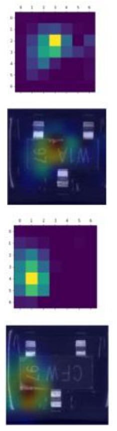
Which area in the semi-conductor is defected?

Nexperia Image Classification II with Deep Learning

1. Introduction

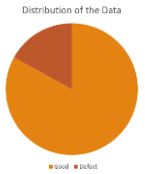
Nexperia provided a dataset for Kaggle in-class contest that aims to classify images of semiconductor devices into two main classes, good and defect. We have created a simple CNN model and apply transfer learning with pre-trained RESNET50, VGG model. We either fine-tuning or just train the top layer in these model to predict the result. We will evaluate the performance of pre-processing technique and model with both the AUC score and the corresponding activation heatmap (Grad-CAM). We put more focus on the heatmap to see if the model can correctly identify the defect area stated in the dataset, then we will analyze the reason behind and propose a method of future improvement.

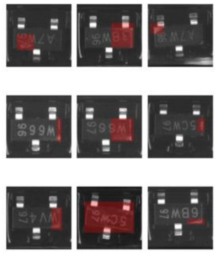


(The graph on the right shows which area are defected by visualize the gradient in the conv layer.)

2. Understand the Dataset

In the dataset, there are 34459 Good sample and 7039 Defect sample. The distribution are highly uneven. Given that the testing data will have similar distribution as the training, we should not oversample the minority to achieve a higher AUC score. To evaluate the performance during training, 20% of the data will be treated as "validation set" and the others will be treated as "training set". We will compare the AUC score in the testing data of doing oversampling or not in the training set and use less training data (only 80% of the dataset) or the whole to train the model.





In the defect sample, we can see some scratch on the semi-conductor, and those may be possible defect area, and in most of the sample, there are alphabet in the middle which have similar intensity with the defect scratch that may confuse the model to distinguish good or bad.

(The red highlight parts is the defect area given in the dataset)

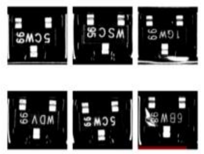
Since in the real-world data, the label maybe wrong, and we don't know what is the real one. We may consider using unsupervised clustering to classify the image, then compare the result with the label. However, we will focus on using CNN in this classification problem.

3. Method

To make it easy for the model to do the classification, we have applied several image processing to evaluate their performance. In method 1, 20-50 pixels near the boundary will be cropped, and the intensity lower than the threshold will be set to 0 (Black). The purpose of doing this is to reduce the noise near the boundary of the image.

In Method 2, we will apply a filter that when the intensity higher than a specific threshold, those pixel will be set to 1, otherwise, set it to 0.

Pre-processing 1


Pre-processing 2

It can highlight the "scratched" part in the image, however, it may also highlight the "noise" that negatively affect the prediction.

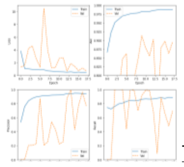
4. Experiment

We first build a simple CNN model to train the data first, then use transfer learning to see if there is an improvement compared with our own mini-CNN model. To prevent overfitting, we have applied all the technique that may help to prevent such as dropout, L2 regularization, early stopping, save the best model during the training.

After experiment, Early Stopping can always achieve higher AUC than keep training the model till the end in all the experiment. (i.e. 50 epoch). Dropout and L2 regularization do not significantly boost the AUC score, but the training curve will become much stable. Fitting the dataset into either one of the model can achieve very high AUC already, however, we want to analysis something more than that. Due to the restriction of Kaggle that can submit at most 5 prediction (Although I have already submitted ~35 prediction csv file), we may not always evaluate the performance of different method with the test dataset.

5. Main Result

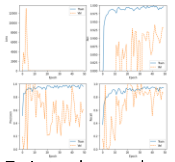
Simple CNN



	Test AUC
without preprocessing:	0.95439
with Pre-processing I:	0.97593
with Pre-processing II:	0.75462

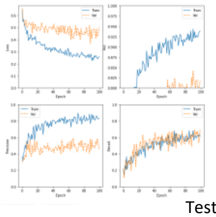
(Always predict negative)

RESNET50



	Test AUC
Train top layer only:	0.88462
Fine Tune whole Model (80% training set only):	0.986
Fine Tune whole Model (100% training set):	0.99157

VGG

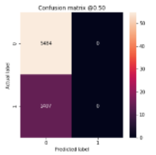


	Test AUC
Train top layer only:	0.89869

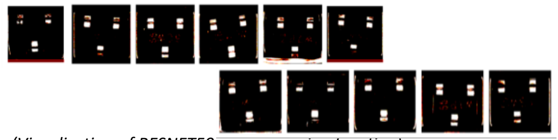
Notes: In most of the case, we just train the model with 80% of the training data for the test prediction only. The actual AUC after using the whole training data will be much higher after we have found the best method. Using the whole training set and usually boost 0.01-0.02 AUC.

5. Analysis

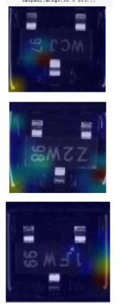
When the model always predict "Negative", the AUC will be 0.7, and the accuracy will be 0.75. A good model should be far higher than the above number, especially in the "Precision". Pre-processing II (Turning the image into black and white) have a negative impact in classification and the model always predict "Negative" in the method. Transfer learning can always give a better AUC compared to a simple model. The pre-trained weight and the very deep network in RESNET50 can help to identify important feature.



The alphabet in the middle of the semi-conductor does not affect the prediction, which is different as what I expected before doing it. We can see that there is no color on the alphabet in the heatmap. When the image pass-through the pre-processing function of RESNET50, the intensity of the character will become very small, or almost disappear.

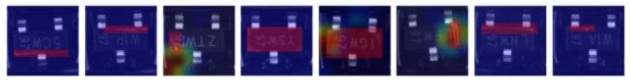


(Visualization of RESNET50 pre-processing function)

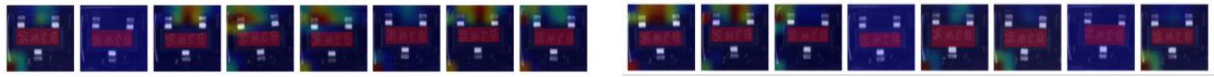


However, the boundary are activated and misclassify as defect in the heatmap on the right. The scratch on the boundary has mislead the model when we do not crop the boundary in the image. We can see that RESNET50 pre-processing function will highlight the boundary part, and we think that cropping the boundary should increase the AUC score significally. In the simple CNN model, the cropping help to get a better AUC (from 0.95 to 0.97), but in the pre-trained model, cropping will drop the AUC from 0.99 to 0.983. Although the model predict the sample as "defect" due to the boundary and the actual label are also "defect", cropping the boundary will lead to a lower confidence of predicting the sample as "defect".

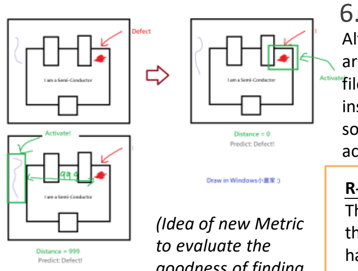
Moreover, we have observed several defect type: straight line, conner, whole conductor, metal or empty. The model can easily identify the defect in the conner of the conductor, however, it is very hard to defect a straight-line type. There is few / no activation on the straight-line scratch in the graph below.



Although the model successfully classify the image as "defect", but if we look at the heatmap, the activation is quite different with the defect area given in the dataset. During the training, we do supervised learning with the correct "label" only, but we didn't' tell the machine that which area are defected. The model have to find out the area by itself through back-propagation, and there are many "noise" in the semi-conductor in each image, causing this kind of activation. The importance of data cleaning which try to make the image as clean as possible like removing the unnecessary boundary scratch should help the prediction process. In this dataset, most of the noise are in the boundary, and we should crop it out for better training. Although the AUC decrease, the activation heatmap will be closer to the given defect area.



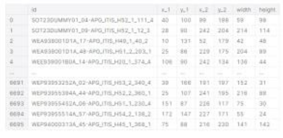
If we want to evaluate how good the model in predicting the defect area, we may define a new metric as the distance between the defect location given in the dataset and the predicted defect location. Here, the predicted defect location will be the most activated location in the heatmap.



(Idea of new Metric to evaluate the goodness of finding the defect area)

6. Possible Future Work for Improvement

Although our current method can give us a brief idea of where is the defect area in the semi-conductor, we want to make good use of the defect area file and want to better supervise the model to identify the defect area, instead of finding it by itself so as to increase the power of prediction. After some research, it seems that the idea of Region Proposal CNN may help us achieve it. Below is a small abstract of this model.



	id	x_1	y_1	x_2	y_2	width	height
0	SOT23DUMMY01_04-APG_ITIS_H52_1_111_4	40	100	99	198	59	98
1	SOT23DUMMY01_09-APG_ITIS_H52_1_12_3	28	90	242	204	214	114
2	WEA93800101A_17-APG_ITIS_H49_1_40_2	10	131	52	179	42	48
3	WEA93800101A_48-APG_ITIS_H51_2_203_1	25	86	229	175	204	89
4	WEE93800180A_14-APG_ITIS_H20_1_374_4	106	90	242	134	136	44
...	...	...	...	...	...	...	...
6691	WEP93953252A_02-APG_ITIS_H53_2_340_4	39	166	191	197	152	31
6692	WEP93955394A_44-APG_ITIS_H52_2_380_1	25	107	241	195	216	88
6693	WEP93955452A_06-APG_ITIS_H51_1_230_4	151	87	226	117	75	30
6694	WEP93955514A_57-APG_ITIS_H54_2_138_2	172	147	227	171	55	24
6695	WEP94000313A_45-APG_ITIS_H45_1_368_1	75	88	216	230	141	142

R-CNN

The region proposal is to find out the possible locations of the target in the figure in advance, which can ensure that the higher recall rate is maintained when fewer windows are selected. And the obtained candidate window has higher quality than the typical Sliding Window Algorithm .

Source: https://medium.com/@nabil.madali/demystifying-region-proposal-network-rpn-faa5a8fb8fce

There are quite a number of paper in the US use this method with transfer learning to find out the manufacturing defect . We attempt to implement this model in MATLAB and Pytorch but we still need more time. Let's try my best later in the coming holiday :)