이상탐지(anomaly detection) 프로젝트

Due: 4/29(화) 23:55

개요

- 이상탐지는 입력된 샘플을 정상, 비정상 샘플로 구별하는 문제로 의료 분야 및 제조 산업 현장 등에서 사용되고 있습니다. 이번 프로젝트의 목표는 Fashion MNIST(https://github.com/zalandoresearch/fashion-mnist) 데이터로 이상탐지를 수행하여 높은 정확도를 달성하는 것 입니다. 정확도 비교는 kaggle(https://www.kaggle.com/t/127d2f3975314b1ba8558114efe62dce) 에서 진행되며, leaderboard 에서 순위 및 정확도를 실시간으로 확인할 수 있습니다. 해당 kaggle 사이트는 비공개로 링크를 통해서만 접속할 수 있습니다.

- 본 프로젝트는 3 인/팀으로 수행하는 팀 프로젝트로서 명단은 i-campus 에서 확인할 수 있습니다.

기본코드 제공

- 이상탐지를 수행하는 기본적인 코드를 제공합니다. 해당 코드는 (https://www.kaggle.com/code/yeonghyeon/baseline-uad-w-fashion-mnist-dataset) 를 기반으로 작성되었습니다. Python 및 Pytorch 를 활용한 인공지능 모델 설계 경험이 있다면, 본 Tutorial 은 무시하셔도 무방합니다. 하지만, Pytorch 를 활용한 인공지능 모델 설계 경험이 없다면 본 Tutorial 을 따라가면서 진행하는 것을 추천합니다.

- 본 튜토리얼 코드를 기반으로 모델을 수정하거나 이전에 제안된 SOTA 이상탐지 방법을 사용해도 좋습니다. 하지만, 코드를 제출할 때 submission form 에 맞게 수정하여 제출하여야 합니다.

제출

- 제출할 파일 목록:

  1.dataframe(.csv) / **2**.모델 코드(.ipynb) + weight(.pt/.pth) / **3**.리포트파일 / **4**.환경 세팅 목록(.txt) / **5**.동료 기여도 평가

- **1**.[팀별 제출]-**kaggle 에 dataframe(.csv)파일 제출**
  kaggle 에 Fashion MNIST 데이터의 test data 를 팀이 구축한 모델에 입력하여 정상/비정상 샘플을 판단한 dataframe 파일(.csv)을 제출해야 합니다. dataframe 파일(.csv)을 생성하는 방법은 tutorial 하단에 설명되어 있습니다. 하루에 10 번까지 제출할 수 있으며 팀 대표로 1 명만 제출해야 합니다.

  2.[팀별 제출]-**i-campus 에 모델 코드(.ipynb)및 weight(.pt/.pth) 제출**
  모델 코드(.ipynb) 와 훈련된 weight(.pt/.pth)를 i-campus 에 제출해야 합니다. 모델 코드(.ipynb)는 jupyter notebook 기반으로 생성되야 하며, 제공되는 submission form 에 맞추어서 제출해야 합니다. T.A.가 해당 파일을 실행하여 검토할 예정입니다. 모델 코드(.ipynb)는 본인이 훈련한 모델의 weight(.pt/.pth)를 자동으로 로드하여 훈련없이 실행할 수 있어야 합니다. 실행 시 오류가 발생할 경우 실행되지 않는 것으로 판단합니다.

  3.[팀별 제출]-**i-campus 에 리포트 제출**
  리포트는 i-campus 에 제출해야 합니다. 리포트는 IEEE Conference style(two column format)을 사용하여 2 페이지 이내로 작성해야 하며 네트워크에 대한 설명, AUROC 그래프 등이 포함되어야 합니다. 예를 들면, 해당 네트워크로 설계한 이유 혹은 다른 SOTA 방법을 활용하였다면, 인용문구와 함께 네트워크 어느 부분이 정확도 개선에 기여하는지에 대해서 기술해야 합니다.

  4.[팀별 제출]-**(해당하는 팀만 제출) i-campus 에 환경 세팅 목록 제출**
  이전에 제안된 이상탐지 모델 방법을 활용하였다면, 환경 세팅 목록을(requirement.txt) 제출해야 합니다.

  5.[개인별 제출]-**i-campus 에 동료 기여도 평가지 제출**
  동료 기여도 평가는 제공되는 양식에 작성하여 i-campus 에 개인별로 제출하셔야 합니다.

  (모든 제출 파일의 **due 는 4/29(화) 23:55** 까지입니다.)

