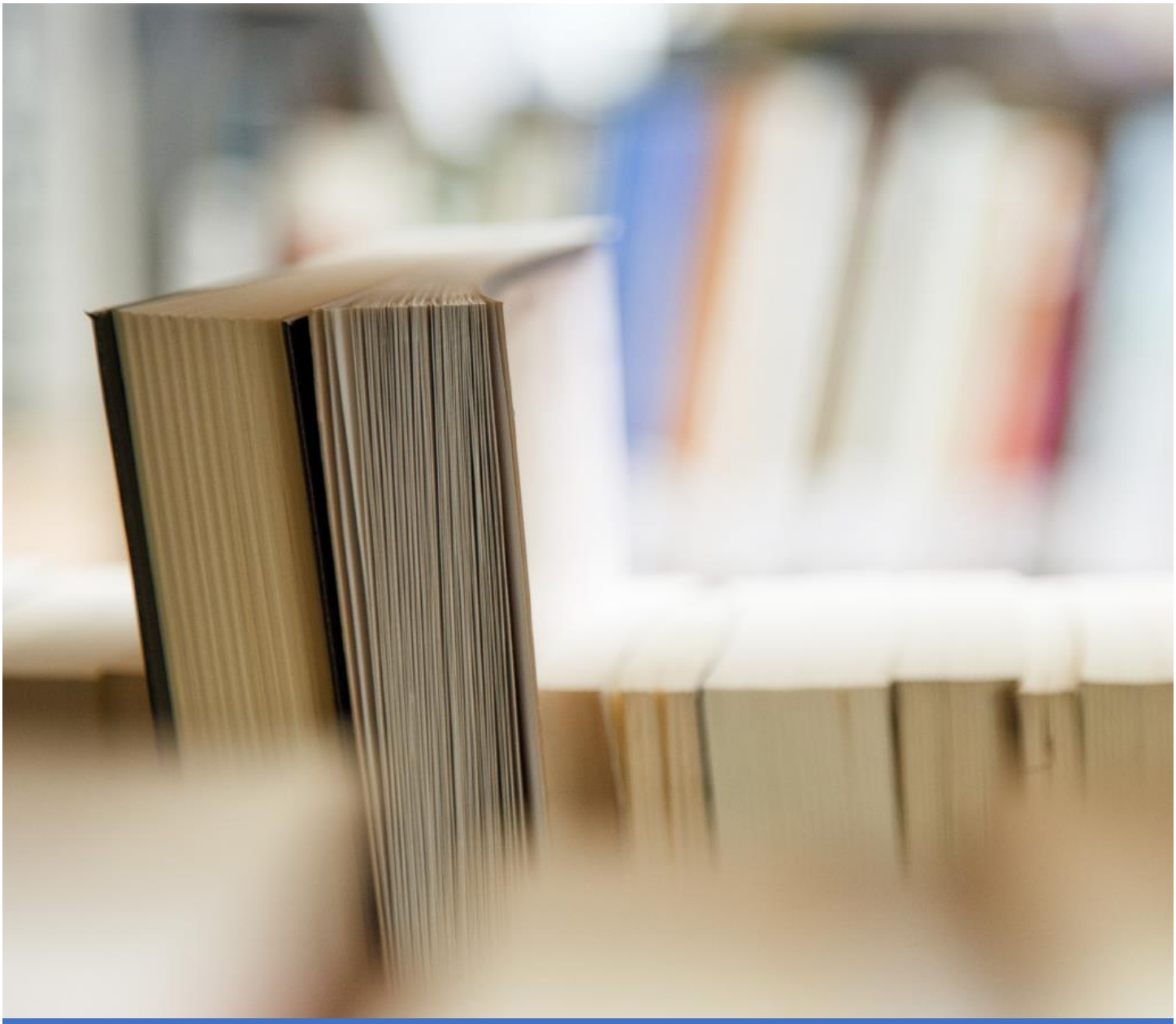


ONE-CLASS CONVOLUTIONAL NEURAL NETWORK

Paper Reproduction

2020021327 조경선[대학원석사과정재학 / 산업경영공학과]

11/15/20

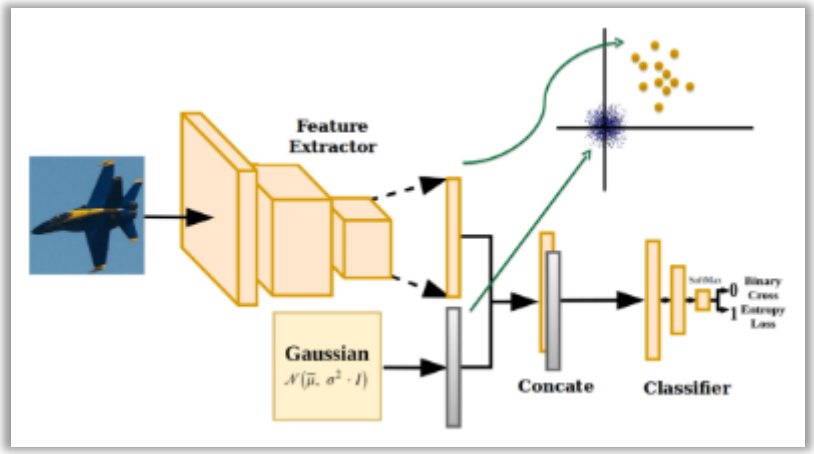
BUSINESS ANALYTICS

Paper Summary

One-class classification 를 위해서 Convolution neural network(CNN)을 활용한 논문이다. 접근 방법의 핵심 특징은 미리 학습된 CNN 을 one-class classification 에 기본 네트워크로 사용한다는 점이다.

논문에서는 사용자 권한, 이상치 탐지에 많은 연관이 있는 UMDAA-02 Face, bnormality-1001, and FounderType-200 datasets 을 통해 one-class CNN 을 평가하였다.

Proposed Approach



전체 네트워크는 기능 추출기 네트워크와 분류기 네트워크로 구성된다. 기능 추출기 네트워크는 기본적으로 입력 대상 클래스 이미지를 feature space 에 포함한다. 추출 된 feature 은 특징 공간의 0 centered 가우시안에서 생성 된 pseudo-negative class data 와 합쳐진다. 합쳐진 feature 은 완전히 연결된 신경망을 특징으로 하는 분류 네트워크로 주입된다. 분류 네트워크는 각각의 feature 를 표현기 위해서 신뢰도 점수로 할당된다. 결과는 1 또는 0 으로 나오게 되는데, 1 은 target class 와 같은 샘플데이터 이고 0 은 그 반대이다. 전체적인 네트워크는 binary cross-entropy loss 를 사용한다.

(A) Feature Extractor

사전 훈련된 CNN 은 feature 추출기로 사용한다. 논문에서는 softmax regression layers 가 제거된 AlexNet 과 VGG15 을 사용하였다. 훈련하는 동안 Convolution layer 를 고정하고 완전히 연결된 레이어만 훈련한다. D 차원의 feature 가 추출됐다면, 가우시안으로 부터 생성된 pseudo-negative data 를 feature 에 추가한다.

(B) Classification Network

원 데이터와 함께 pseudo-negative class data 가 합쳐졌기 때문에, 분류 네트워크의 input 은 배치사이즈 2 이다. Softmax regression layer 를 마지막에 사용하고 그 결과도 2 로 셋팅된다.

(C) Loss Function

다음과 같은 binary cross-entropy loss function 을 사용한다.

$$L_c = -\frac{1}{2K}\sum_{j=1}^{2K}\left(y\log(p) + (1-y)\log(1-p)\right)$$

Optimizer 는 Adam optimizer 를 사용하고, learning rate 는 10^{-4}이다.input image 의 배치사이즈는 64 로 셋팅하였고, 모든 실험의 $\bar{\mu}$ 와 σ 는 $\bar{0}$과 0.01 이다.

