_dist_loss + 0.5*oe_loss # 0.5 for visual task
losses.append(loss.cpu().detach())

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

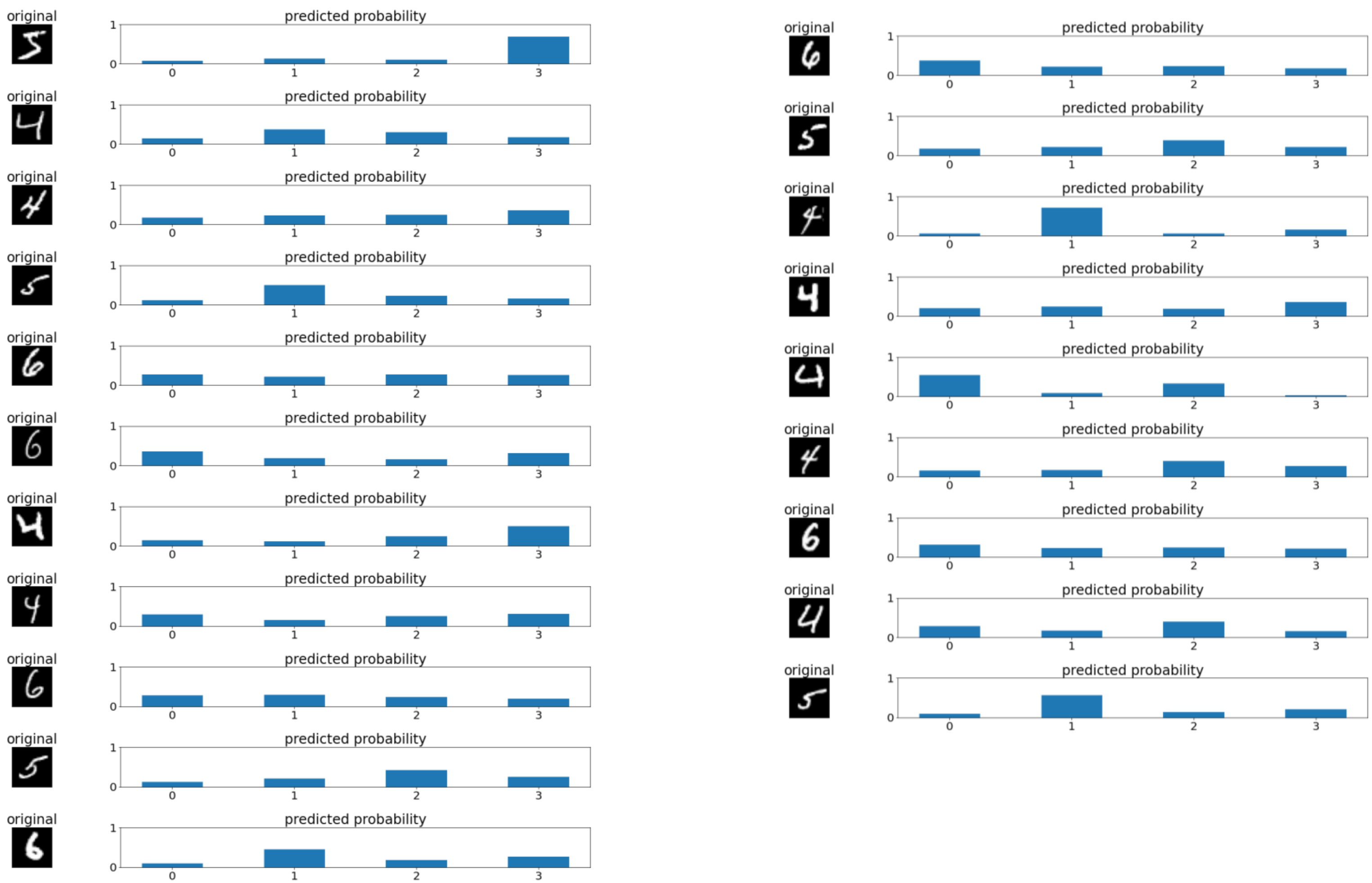
Outlier exposure: result





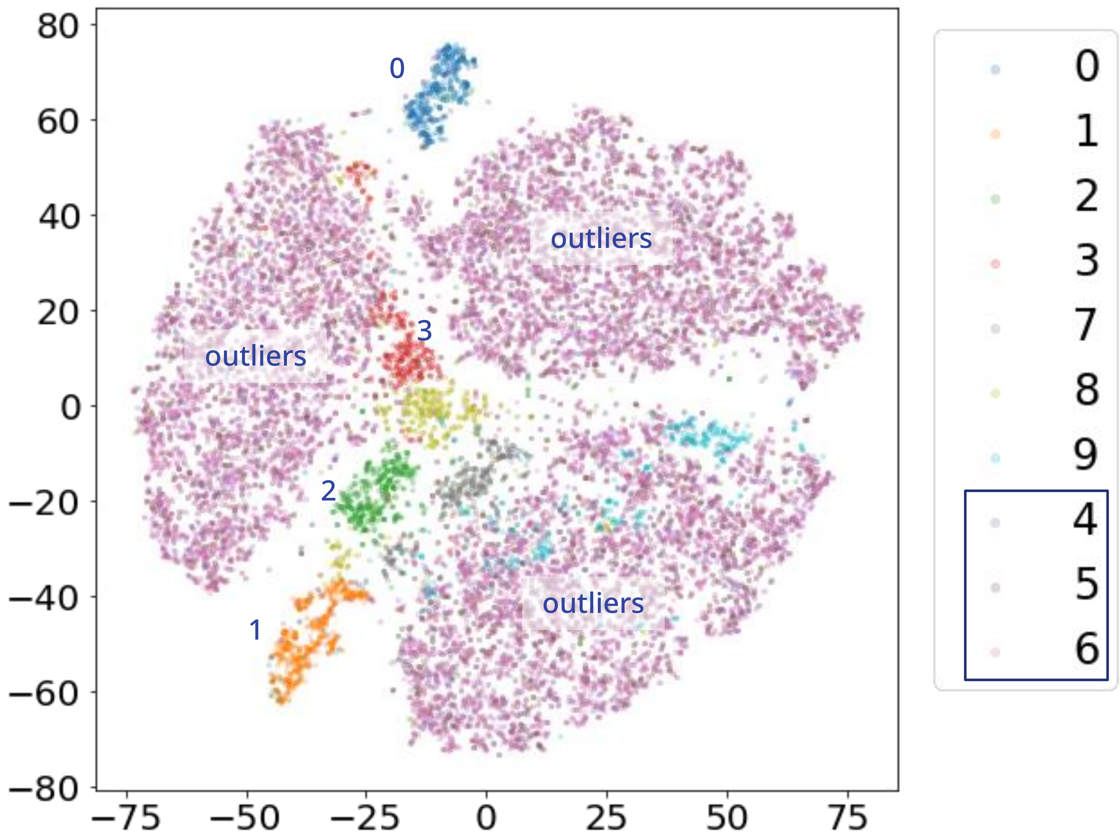
Outlier exposure: result

- Outlier prediction probability (seen outlier data during training)



