Defect Detection in Semi-conductor Images

Kai Wang
Department of Computer Science
HKUST
Clear Water Bay, Kowloon
kwangbn@conncet.ust.hk

Weizhen Ding
Department of Computer Science
HKUST
Clear Water Bay, Kowloon
wdingai@conncet.ust.hk

Abstract

CNN has proved to be effective in computer vision tasks. In this project, we build a light CNN model from scratch to tackle the defect detection task. We conduct several experiments to test the performance of the model. The experiments demonstrate that the proposed model can achieve superior performance in limited training epochs. We further investigated the mispredicted data and explained the reasons for the failure. Based on the investigation, we also propose several plans that can improve the performance of the model on the task.

1 Introduction

Deep learning is one of the most popular topics in computer vision research due to its great success in various image analysis tasks. Among all the proposed deep learning models, Convolutional Neural Network (CNN) is the most prevalent one, drawing attention from different research circles. Many CNN-based models have been proposed in the last decade, including VGG, ResNet, YOLO, StyleGAN[1]–[5]. These models are useful in image classification, object detection, and image generation. Although these models are trained based on a specific dataset, usually they can be adapted for other tasks; such adaptation is defined as transfer learning. Utilizing existing CNN models by transfer learning is a default option in computer vision tasks since designing and training a CNN model from scratch can be time-consuming and tricky.

However, we cannot adapt transfer learning in this project since the dataset is unique, and we failed to use any pre-trained models to carry out the task. In the project, we are given a dataset of Semi-conductor device images from Nexperia, which is a big semi-conductor company. These images contain two types of semi-conductor devices, good ones and unqualified (defect) ones. Unlike the conventional image classification tasks, the difference between the image of these two types of devices is minimal, therefore bringing about a challenge in exploiting existing models. In this sense, we develop a light CNN model based on VGG16. We adopt the-state-of-the-art optimization techniques to enable the model to achieve good accuracy in minimal training epochs.

The rest of the paper is organized as follows. We first introduce the basic statistical information of the task's dataset and the data augmentation techniques we implemented in the experiment in Section 2. Section 3 described the proposed model's architecture for the defect detection task and the ideas behind the model. Section 4 explains the metrics we use to measure the performance of the proposed model. In Section 5, we demonstrate the result of the experiment. In Section 6 and Section 7, we discussed the project's findings and the potential future work for the task.

2 Data preprocessing

In the data preprocessing stage, we first conduct exploratory data analysis (EDA) and then perform data augmentation based on EDA.

2.1 Exploratory data analysis

In the given dataset, we have 34459 images of the semi-conductor with a size of around 267×275. Very few of the images have a slightly different image size. We think such differences will not affect the image classification. Thus we ignore the difference in the following model design.

Among the images, roughly only ¼ of them are labeled as defective devices, while the other ¾ are labeled as good devices, as showed in Figure 1. Other than the given image, an extra defect area excel is given, which indicates the possible defect area for the corresponding image. In the excel file, there are 6696 records of the images, which is 343 fewer than the images labeled as a defect. We observe the 343 images carefully and find that most of these images are disputable to be classified as unqualified devices since they look like the good ones. Therefore, we remove these images from our dataset to avoid contamination of our model.

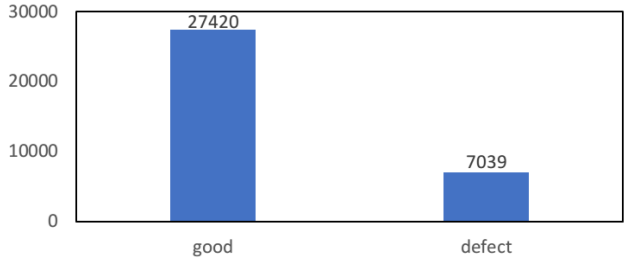

Figure 1: Number of images of the dataset

2.1 Data augmentation

Data augmentation is a standard operation in image classification tasks to avoid overfitting, especially when the data source is not big enough. Various data augmentation techniques have been developed for computer vision tasks. Basically, there are two types of approaches, basic data manipulation, and deep learning-based augmentation[6]. In this project, the data size is sufficient with respect to the image size—however, the actual difficulty lies in the imbalance of the data distribution. Theoretically, the extra amount of good images could result in a bias in the model. Thus, we aim to increase the number of images with the "defect" label by resampling them to be comparable to the images with the "good" label.

3 Models

We designed the CNN model based on the architecture of VGG16. There are 11 layers in our model, and these layers can be divided into four units, three convolutional units, and a fully connected unit (see Figure 2).

The first convolutional unit comprises two layers, a convolutional layer, and a max-pooling layer. In the convolutional layer, we maintain the kernel size as 3×3 but modify the input image size from 224×224 to 275×267 so that it is consistent with the majority of the given dataset. By utilizing the full image size, we avoid cropping the images before inputting them into the model. We also decrease the number of filters from 64 to 16 since we find that the 16-fileter model has higher validation accuracy than the 64-filter model in the pilot experiment stage. We speculate the reason is that the 16-filter model is easier to train. Moreover, fewer input filters indicate fewer parameters, therefore saving the computation overhead of the model. Followed by the convolutional layer, we have a max-pooling layer to reduce the size of the previous layer's output.

We have 2 and 3 convolutional layers respectively in the next two convolutional units to detect more complex features. We also apply max-pooling layers to extract essential features

while reducing the model size. In the fully connected unit, the 3-dimensional feature cubes are flattened into a 1-dimensional feature vector. Then we have 2 fully connected dense layers. Unlike the conventional VGG model, we adapt batch normalization and dropout techniques in the model. These techniques can correct overfitting problems and accelerate the training process.

In summary, the proposed model is a simplified VGG model that adapt the state-of-the-art optimization techniques.

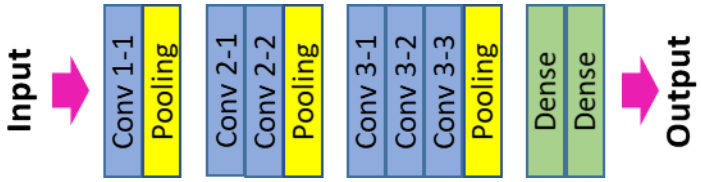

Figure 2: Architecture of the proposed model

4 Metrics

We applied several metrics in the experiment to test the proposed model's performance, including precision, recall, f1-score, support, and confusion matrix. Precision measures the ratio of correctly predicted negatives with true positive and false positives. Recall measures the ratio of correctly predicted positives with the true positives and false negatives. F1 score is the weighted average of precision and recall. It is more robust than accuracy when the data distribution is imbalanced data. The confusion matrix consists of true positive, true negative, false positive, and false negative. It is used to compute precision, recall, and F1 score.

5 Experiment

In the experiment stage, we trained the proposed model with the original data and augmented data (by resampling). We divided the dataset into two parts, training data, and validation data. Training data account for 90% of the whole dataset, and validation data account for 10%. The experiment is conducted on a 24-core AMD Ryzen Threadripper 3960X CPU and 64GB Ram. The performance is measure by metrics that measure the correctness of the prediction.

5.2 Model training

The proposed CNN model achieves the overall validation accuracy of 98.87% after 20 epochs of training on the augmented data. The model that runs on the originated data has slightly worse performance, yet the accuracy is still above 98% (see figure 3). Due to the limitation of computational resources (no GPU), a single epoch takes around half on our device, so we stop the training in the 20th epoch. However, the validation accuracy trend indicates that we can achieve even better performance with further training.

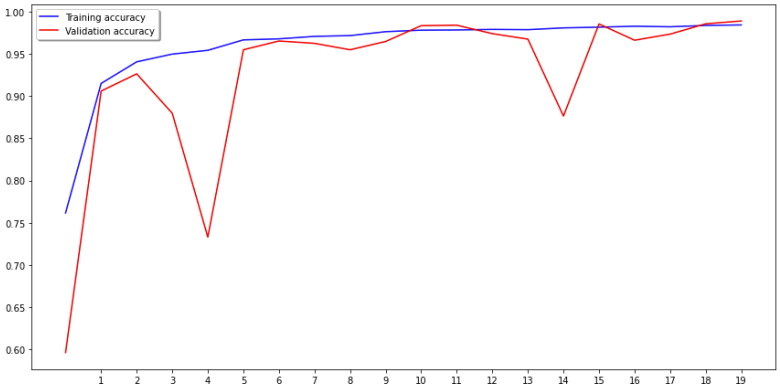

Figure 3: Training accuracy and validation accuracy

5.2 Effectiveness of data augmentation

The performance of the models trained on different datasets with different metrics is listed in Table 1. Both the models have been trained for 20 epochs with the same hyperparameters. As we can see, though both models have similar overall accuracy, the value of precision, recall, and F1 score are significantly different. By resampling the data with the "defect" label, we have a 2.36% gain in precision, 5.29% gain in the recall and 3.89% gain in F1 score for the defect class. That is because the extra data eliminate the bias of the model towards the dominant class. It is worth noting that we even have a 0.11% gain in F1 score for the good class, which indicates that there is room for improvement of the model given further training.

Table 1: Performance of the proposed model trained on different datasets

	label	precision	recall	F1 score	support	overall accuracy
original data	defect	96.20%	93.78%	94.97%	675	98.04%
	good	98.47%	99.09%	98.78%	2737	
augmented data	defect	98.66%	99.06%	98.86%	2670	98.87%
	good	99.09%	98.69%	98.89%	2751	

5.3 Investigation of the false prediction

The model trained with augmentation achieved 98.87% accuracy, false predicting 61 out of the 5421 images according to the given ground truth (see Figure 4). To understand how the model failed to fit the 61 images, we further investigate the detail of the data.

defect	**2645**	**25**
good	**36**	**2715**
	defect	good

Figure 4: Confusion matrix of the model with resampling

There are 36 false negatives in the validation dataset. We examine these images and find that 30 of the failures are attributed to the possible incorrect labels. These images are very likely to be incorrectly classified into good devices since there are apparent scratches on the chips. There is no consistent pattern of the scratches among these images (see Figure 5). Hence, we conjecture that the errors come from the manual classification process. If we correct the label of the 30 images, the corrected validation accuracy can reach 99.43%, which is a significant improvement of the performance.

Moreover, since the validation dataset is randomly selected from the image dataset, there should be a similar ratio of possible incorrect labeled images in the training dataset. Therefore, overall, there should be about 300 devices that are incorrectly classified into the good class. By correcting the mislabeled images, we can achieve at least 99.43% accuracy within 20 epochs' training.

Similarly, we also investigated the 25 false-positive images. We find that all of them have the correct labels, but the proposed model failed to predict the labels. Surprisingly, most of the images have typical scratches that are evident to human eyes. We think a possible explanation is that we have an uneven distribution of different unqualified devices in the dataset. There are not enough training data that have the same scratches in a similar position. Therefore the model failed to learn the pattern for certain types of images in the defect class. This issue can be resolved by increasing the number of training epochs. Since we only implement 20 epochs in our training, there should be room for improvement of performance.

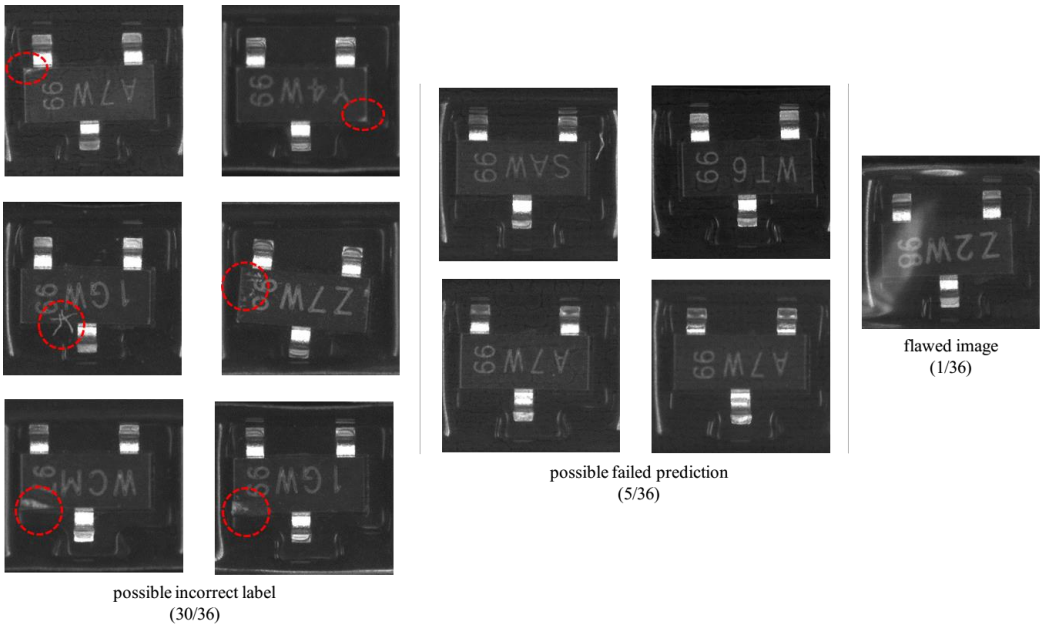

Figure 5: Representative false negative images

6 Discussion

At the beginning of this project, we planned to exploit transfer learning to perform the defect detection task. However, all the models (VGG16, ResNet50, Densenet121) we tried with pre-trained weights perform badly on the dataset. None of them can achieve 80% accuracy. We think the reason is that the difference between images labeled as good and images labeled as defective is very minimal, which is different from the training data that the existing model is trained. Therefore, the pre-trained weights are not able to capture the features of the semi-conductor devices. Due to this reason, we decided to design a model from scratch to cope with this task.

Inspired by VGG16, we construct a simplified yet enhanced VGG-like CNN model. The proposed model is very light and easy to train. The parameters' size is significantly smaller than the original VGG16 and most of the other classical CNN models (see table 2). Also, the proposed model has the smallest depths, which makes it easy to train. The model has never been trained for over 20 epochs in the experiment, but the accuracy is already prominent.

	Proposed model	**VGG16**	**VGG19**	**ResNet 50**	**DenseNet 121**	**DenseNet 169**	**DenseNet 201**
Params	9MB	132MB	137MB	24MB	8MB	14MB	19MB
Depths	11	23	26	50	121	169	201

Table 2: Parameter size and depths of the proposed model and other CNN models

7 Conclusion and Future Improvement

In this project, we designed a model from scratch for semi-conductor device defect detection task. The proposed model is light and easy to train. It has achieved outstanding performance compared with utilizing transfer learning of other existing models. Also, we proved that data augmentation could help increase the accuracy of both classes of the images despite we only resampled one class of them. By investigating the mispredicted images, we find that the performance can be improved by correcting the current dataset's noise and increasing the

number of training epochs.

We will implement other data augmentation techniques and find the most suitable ones for the particular dataset in future work. We also plan to increase the number of layers when the parameters remain in a reasonable range and test how the depth of the model architecture can affect the model's performance.

Declaration

Both authors have the same spirit to complete the project together and contribute equally to this project.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2015, Accessed: Dec. 14, 2020. [Online]. Available: http://www.robots.ox.ac.uk/.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2016, vol. 2016-December, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *Cvpr*, Jun. 2015, Accessed: Oct. 28, 2020. [Online]. Available: http://arxiv.org/abs/1506.02640.

[4] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2019, vol. 2019-June, pp. 4396–4405, doi: 10.1109/CVPR.2019.00453.

[5] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 2261–2269, Aug. 2016, Accessed: Dec. 14, 2020. [Online]. Available: http://arxiv.org/abs/1608.06993.

[6] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J. Big Data*, vol. 6, no. 1, p. 60, Dec. 2019, doi: 10.1186/s40537-019-0197-0.