점수 반영(Subject to change)

- Kaggle 리더보드 등수 - 70%
- 리포트(모델 코드 포함) - 30%

주의사항

- 다른 SOTA 방법을 활용 했다면, 명시를 해야 합니다. 인용을 하지 않은 경우 문제가 될 수 있습니다.
- 다른 SOTA 방법을 활용 했다면, 코드를 submission form 에 맞추어서 직접 수정 및 훈련해야 합니다. github 에 공유되어 있는 코드 파일들과 사전훈련된 weight 를 그대로 제출하는 것은 허용되지 않습니다.
- 다른 팀의 코드를 재사용 하는 것은 불가합니다.
- 비정상적으로 네트워크의 정확도를 개선하는 경우는(e. g., test data 를 훈련에 사용, dataframe(.csv) 파일을 수동으로 변경하여 제출 등) 0 점 처리 합니다.
- 리포트에 사용한 AUROC 그래프와 코드로 생성된 AUROC 가 다를 경우 문제가 될 수 있습니다.

문의

컴퓨터비전 연구실(산학협력센터 85745)
박현규 T.A.
mjss016@skku.edu

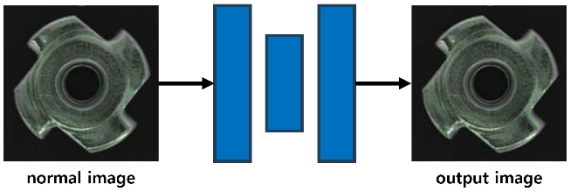
i-campus 쪽지(X) 이메일 문의(O)

Brief tutorial

이상탐지(anomaly detection)

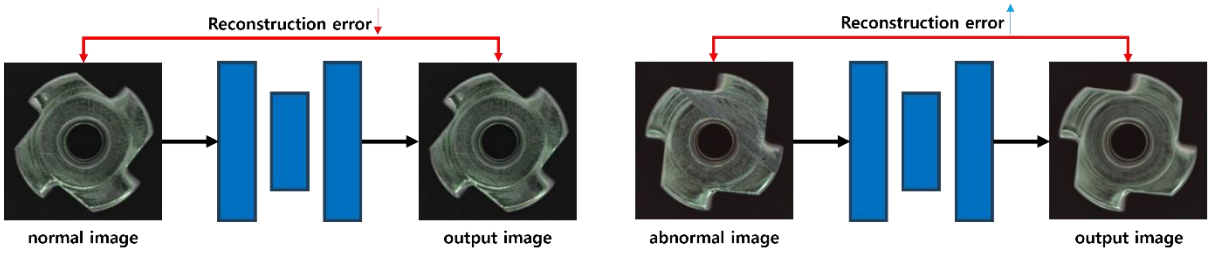
본 tutorial 은 reconstruction-based anomaly detection 을 기반으로 합니다.

< Train >

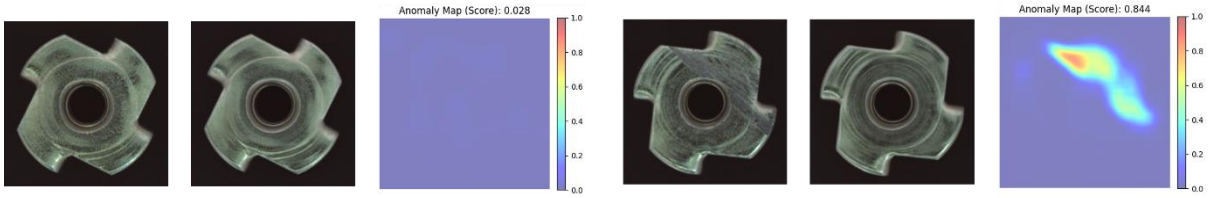


일반적으로 reconstruction-based anomaly detection은 정상 샘플만을 이용하여 오토인코더를 학습시킵니다.

< Test >



이론상 학습된 오토인코더는 훈련 과정에서 정상 샘플만 활용했기 때문에 정상 샘플이 입력되면 정상 샘플로 복원을 할 수 있지만, 훈련 과정에서 사용하지 않은 비정상 샘플이 입력되면 비정상 샘플로 복원을 할 수 없습니다.

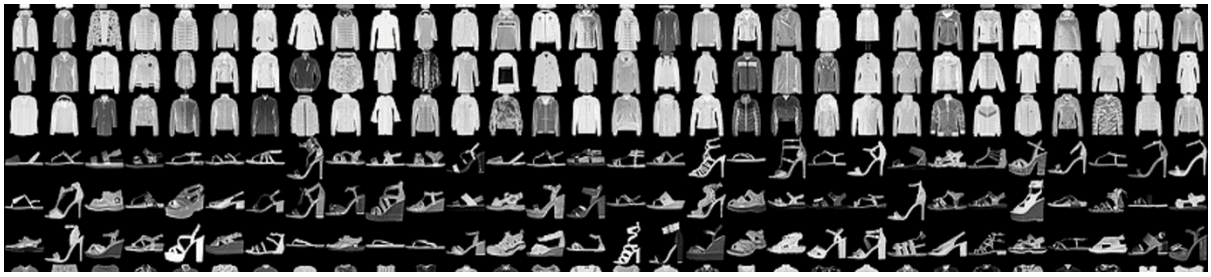


이를 활용하여 입력되는 샘플과 출력되는 샘플의 reconstruction error를 측정하여 정상 샘플과 비정상 샘플을 구별할 수 있습니다.

하지만, 오토인코더는 일반화 성능(generalization ability)이 우수하여 훈련과정에서 활용하지 않은 비정상 샘플이 입력되도 비정상 샘플로 잘 복원하는 문제점이 발생합니다. 따라서 네트워크를 설계할 때 **일반화 성능을 억제하여 정상 샘플은 잘 복원하되 비정상 샘플은 복원하지 못하도록 설계**하는 것이 중요합니다.

데이터셋

Anomaly detection 에서 사용되는 다양한 데이터셋이 있지만, 수강생들의 서버 보유 여부를 고려하여 Fashion MNIST(https://github.com/zalandoresearch/fashion-mnist) 데이터셋을 활용합니다. Fashion MNIST 는 28x28 사이즈의 회색조 이미지며 아래의 표와 같이 **0-9** 까지의 레이블이 존재합니다. 훈련 데이터셋은 **0-9** 번 레이블 6 만장으로 테스트 데이터셋은 0-9 번 레이블 1 만장으로 이루어져있습니다.



2번 레이블(Pullover)을 정상 샘플로서, 4, 6번 레이블은 비정상 샘플로 활용해야 합니다. 네트워크를 훈련할 때 반드시 train 데이터셋의 2번 label만 활용해야 합니다.

Label	0	1	2	3	4	5	6	7	8	9
object	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot

< Fashion MNIST >

Label	2
object	Pullover

< Train dataset >

Label	2	4	6
object	Pullover	Coat	Shirt

< Test dataset >

따라서, **Fashion MNIST의 훈련 데이터셋에서 2번(Pullover) label만 훈련에 활용**해야 하며, **Fashion MNIST의 테스트 데이터셋에서 2(Pullover), 4(Coat), 6번(Shirt) label만 활용**해야 합니다.

Code

In [1]:
```python
# call library
# kaggle 기준 해당 .ipynb파일이 작동하는 것을 확인했습니다. 만약, 로컬환경에서 실행할 예정이라면, 아래 링크를 활용해서 설치해주세요.
# https://pytorch.org/get-started/previous-versions/
# 로컬에서 테스트된 환경은 다음과 같습니다. pytorch 2.0.0 with python=3.9, cuda=11.7, cudnn=8.0, torchvision==0.15.0
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

import torch, torchvision
import torch.nn.functional as F
from torch import nn, optim
from torchvision import transforms, datasets
```

In [2]:
```python
# hyperparameter setting
# https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html
EPOCH = 5
BATCH_SIZE = 4
LEARNING_RATE = 0.005

# Computational device
# Device will be set to GPU if it is available.(you should install valid Pytorch version with CUDA. Otherwise, it will be computed using CPU)
USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
print("Using Device:", DEVICE)
```

```
Using Device: cuda
```

In [3]:
```python
# Fashion MNIST dataset
# https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
# dataset detail
# https://github.com/zalandoresearch/fashion-mnist
trainset = datasets.FashionMNIST(
    root      = './.data/',
    train     = True,
    download  = True,
    transform = transforms.ToTensor()
)
testset = datasets.FashionMNIST(
    root      = './.data/',
    train     = False,
    download  = True,
    transform = transforms.ToTensor()
)
```

In [4]:
```python
# basic autoencoder
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 2),
        )
        self.decoder = nn.Sequential(
            nn.Linear(2, 128),
            nn.ReLU(),
            nn.Linear(128, 28*28),
            nn.Sigmoid(),
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return encoded, decoded
```