Outlier exposure: result

- t-SNE plot



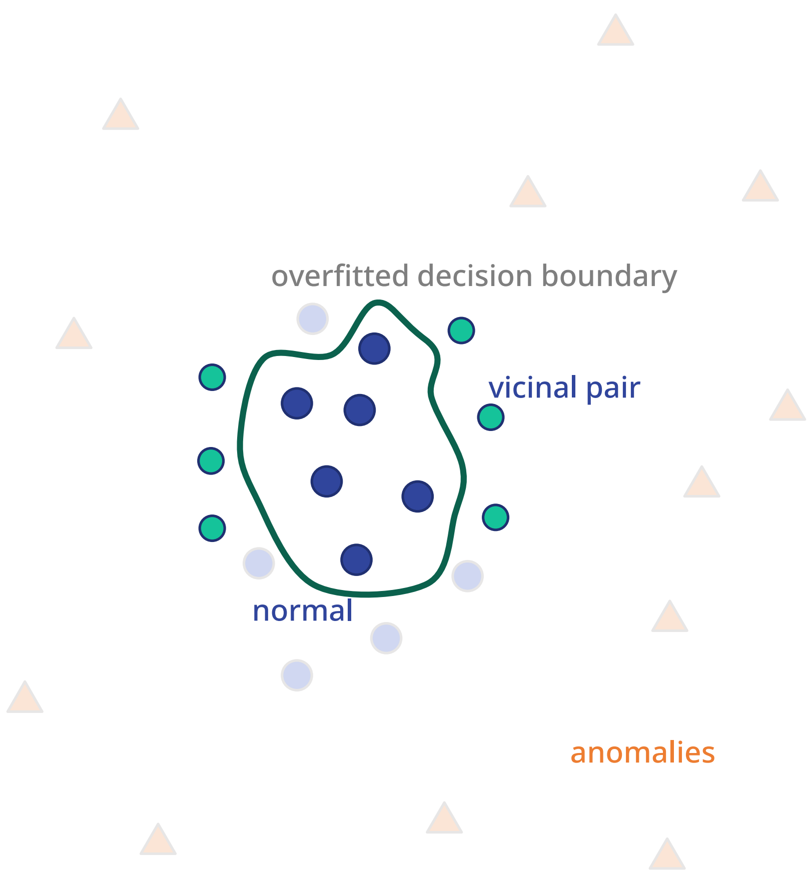
Other pretext tasks

1. Reconstruction
2. Classification
3. Mixup

mixup: Beyond Empirical Risk Minimization
https://arxiv.org/abs/1710.09412

Overfitting problem

- Empirical risk minimization (ERM)



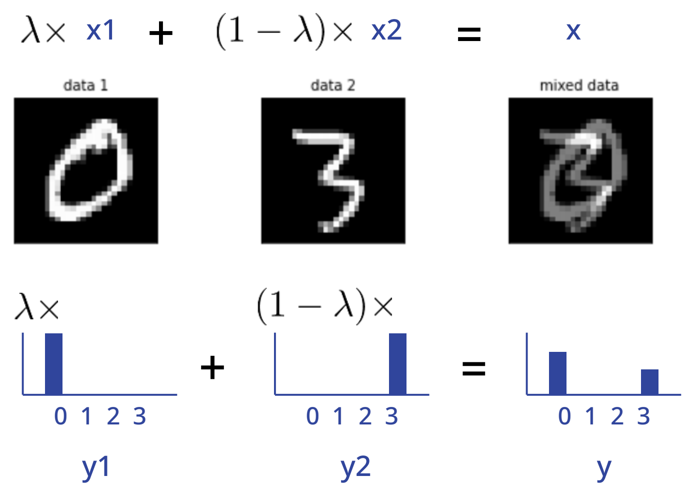
- DNN model minimizes expected risk at the observed points
  - Empirical risk minimization
- To minimize ERM, a model can memorize the training data
  - Vulnerable to unseen data
- What if we can find a virtual feature-target pair in the vicinity of the training pair?
  - Empirical vicinal risk minimization!

Mixup

- Mix two classes

  - Mix up both input and target probability

```
# y1, y2 should be one-hot vectors
for (x1, y1), (x2, y2) in zip(loader1, loader2):
    lam = numpy.random.beta(alpha, alpha)
    x = Variable(lam * x1 + (1. - lam) * x2)
    y = Variable(lam * y1 + (1. - lam) * y2)
    optimizer.zero_grad()
    loss(net(x), y).backward()
    optimizer.step()
```



  - Not just an augmentation
    (target probability is redefined)

  - Model is trained to predict mixed
    probability *y*

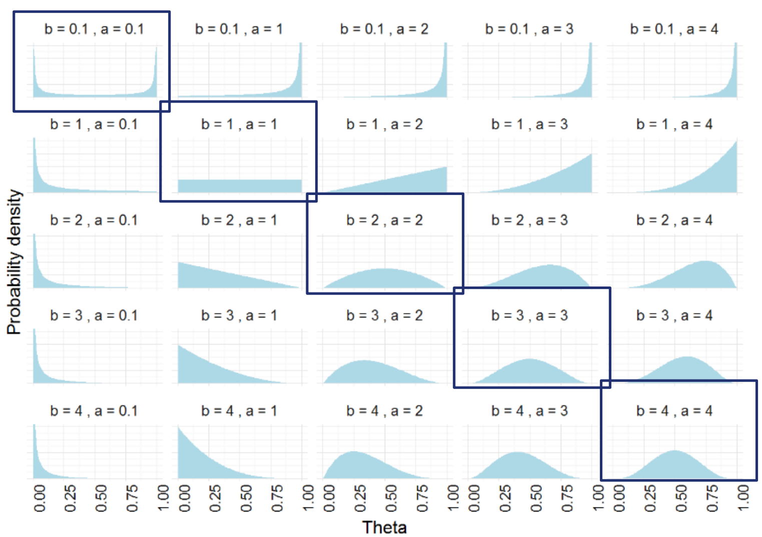
Mixup

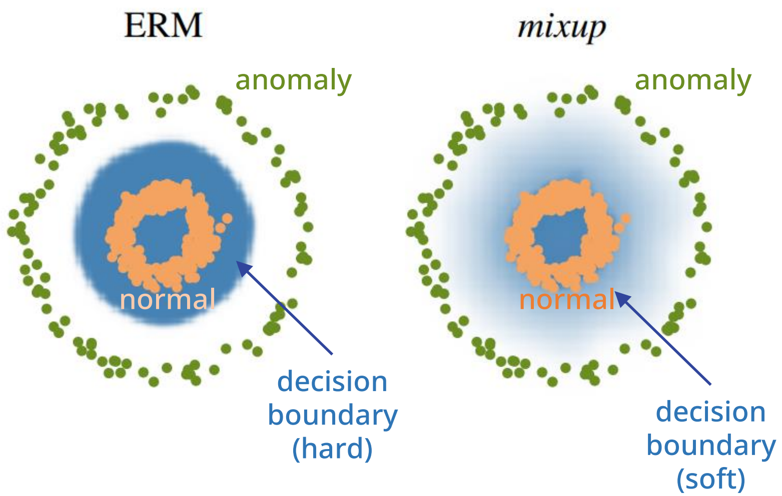
- Choice of λ
  - λ is sampled from Beta distribution

$$\lambda \sim \mathrm{Beta}(\alpha, \alpha)$$

  - α controls the strength of interpolation

- Beta distribution (Beta(a,b))





Mixup: result

- MNIST dataset
  - Without label mixing, Normal: [0,…, 3], Anomaly:[7, 8, 9]