The First Progress Report
on
Advanced Data Analytics for Abnormal Detection of Semiconductor Devices

1 Introduction

Semiconductors play a key role in many of today's electronic devices, bringing convenience to the society. However, a flawed one can disable the device it is applied to, and, even worse, might bring in safety concerns. Therefore, it is crucial for semiconductor manufacturers to pick out the defect devices (See Figure 1). Nexperia is one of the largest semiconductor manufacturers in the world, producing millions of devices each week, too many for human examiners to handle, so we are looking for a model to automate the abnormal detection.

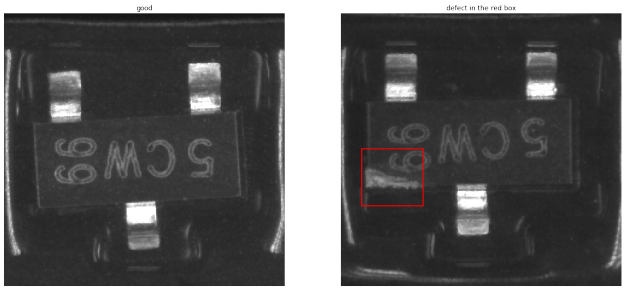

Figure 1. Left: a good semiconductor device; right: a defect device with defect area bounded by a red box

To achieve the goal, Nexperia prepared the following dataset, `https://www.kaggle.com/c/semi-conductor-image-classification-second-stage`, that aims to classify images of semiconductor devices into two main classes, good and defect. Please refer to Section 3.1 for more details about the data. There are nine types of defects, so the task is essentially an image classification task among ten classes (including the good class). The data we obtain are quite imbalanced, with 30523 passed images and 7799 defect images. Moreover, among the nine defect classes, data are not evenly distributed. Some defect classes have as few as 5 images in total, while the largest defect class contains 3681 images. In the assemble line of semiconductors, the proportion between passed and defect images are supposed to be even more skewed. The key is to detect as many defect images as possible while not sacrificing too many passed ones.

Two deep neural network models, ResNet18 and VGG19 with back normalization are used as the plain models, and they are modified to combine spatial transformer networks (STN) or attention networks. The research compares the performance of five models in total. In order to measure the performance, defect images are set as positive (1), and passed ones negative (0). For each model, AUC and the false positive rates (FPR) against true positive rates (TPR) being 99.1%, 99.3%, 99.5%, 99.7%, and 100%, respectively, are estimated. The result shows that a plain ResNet18 model can detect defects fairly accurately with

relatively short training time, with FPR below 5% when TPR is boosted to 99.7%. No other models can vie with a plain ResNet18 when TPR is up to 99.7%. However, when TPR is set to be 100%, i.e., all defect images are picked out, a VGG19 model with STN works best, with FPR below 25% when TPR $= 100\%$ and below 8% when TPR $\leq 99.7\%$, despite the long training time and large memory the model consumes.

Examining the wrongly classified images, the research finds that many of false positive ones (good devices with high defect scores) are indeed wrongly labelled by human examiners and should fall into the defect classes, while the false negative ones (defect devices with high good scores) reflect the inability of the model to detect some anomalies. For the former finding, the higher the defect score is, the more reassuring that an image has defects, while those with relatively lower defect scores are more likely to be considered on the borderline between good and defect (or having mild defect) or even without defect. The latter finding might be caused by insufficient defect images, over-fitting of the models, or misleading information (wrongly labelled data) fed for model training, etc.

There are three implications for the findings:

1. The deep learning models (e.g. ResNet18) used in the research are doing well in abnormal detection of the semiconductors. For products that can bear a small defect rate, the models can be applied to automate the quality control work.
2. The models are able to correct human examiners' mistakes by singling out the defect images mixed in the good ones even if they are trained with wrongly labelled data. Currently the human error rate is estimated at above 1%.
3. There are defects that the models fail to pay attention to. In order to meet 0% error requirement for some industries, e.g. automobiles, a large percentage (around half the products) have to be discarded.