In [5]:
```python
# Set for data loader
# https://pytorch.org/tutorials/beginner/basics/data_tutorial.html
SELECT_NORMAL = 2 # Set 2 class as train dataset.
trainset.data = trainset.data[trainset.targets == SELECT_NORMAL]
trainset.targets = trainset.targets[trainset.targets == SELECT_NORMAL]

train_loader = torch.utils.data.DataLoader(
    dataset     = trainset,
    batch_size  = BATCH_SIZE,
    shuffle     = True,
    num_workers = 2
)

test_label = [2,4,6] # Define actual test class that we use
actual_testdata = torch.isin(testset.targets, torch.tensor(test_label))
testset.data = testset.data[actual_testdata]
testset.targets = testset.targets[actual_testdata]

test_loader = torch.utils.data.DataLoader(
    dataset     = testset,
    batch_size  = 1,
    shuffle     = False,
    num_workers = 2
)
```

In [6]:
```
# To visualize training procedure
view_data = []
for i in test_label:
    view_data.append(testset.data[testset.targets == i][0].view(28*28))
view_data = torch.Tensor(np.array(view_data))
view_data = view_data.type(torch.FloatTensor)/255.
```

In [7]:
```
# Initialization of autoencoder and loss function
# https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html
autoencoder = Autoencoder().to(DEVICE) # generating instance of model that you build.
print(autoencoder) # you can check your model
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=LEARNING_RATE) # if you want to utilie other optimizer, replace Adam to other.
criterion = nn.MSELoss() # you can change loss function.
```

```
Autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=2, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=2, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=784, bias=True)
    (3): Sigmoid()
  )
)
```

In [8]:
```
# Training function
# https://pytorch.org/tutorials/beginner/introyt/trainingyt.html
def train(autoencoder, train_loader):
    autoencoder.train()
    for step, (x, label) in enumerate(train_loader):
        x = x.view(-1, 28*28).to(DEVICE)
        y = x.view(-1, 28*28).to(DEVICE)

        encoded, decoded = autoencoder(x)

        loss = criterion(decoded, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

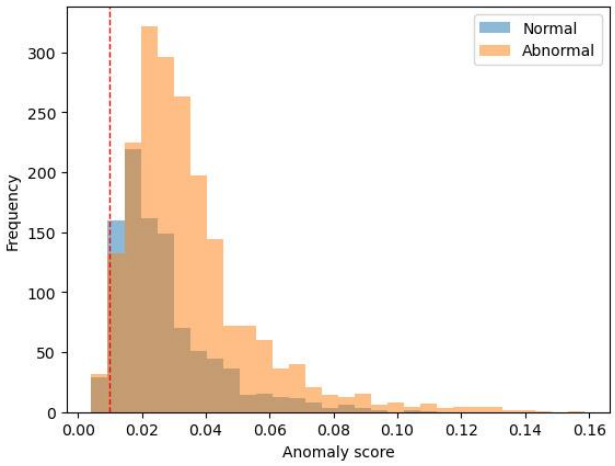
In [9]:
```
# Training process including visualization
for epoch in range(1, EPOCH+1):
    train(autoencoder, train_loader)

    test_x = view_data.to(DEVICE)
    _, decoded_data = autoencoder(test_x)

    f, a = plt.subplots(2, len(test_label), figsize=(len(test_label), 2))
    print("[Epoch {}]".format(epoch))
    for i in range(len(test_label)):
        img = np.reshape(view_data.data.numpy()[i],(28, 28))
        a[0][i].imshow(img, cmap='gray')
        a[0][i].set_xticks(()); a[0][i].set_yticks(())
        if(i == 0): a[0][i].set_ylabel('Input')

    for i in range(len(test_label)):
        img = np.reshape(decoded_data.to("cpu").data.numpy()[i], (28, 28))
        a[1][i].imshow(img, cmap='gray')
        a[1][i].set_xticks(()); a[1][i].set_yticks(())
        if(i == 0): a[1][i].set_ylabel('Output')
    plt.show()
```

[Epoch 1]

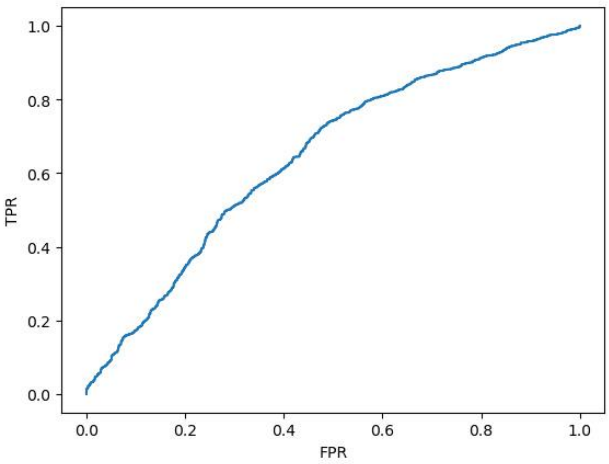
[Epoch 2]

[n [10]:]
```python
# Test
THRESHOLDVAL=0.01 # threshold val
dic_loss = {'id':[], 'label':[], 'score':[],'normal':[]}

count=0
for step, (x, label) in enumerate(test_loader):
    x = x.view(-1, 28*28).to(DEVICE)
    y = x.view(-1, 28*28).to(DEVICE)

    encoded, decoded = autoencoder(x)
    loss = float(criterion(decoded, y).cpu().detach().numpy())
    dic_loss['id'].append(step)
    dic_loss['label'].append(int(label==SELECT_NORMAL)) # 1: normal, 0: abnormal
    dic_loss['score'].append(loss) # abnormal score
    if loss>THRESHOLDVAL: dic_loss['normal'].append('0')
    else: dic_loss['normal'].append('1')
```

[n [11]:]
```python
# Gernerating a plot
arr_label = np.array(dic_loss['label'])
arr_score = np.array(dic_loss['score'])
score_min = arr_score.min()
score_max = arr_score.max()
plt.hist(arr_score[np.where(arr_label == 1)[0]], bins=30, range=(score_min, score_max), alpha=0.5, label='Normal')
plt.hist(arr_score[np.where(arr_label == 0)[0]], bins=30, range=(score_min, score_max), alpha=0.5, label='Abnormal')
plt.xlabel("Anomaly score")
plt.ylabel("Frequency")
plt.axvline(THRESHOLDVAL,0,1, color='red',linestyle='--',linewidth=1)
plt.legend(loc='upper right')
plt.savefig("plot.png")
plt.show()
```



[n [12]:]
```python
# Generating AUROC
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html
fpr, tpr, thresholds = roc_curve(dic_loss['label'], dic_loss['score'], pos_label=0)
plt.plot(fpr, tpr)
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.savefig("auroc.png")
plt.show()
auroc = auc(fpr, tpr)
print("AUROC: {}".format(auroc))
```




```
AUROC: 0.6446462500000001
```

In [13]:
```
# Leveraging the pandas library to convert a dict to a dataframe is more convenient when checking values.
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html
df = pd.DataFrame.from_dict(dic_loss)
df
```

Out[13]:

	id	label	score	normal
0	0	1	0.037945	0
1	1	0	0.030622	0
2	2	0	0.035198	0
3	3	0	0.031617	0
4	4	0	0.019245	0
...	...	...	...	...
2995	2995	0	0.020670	0
2996	2996	0	0.036239	0
2997	2997	1	0.057431	0
2998	2998	0	0.042546	0
2999	2999	0	0.081831	0

3000 rows × 4 columns

In [14]:
```
# In order to submit .csv file to kaggle, dataframe should fit following format
# id[1,2,3...,3000], predicted anomalies[0,1,0....,0]

# 'pop('score,None')' delete one of the item in dict
# 'del df['item']', is also available.

# If you try to remove invalid itmes in the dict, message that you set will be returned.
# set to None, nothing will be returned
dic_loss.pop('score',None)
dic_loss.pop('label',None)
df = pd.DataFrame.from_dict(dic_loss)
df
```

Out[14]:

	id	normal
0	0	0
1	1	0
2	2	0
3	3	0
4	4	0
...	...	...
2995	2995	0
2996	2996	0
2997	2997	0
2998	2998	0
2999	2999	0

3000 rows × 2 columns

In [15]:
```
# to_csv command will convert your dict to .csv file with the name of your teamnumber
# Do not forget to submit the .csv file to kaggle. If you upload .csv file properly to kaggle, you can check your result immediately at the lead
teamnumber = 1 # insert your teamnumber
df.to_csv("result_team{}.csv".format(teamnumber), index =False) # Index should be not included in the .csv file.
```

In [16]:
```
# https://pytorch.org/tutorials/beginner/basics/saveloadrun_tutorial.html
# you can save your weight.
torch.save(autoencoder.state_dict(), 'model_team{}.pth'.format(teamnumber))
```

In [ ]: