ECE272B Homework 2
Author: Zhengqi Yang
TA: Min Jian Yang
Date: 02/09/2020

Introduction:

In this lab, we were required to automate the visual inspection process that identifies wafer images falling into a certain defect class via Convolutional Neural Network. We attempted different neural network architectures, parameters, optimization techniques, and image augmentation techniques to find a better model to fit this classification task. The overall accuracy is around 93% as shown in the CodaLab competition submissions. The best model so far achieved 95.75% accuracy by using three convolutional layers followed by max pooling and 0.25 dropout and vertical flipping as the image augmentation technique.

Data Preprocessing:

At first, we preprocessed the training images and the labels. We reshaped the training data to 400x64x64x1. Since there are 6 different defect classes and we expected that the neural network returned 6 scores for each of the class, for each classified image we would like to have each class matching with one score and also know the ground truth class. Hence, we cannot just convert the string labels into integers. Because of this, we used a method called One Hot Encoding to represent categorical variables as binary vectors. To do this, we converted the string labels into integers first. Then each integer value is represented as a binary vector that is all zero values except the index of the integer marked with a 1. This is shown in Figure 1.

```
[5] # Preprocessing Training label set Y


    def prepare_labels(labels):
        ohe = OneHotEncoder()
        # reshape labels to be 2D array
        labels = labels.reshape(len(labels),1)
        ohe.fit(labels)
        labels_enc = ohe.transform(labels).toarray()
        return labels_enc


    y_processed = prepare_labels(train_y)
```

Figure 1: Label Preprocessing – One Hot Encoding.

Secondly, we split the training set and validation set with a 7:3 ratio. Initially, we used 8:2 but we considered that there would be only 80 validation images and the experiments with 7:3 showed better results than the ones with 8:2. Hence, we decided to use 7:3 splitting.

CNN Architecture:

Layer number:

Since the wafer image size is 64x64x1 and the training dataset size is 280, we cannot construct a very deep neural network in order to avoid overfitting and vanishing gradients

problems. Initially, we tried 4 convolutional layers and turned out that training accuracy kept improving but the validation accuracy was not improving at all after a few epochs. We concluded that the overfitting occurred and 4 convolutional layers were too deep for this dataset. Hence, we decided to use 3 convolutional layers. After switching to 3 convolutional layers, the overfitting issue was fixed.

Since we decided the basic architecture of the neural network, we started play around the parameters in convolutional layers, dropout rate, epochs, batch size, image augmentation etc.

Filter Number:

At first, we tried the different numbers of filters for each convolutional layer. We started with 4 filters for the first layer and times two for the next convolutional layer. The accuracy turned out to be around 90%, which is reasonable. However, when we started with 32 filters for the first convolutional layer and ended up with 128 filters for the last convolutional layer, the overfitting occurred again. Hence, we decided to start with 16 for the first convolutional layer, 32 for the second one, and 64 for the last one.

Kernel and Bias Initializer:

The decision to use 'RandomNormal' as the kernel and bias initializer was mostly based on the experiments. We tried 'glorot_uniform', 'zeros', and 'ones'. It seemed that with the 'RandomNormal' the neural network consistently generated more robust results.

Activation function:

We used the rectified linear unit as the activation function at the beginning since the activation function is computational efficient and good at overcoming slow weights updating issues.

Dropout rate, epoch, and batch size:

For these three parameters, the decision was mostly based the observations for the training accuracy and validation accuracy.

The dropout layer is necessary because it can effectively prevent the overfitting issue for this case. Since this is not a very deep network and the number of filters is not large, the dropout rate after each convolutional computation should not be too large. Hence, we used a middle number 0.25. However, considering the large number of parameters at the last layer, we used 0.5 as the dropout rate for the last layer.

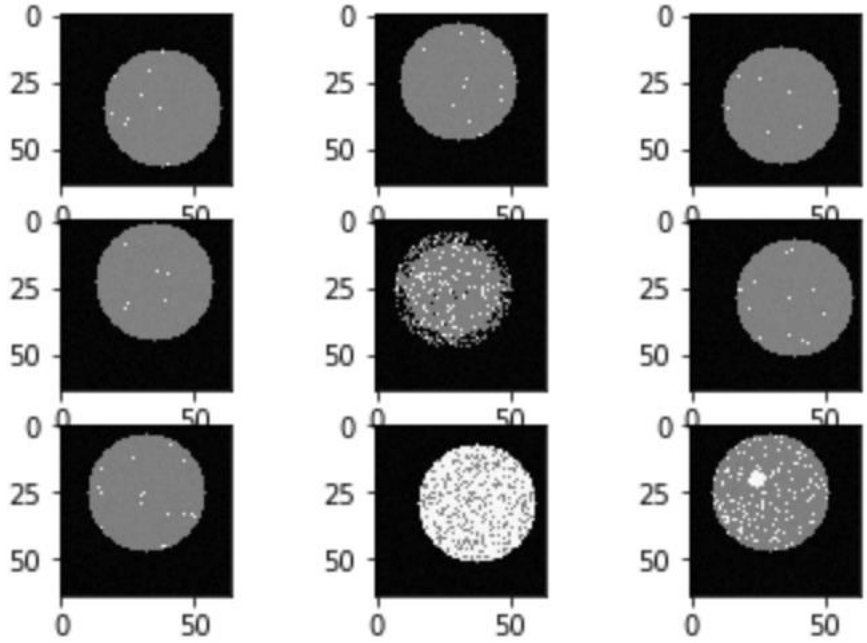
Initially, we trained the neural network for 100 epochs. Since it was not very obvious to notice that the validation accuracy stopped improving at a certain epoch, we decided to train 200 epochs. However, we noticed that after around 150^{th} epoch the validation accuracy did not improve that much but the validation accuracy was still improving a bit between 100^{th} and 150^{th} epochs. Therefore, we decided to train the neural network for 150 epochs.

Since the training data size is 280 and the validation data size is 120, the batch size should be around 10-20. With a few experiments, we noticed that the overall accuracy was slightly higher when we used 10 images for a batch.

Image Augmentation:

The available image augmentation techniques are the following: setting input mean to 0 over the dataset or each sample, random rotation, shearing, zooming, shifting, flipping, zca

whitening etc. Considering the wafer image is grey-scale and the shape of each image is circle, we thought the zca_whitening, rotation, standardization would not help training that much. This was proved by the experiments with these techniques. Also, we experimented with shearing and zooming. Experiments with shearing did produce slightly better results, which improved the overall accuracy around 93%. However, experiments with zooming did not show that much improvements. The best image augmentation technique we attempted so far was flipping. It improved the overall accuracy to around 94%-95%. Hence, we used vertical flipping as the image augmentation technique. Since the property of this dataset rules out many techniques, we did not use multiple techniques as a combination. Example vertical flipped wafer images.



Results:

The following is the final neural network design, confusion matrix, classification reports, loss plot and training vs validation accuracy plot. By observing these two plots, we can conclude that the overfitting did not occur and the model was trained within our expectation. The performance of the model was verified by the f1-score from the classification reports and the layouts of the confusion matrix.

Layer	Parameter (#filters, kernel size, stride, padding, activation, kernel initializer, bias initializer)
Layer 1	16, 3x3, (1,1), same, relu, RandomNormal, RandomNormal, input shape (64,64,1)
	Max pooling with pool size (2,2)
	Dropout 0.25
Layer 2	32, 3x3, (1,1), same, relu, RandomNormal, RandomNormal
	Max pooling with pool size (2,2)
	Dropout 0.25
Layer 3	64, 3x3, (1,1), same, relu, RandomNormal, RandomNormal
	Max pooling with pool size (2,2)

	Dropout 0.25
Layer 5	Flatten
Layer 6	Dense, 128, relu
	Dropout 0.5
Layer 7	FC, Softmax generating 6 classes results

Confusion Matrix

```
array([[ 3,  0,  0,  2,  0,  0],
       [ 0,  7,  0,  0,  0,  0],
       [ 0,  0,  7,  0,  0,  0],
       [ 0,  0,  0,  8,  0,  0],
       [ 0,  0,  0,  0, 91,  0],
       [ 0,  0,  0,  0,  0,  2]])
```

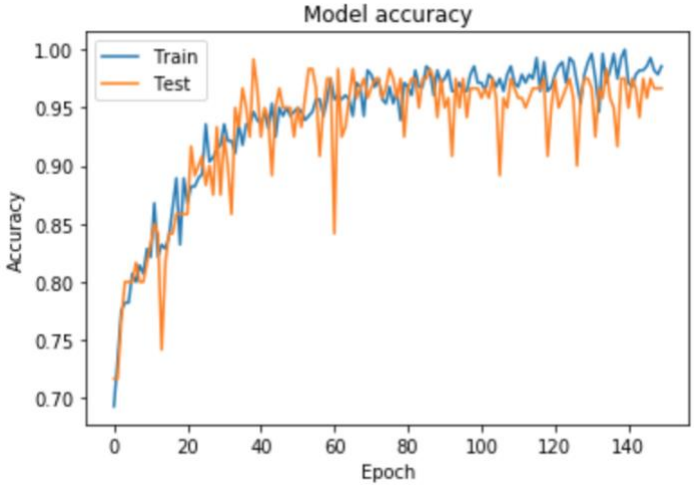
Classification Report

```
              precision    recall  f1-score   support

       crack       1.00      0.60      0.75         5
      deform       1.00      1.00      1.00         7
        edge       1.00      1.00      1.00         7
      nodule       0.80      1.00      0.89         8
        pass       1.00      1.00      1.00        91
  total_loss       1.00      1.00      1.00         2

    accuracy                           0.98       120
   macro avg       0.97      0.93      0.94       120
weighted avg       0.99      0.98      0.98       120
```

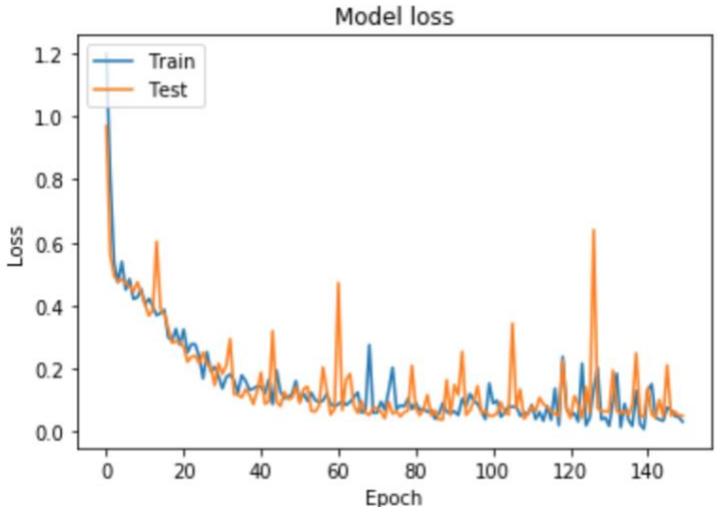
Last Epoch

```
Epoch 150/150
28/28 [==============================] - 0s 7ms/step - loss: 0.0532 - acc:
0.9821 - val_loss: 0.0305 - val_acc: 0.9833
```

Training Accuracy vs Validation Accuracy



Loss


Model loss
Train
Test
Loss
Epoch
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
140