Experiments

논문에서 User Active Authentication 문제를 해결하기 위해 UMDAA-02 Face, Abnormality Detection 문제를 위한 abnormality-1001, Novelty Detection 문제 해결을 위한 FounderType-200 datasets 을 사용하였다. 비교군으로 사용한 one-class classification method 는 다음과 같다.
OC-SVM, BSVM, MPM, SVDD, OC-NN, OC-SVM+

이 논문에서 사용한 OC-CNN 기법이 꽤나 User Active Authentication 문제에서 우수한 것으로 나타났다. 그 밖에 문제에 대해서도 다른 기법들에 비해 높은 우수성을 나타내었다.

TABLE I
COMPARISON BETWEEN THE PROPOSED AND OTHER METHODS USING **ALEXNET** AS THE BASE NETWORK. RESULTS ARE MEAN OF PERFORMANCE ON ALL CLASSES. BEST AND THE SECOND BEST PERFORMANCE ARE HIGHLIGHTED IN BOLD FONTS AND ITALICS, RESPECTIVELY

Dataset	OC-SVM	BSVM	MPM	SVDD	OC-NN-lin	OC-NN-sig	OC-NN-relu	OC-CNN	OC-SVM+
Abnormality-1001	0.6057	0.6126	0.5806	0.7873	0.8090	0.6391	0.7372	*0.8264*	**0.8334**
UMDAA-02 Face	0.5746	0.5660	0.5418	0.6448	0.6173	0.6452	0.5943	**0.7017**	*0.6736*
FounderType-200	0.7124	0.7067	0.7085	0.8998	0.8884	0.8696	0.8505	*0.9303*	**0.9350**

TABLE II
COMPARISON BETWEEN PROPOSED AND OTHER METHODS USING **VGG16** AS THE BASE NETWORK. RESULTS ARE MEAN OF PERFORMANCE ON ALL CLASSES. BEST AND THE SECOND BEST PERFORMANCE ARE HIGHLIGHTED IN BOLD FONTS AND ITALICS, RESPECTIVELY

Dataset	OC-SVM	BSVM	MPM	SVDD	OC-NN-lin	OC-NN-sig	OC-NN-relu	OC-CNN	OC-SVM+
Abnormality-1001	0.6475	0.6418	0.5909	0.8031	0.7740	0.8373	0.5821	*0.8424*	**0.8460**
UMDAA-02 Face	0.5829	0.5751	0.5473	0.6424	0.6193	0.6200	0.5788	**0.7350**	*0.7230*
FounderType-200	0.7490	0.7067	0.7444	0.8885	0.8986	0.8677	0.8506	*0.9290*	**0.9419**

Reproduction

논문에서 많은 비교군 method 를 사용하였다. 그 중에서 oc-cnn 과 oc-svm 의 비교를 통해 논문을 구현하고 비교해 보았다. 사용한 데이터는 이상치 감지에 많이 사용되는 Abnomality-1001 을 이용하였고, 자동차와 자동차가 아닌 것을 구별하도록 Data 를 구성하였다.

Data Preparation

```
from IPython.display import Image, display
import numpy as np
import os
from os.path import join
from PIL import ImageFile
import pandas as pd
from matplotlib import cm
import seaborn as sns
from keras.models import Sequential
from keras.layers import Dense, Flatten, GlobalAveragePooling2D
from keras.applications.resnet50 import preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import ResNet50
from keras.preprocessing.image import load_img, img_to_array
from sklearn.metrics import mean_squared_error, mean_absolute_error,
roc_auc_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import IsolationForest
from sklearn import svm
from sklearn.mixture import GaussianMixture
from sklearn.isotonic import IsotonicRegression
import re

ImageFile.LOAD_TRUNCATED_IMAGES = True
plt.style.use('fivethirtyeight')
%matplotlib inline

# import car images from natural images
train_img_dir_n = "../../Abnormal_1001/Car"
train_img_paths_n = [join(train_img_dir_n,filename) for filename in
os.listdir(train_img_dir_n)]

# split cars data into train, test, and val
train_img_paths, test_img_paths_car = train_test_split(train_img_paths_n,
test_size=0.25, random_state=42)
train_img_paths, val_img_paths_car = train_test_split(train_img_paths,
test_size=0.25, random_state=42)

#   import ~car images
natural_images_path = "../../Abnormal_1001/"
test_img_paths_no_car = []
for d in [d for d in os.listdir(natural_images_path) if d != "Car"]:
    test_img_dir_na = natural_images_path + d
    test_img_paths_no_car.append([join(test_img_dir_na, filename) for
filename in os.listdir(test_img_dir_na)])

test_img_paths_no_car_flat = [item for sublist in test_img_paths_no_car for
item in sublist]
test_img_paths_no_car, val_img_paths_no_car =
train_test_split(test_img_paths_no_car_flat, test_size=0.25,
                                                    random_state=42)

def natural_img_dir(image_path):
    path_regex = r"Abnormal_1001\/(\w*)"
    if 'Abnormal_1001' in image_path:
        return re.findall(path_regex,image_path,re.MULTILINE)[0].strip()
```

OC-SVM

```
# Train classifier and obtain predictions for OC-SVM
oc_svm_clf = svm.OneClassSVM(gamma=0.001, kernel='rbf', nu=0.08)  #
Obtained using grid search
if_clf = IsolationForest(contamination=0.08, max_features=1.0,
max_samples=1.0, n_estimators=40)  # Obtained using grid search

oc_svm_clf.fit(X_train)
if_clf.fit(X_train)

oc_svm_preds = oc_svm_clf.predict(X_test)
if_preds = if_clf.predict(X_test)

svm_if_results=pd.DataFrame({
  'path': all_test_paths,
  'oc_svm_preds': [0 if x == -1 else 1 for x in oc_svm_preds],
  'if_preds': [0 if x == -1 else 1 for x in if_preds]
})

svm_if_results=svm_if_results.merge(test_path_df)
svm_if_results.head()
```

OC-CNN

```
import datetime
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from keras import Model
from keras.models import Sequential
from keras.utils import to_categorical
from keras.losses import categorical_crossentropy
# from keras.preprocessing.image import Im
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

from tensorflow.keras import datasets, layers, models

X=X_train.reshape(X_train.shape[0],56,56,48).astype('float32')
Y=X_test.reshape(X_test.shape[0],56,56,48).astype('float32')
num_classes = 1

# AlexNet model
alexnet_model = models.Sequential([
    layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
activation='relu', input_shape=(56,56,48)),
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),
activation='relu', padding="same"),
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(4,4), strides=(2,2)),
    layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"),
    layers.BatchNormalization(),
    layers.Conv2D(filters=384, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"),
    layers.BatchNormalization(),
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='softmax')
])

alexnet_model.compile(loss='binary_crossentropy',
            optimizer=tf.keras.optimizers.Adam(0.0001),
            metrics=['accuracy'])
alexnet_model.summary()

history = alexnet_model.fit(X, np.ones((X.shape[0],)) , batch_size = 64,
validation_split = 0.2, epochs = 100, verbose = 0)

test_loss, test_acc = alexnet_model.evaluate(Y,
np.array(test_path_df['is_car']), verbose=2)
```

Conclusion

결론적으로 논문의 결과보다 OC-SVM 은 더 좋은 수치를 보였고, 구현한 OC-CNN 은 너무도 나쁜 결과를 나타냈다.

	OC-SVM	OC-CNN
Accuracy	0.96	0.04

여러가지 이유가 있겠지만 가장 큰 원인은 구현의 미흡함이라고 생각한다. 이번 논문에서는 정확한 Parameter 까지 제시해 주었고, 또한 이론이 많지 않는 논문이어서 논문을 이해하였다고 생각한다. 하지만 논문에서 제시한 정규화, CNN 등 많은 기법을 한번에 구현하기에는 나의 실력이 아직 많이 부족하고 deep learning 등에 대한 이해도가 많이 떨어짐을 실감하였다.

모든 기법을 다 충족 시키기에는 좀 더 learning time 이 소요될 것이다.

이번 구현은 이 정도에서 멈추지만, 좀 더 실력을 쌓은 후에 다시 한 번 더 좋은 구현모델을 만들어 보겠다는 마음으로 마무리를 짓는다.