The report is elaborated as follows: in Section 2, methodology is introduced with background knowledge; experiment settings are presented in Section 3; Results and analysis are displayed in Section 4; Section 5 gives direction to future studies; Section 6 concludes the report. During the first year period (Sep 2 2019 - August 31 2020), we meet once or twice every month, with each meeting lasting for about 2 hours.

2 Method

2.1 Deep Neural Networks

Deep neural networks have been well studied in the past decade. Different structures such as ResNet[1], VGG[5] have been proven effective in training large datasets, e.g., ImageNet, CIFAR. However, their effectiveness on more diverse data in the real world is far less explored. We borrow the pre-trained ResNet18 and VGG19 with back normalization from PyTorch to train the semiconductor images, and find that the models can well pick out the abnormal devices with accurate attention paid to the defect areas shown on the heatmaps of the images.

2.2 Spatial Transformer Networks (STN)

The same objects in different images may appear in different orientation, size, or position, adding up the difficulty for neural networks to recognize them. A spatial transofrmer network (STN) is a network that is trained to transform each image through rotation, scaling, and translation so that all the same objects appear in the same shape and position on each image[2] (See Figure 2).

Transformation
For the source coordinate (x_i^s, y_i^s) of each input pixel i, it can be transformed to the target coordinate (x_i^t, y_i^t) by the following linear operation:

$$(x_i^t, y_i^T) = A_\theta \begin{pmatrix} x_i^s \\ y_i^s \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^s \\ y_i^s \\ 1 \end{pmatrix}.$$

STN is thus used to train A_θ with the train image set or their corresponding feature maps obtained from any layer followed by STN.

Localization
STN contains 2 convolutional layers, each followed by a maxpool layer and a ReLu layer, and 2 fully connected layers, outputting the number of trainable parameters required in A_θ

A natural initialization is the identity transformation, i.e., $A_0 = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{smallmatrix}\right)$.

Furthermore, the transformation can be constrained by training only some of the entries in A_θ. For example, if only isotropic scaling and translation are desired, one can modify $A_\theta = \left(\begin{smallmatrix} \theta_1 & 0 & \theta_2 \\ 0 & \theta_1 & \theta_3 \end{smallmatrix}\right)$. In this case, only 3 parameters are trained, so the STN outputs a vector of dimension 3.

STN is convenient and easy to implement because it can be inserted into any CNN between any layers or just right before or after the network with only a small increase of time and memory consumption. When an STN is inserted in the middle or after a CNN, it transforms the feature extraction fed into it. Moreover, STN can be viewed as another approach to data augmentation without introducing new data.

2.3 Attention Networks

An attention network is a modification of existing neural network. It preserves features in the middle of the network by adding one or more attention layers, combining the outputs of the attention layers, and feeding them into the last fully connected layer to give the final score of each class[3].

Figure 3 illustrates the structure of an attention network. Each input image data first goes through the CNN to the second last layer without visiting the attention layers to obtain a

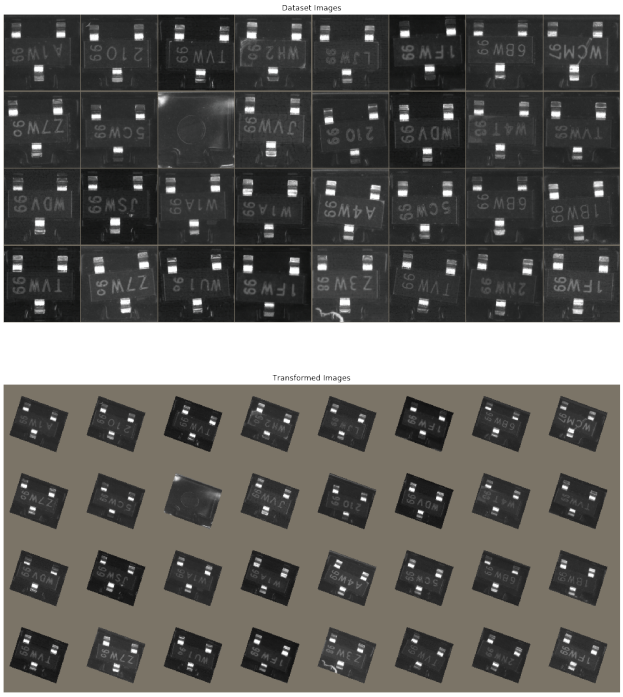

Figure 2. This shows how STN inserted to a VGG19 network with back normalization transform the images. Top: original images; bottom: STN added.

vector g.

For each attention layer s, $\mathcal{L}^s = (l_1^s, \ldots, l_n^s)$ denotes the input feature map, where n is the number of the pixels of the map for each channel, l_i^s a vector of length equal to the number of the channels corresponding to pixel i.

A compatibility scores $\mathcal{C}(\mathcal{L}^s, g) = (c_1^s, \ldots, c_n^s)$ is computed, where

$$c_i^s =< u, l_i^s + g >, i \in \{1, \ldots, n\}.$$

Note that the $+$ operator here denotes vector concatenation. u is a parameter to be trained. The operation can be realized on PyTorch by feeding $l_i^s + g$ to a linear layer without bias term to with a 1×1-dimensional output.

Then a softmax function is be applied to the compatibility score to obtain an attention map at layer s, $\mathcal{A}^s = (a_1^s, \ldots, a_n^s)$, where

$$a_i^s = \frac{\exp(c_i^s)}{\sum_{j=1}^{n} \exp(c_j^s)}, i = 1, \ldots, n.$$

Finally, the attention layer outputs

$$g_a^s = \sum_{i=1}^{n} a_i^s \cdot l_i^s.$$

For a network with S layers, all the attention layer outputs are concatenated as $g_a = [g_a^1, \ldots, g_a^S]$ and fed into the last fully connected layer to obtain the final output of the whole network.

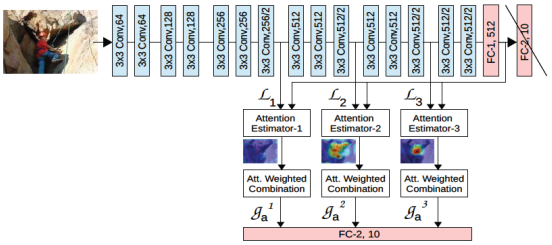

Figure 3. This is an example of 3 attention layers inserted after the last 3 convolutional blocks of a VGG network, respectively. 2 more convolutional layers are inserted between the last block and the first fully connected layer.[3]

One major advantage of an attention network is that it preserves features better than the unmodified network. However, on the other hand, the preservation is at the cost of longer training time and mass memory consumption.

3 Experiment Settings

3.1 Data Preprocessing

Nexperia provides 38322 images in total, with 30523 good ones and 7799 with defects. The defect images are further divided into 9 types: chipping, device flip, empty pocket, foreign material, lead defect, lead glue, marking defect, pocket damage, and scratch, among which 15 out of 135 device flip images also suffer lead defect. Thus, the data are divided into 10 classes in total.

We split the original data into training set of size 31025 (good 24724, bad 6301), validation set of size 3447 (good 2747, bad 700), and test set of size 3830 (good 3052 bad 778). The splitting process is as follows:

1. 10% of the images in each class are randomly selected as the test dataset. Since the image classes are very imbalanced, with some class with less than 10 images in total (e.g., only 5 images are in the pocket damage class. For such classes, only one image is randomly picked out as the test data.
2. For the remaining images, 10% of them are again randomly chosen as the validation set. For the classes with less than 10 images, as in test set, only one is chosen randomly as the validation set.
3. All the remaining images are in the train dataset.

After the split, the data remain in their current set for all model training to ensure consistency. Then the train dataset is used to train models, the validation set to validate the

performance of the models in each training epoch so that the best model for validation set is saved for testing the test dataset.

As to the data in `https://www.kaggle.com/c/semi-conductor-image-classification-second-stage`, we merge the train and validation dataset as well as all defect type into one class for simplification. In this way, each of the images with multiple defects (in this case with 15 images suffering both lead defect and device flip) occurs only once in the defect folders, so 13 of them in the train dataset, while 2 in the test dataset. Moreover we move the 49 defect escapees in the test dataset and 51 defect escapees to the defect folder where they should belong to (See Section 4.2 for details about noisy labels). Therefore, the datasets in the Kaggle contest contain 34457 train images (27420 good and 7037 bad) and 3830 test images (3003 good and 827 bad).

Before feeding the data into any model, we convert each image to RGB (3-channel) format, resize and centre-crop it to fit them to the size 224×224. To train all models except for the STN ones, we randomly rotate each image by within 10 degrees, change the brightness, contrast and saturation and do a slight affine transformation for the train dataset so as to augment the data. For all datasets in all model training, validating, or testing, they are normalized with mean$= [0.485, 0.456, 0.406]$ and standard deviation$= [0.229, 0.224, 0.225]$ for each channel, respectively.

3.2 Baseline Models

Resnet18 and VGG19 with back normalization (we will call it VGG19_bn for simplicity from now on) and their modifications are experimented. The baseline models Resnet18 and VGG19_bn are downloaded from Pytorch with pretrained parameters. The last linear layer of each is adjusted to 10 outputs to fit the 10-class classification task.

Cross entropy loss is used as the loss function, i.e., for data with class c (from 1 to 10) and output vector x, the loss is

$$loss(x, c) = -\log\Big(\frac{\exp x[c]}{\sum_{i=0}^{9} \exp x[i]}\Big).$$

We apply SGD as the optimization algorithm, with learning rate$= 0.001$, decaying every 14 epochs at the rate of $\gamma = 0.1$ momentum$= 0.9$. The models are both trained fro 50 epochs.

3.3 STN Modification

Then an STN is added before both Resnet18 and VGG19_$n$bn. The new models are denoted as Resnet18_stn and VGG19_bn_stn, respectively. The loss function and optimization scheme are the same as in training the baseline models except that the momentum is adjusted to 0.6 for Resnet18_stn and 0.5 for VGG19_bn_stn. The models with STN are trained for 100 epochs each.

For the four above-mentioned models (2 baseline models and 2 STN models), the model weights giving top ten validation AUCs are saved for testing for each model.

3.4 Attention Modification

Due to limited memory on one GPU, only a Resnet18 model is modified with attention layers (See Table 1 for details), denoted as Resnet_att_2. The optimization scheme and learning rate scheduling are the same as training the baseline models.

layer name	Resnet18
conv1	
conv2	$\begin{bmatrix} 3\times 3, 64 \\ 3\times 3, 64 \end{bmatrix} \times 2$
conv3	$\begin{bmatrix} 3\times 3, 128 \\ 3\times 3, 128 \end{bmatrix} \times 2$
conv4	$\begin{bmatrix} 3\times 3, 256 \\ 3\times 3, 256 \end{bmatrix} \times 2$
conv5	$\begin{bmatrix} 3\times 3, 512 \\ 3\times 3, 512 \end{bmatrix} \times 2$
~~avgpool~~ conv6	$3\times 3, 512$ maxpooling
output	

Table 1. To modify a Resnet18 for an attention network, we add an attention network after each of the last two convolutional layers (conv4 and conv5). Furthermore, we replace the average pooling layer (avgpool) between the last convolutional layer (conv6) and the last fully connected layer with a convolutional layer and a maxpooling layer (conv6).

3.5 Performance measurement

ROC (receiver operating characteristics) is employed to measure the performance of each model. An ROC curve plots TPR (true positive rate) against FPR (false positive rate) against, where

$$\text{FPR} = \frac{\text{true negatives predicted as positives}}{\text{all true negatives}},$$

while

$$\text{TPR} = \frac{\text{true positives predicted as positives}}{\text{all true positives}}.$$

In this project, defect images are set as positive, the good ones negative.

The ROC curve reflects the percentage of good devices one has to abandon (FPR) to achieve the desired TPR (The percentage of defect devices picked out, which is desired to be as high as possible). Based on the ROC curve, we compare the AUC (area under ROC curve), and FPRs when TPR$= 99.91\%, 99.93\%, 99.95\%, 99.97\%$, and 100.00%, respectively to select the best models for different real world situations.

4 Results

4.1 Performance

We train 5 deep neural network models, Resnet18, VGG19_bn, Resnet18_stn, VGG19_bn_stn, and Resnet_att_2 and compare the results in Table 2.

mean (std) %			Resnet18	VGG19_bn	Resnet18_stn	VGG19_bn_stn	Resnet_att_2
AUC			99.83 (0.00)	99.70	99.69 (0.01)	99.82	99.85 (0.04)
TPR$\geq$ %	99.1	FPR	1.25(0.08)	2.80	3.05(0.31)	1.69	1.07 (0.26)
	99.3		2.21 (0.41)	2.90	10.74(0.80)	4.46	1.45 (0.35)
	99.5		3.60 (0.31)	6.19	23.73 (4.56)	9.12	4.31 (2.49)
	99.7		4.71 (0.40)	8.83	34.07 (5.52)	15.73	10.84 (5.65)
	100		36.80 (2.06)	86.71	48.44 (1.48)	37.68	44.04 (23.45)

Table 2. For the models based on Resnet18 (Resnet18, Resnet18_stn, and Resnet_att_2), top ten models weights giving the highest AUC on the validation dataset are saved for the test data. Therefore, for each of these three models, the results include averages and standard deviations (in the brackets) of the performances of the top ten model weights.

From the model, we can see that Resnet18 outperforms the other models in all metrics. The difference between FPRs of Resnet18 and VGG19_bn_stn when TPR is set to 100% is small, meaning that for devices with zero tolerance of defects (e.g., for the car industry), both Resnet18 and VGG19_bn_stn can help detect anomalies. However, the training time and memory consumption of Resnet18 are far less than the latter. On the whole, Resnet18 is the best choice among the five models tested in the report.

Reproducible codes can be found at the following website:

`https://github.com/huangkaiyikatherine/nexperia`

4.2 Noisy Labels

One major finding on this stage is that deep neural networks can detect noisy labels accurately even if the models are trained with noisy data.

We select 55 and 51 good images in the test and train dataset, respectively, bearing the highest defect scores given by Resnet18 and reexamine them. It turns out that, for the test data, 46 of them have defects, while 3 on the borderline between good and defect, with only 3 images being actual good ones, and all except 6 of the 51 train data are confirmed defect escapees, while the remaining 6 are on the borderline, ie, can also be classified as defect. Considering the fact that the total number of good examples in test dataset is 3052, we estimate that the human error rate (percentage of noisy labels among all good devices) is over 1%. Moreover, the 3 images on the borderline rank low in defect score

among the images reexamined, indicating that the deep learning model is able to assign high defect scores to images suffering more obvious defects.

After correcting the known wrong labels (including those on the border line), the test result of Resnet18 is updated on Table 3. The FPRs are improved by around 1% for the 5 TPRs. Due to the existence of potential unknown noisy labels, the results are less trustworthy, but the network can help spot the noises, and together with human-machine interaction, we can gradually correct all the labels to obtain real performance of the models.

4.3 Model Failures

Another focus on the results is the defect images the models fail to detect. Some images have minor defects that even human eyes find hard to distinguish (see Figure 4 on the left), while the others with obvious defects escape the models' attention for unknown reasons (see Figure 4 on the right).

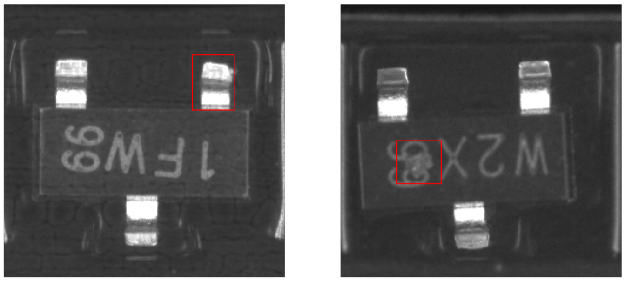

Figure 4. