Supervised Learning for Lemon Quality Classification

Minho Kim, 18101308

Department of Electrical and Information Engineering

1. Introduction

The task of this project is to solve the binary classification problem of classifying high-quality lemons and p oor-quality lemons. The problem is a simple binary classification problem, but its purpose is simi-lar to the an omaly detection problem, especially the supervised anomaly detection problem.

Anomaly detection is a field that is actively being studied for finding defective products in manufacturing fa cilities or analyzing medical images. The AI model that I wanted to implement in this project is a very simple version of the model that picks out poor quality lemons from lemons in a facility that makes lemons a commo dity.

I used “Lemon Quality Dataset” [1], which has 950 images of bad quality lemons and 1,124 images of good quality lemons. However, When I used these all images for binary classification, it was so easy to achieve 100% accuracy. So, I removed several bad quality images for making other setting like Table1.
By splitting dataset like Table1, we can look this project more like supervised anomaly detection.

In supervised anomaly detection, the main topic to solve the problem is “how deal with class-imbalance”.
So, first, I built the basic CNN model that has only convolution layers and fully connected layers. And then, In order to solve this problem, I added a technique that uses feature distance based metric that I intuitively thought that will work well.

2. Details of Approach

2.1 CNN model

First, I built CNN models by adding conv2D layers, FC-layers with activation function ReLU and last FC-la yer with activation function sigmoid (Fig. 1). And by adding Batch Normalization into the two of FC-layers, I achieved better performance compare with no batch normalization.

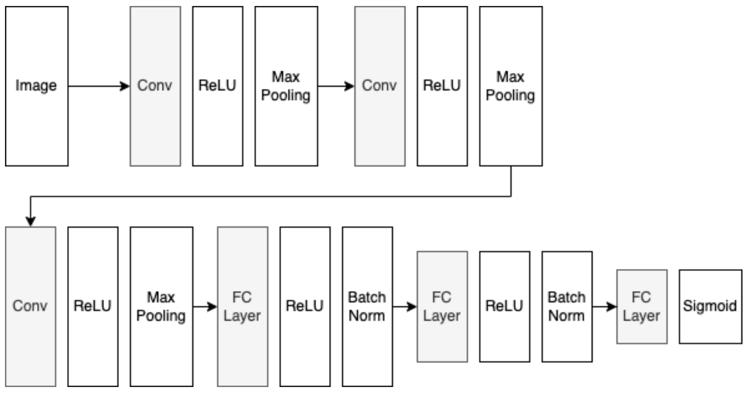

Figure1: Basic network

And optimize the model during training by using Adam optimizer with weight decay technique, and BCE Lo ss (binary cross entropy loss) [2],

$$BCELoss(x) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\big(h(x_i;\theta)\big) + (1-y_i)\log\left(1-h(x_i;\theta)\right) \quad (1)$$

as a objective function because there are only two of labels 0 and 1.

2.2 Feature Distance based Metric

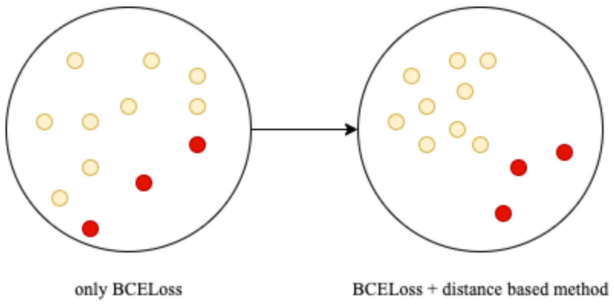

Figure2: feature distance based metric

Intuitively, I guessed if the model could gather the feature vectors extracted from images of "good quality in one point, the generalization performance of the model will be greater than before like figure2. So first, I considered how to define the distance between vectors and which vector to define as a reference vector. And then, I decided to use l2-distance (Euclidean distance) [3] as a metric for calculating distance between vectors,

$$l_2 - distance = |\boldsymbol{f}|_2^2 = f_1^2 + f_2^2 + f_3^2 + \cdots + f_n^2 \quad (2)$$
$$(\boldsymbol{f} : feature\ vectors\ from\ model)$$

and defined centroid point of the vectors extracted from images of good quality as a reference vector,

$$centroid\ vector = \frac{1}{N}\sum_{i\in Y_1}^{N} \boldsymbol{f}_i \quad (3)$$

where Y_1 *is the set of indexes of data with label* 1 (good quality) and N is the size of Y_1.

2.3 Training

```
Training loop (pseudo code)
for each epoch:

    c = centroid vector by (3)

    for each batch:
        bce = BCELoss (1)
        l2d = l2-distance (2) by using c
        loss = bce + l2d

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

Before using data of batch unit, first I calculated centroid vector by (3). And the model is updated by calculating BCELoss and l2-distance loss by using centroid vector and each vector from last to second layer of the network. Note that the l2-distance metric is calculated only for samples with label 1. Lemon of bad quality would be much more diverse than data, and I thought that generalization performance would be very poor if I applied this method to bad quality using only a small amount of data.

3. Results

For experiments, I used Lemon-Quality dataset from Kaggle. And I used 828 images for training and 674 images for testing. There are two labels which means quality, “good” and “bad”.

Table 1. dataset split

	Lemon Quality	
	good	bad
train	788	40
test	337	337

Because the problem and data were so simple that it did not take much effort to achieve 100% accuracy, So I removed a large amount of bad quality images from the trainset and created a properly challenging problem. There are 788 “good” images and 40 “bad” images (Table 1) and 337 “good” images and 337 “bad” images.

And there are scores of three methods that I implemented.

Methods	accuracy (%)
Basic CNN	64.4
Basic CNN + Batch Norm	92.1
Basic CNN + Batch Norm + l2 distance loss	95

Table2. accuracy score of each version of models

4. Discussion and Conclusions

To visualize the operation of methods that I implemented, I drew a t-SNE to show how the score vector of each image is projected onto a two-dimensional plane.

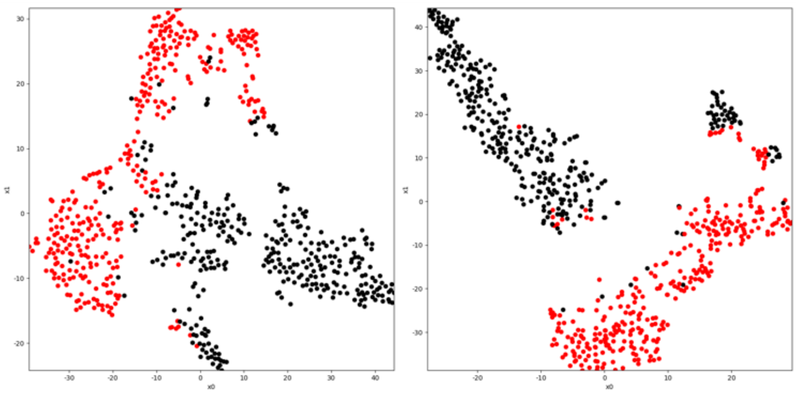

Figure3: t-SNE of each method. From the left, it's CNN+BN, CNN+BN+l2-distance loss.
(Red points mean label 1 and black points mean label 0.)

Looking at t-SNE, the goal to learn while reducing the distance between feature vectors was achieved to some extent and showed small performance improvement. By adding an objective function to increase vector similarity of samples corresponding to relatively many classes, the model could be learned to create a better organized distribution for bad quality lemons belonging to 337 test sets other than 40 bad quality lemons included in the trainset.

Why was l2-Distance loss not applied to bad quality lemon?

The reason why the distance metric for good quality lemon was not applied to lemon of bad quality is that learning by collecting only 40 Feature vectors would worsen the overfitting problem of machine learning and adversely affect generalization performance. In fact, the experiment showed low accuracy of about 87%.

5. References

[1] Kaggle, Lemon Quality Dataset : https://www.kaggle.com/datasets/yusufemir/lemon-quality-dataset

[2] Binary Cross Entropy Loss formular and code: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

[3] L2-norm formular: https://mathworld.wolfram.com/L2-Norm.html

[4] t-SNE code from scikit-lerarn: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html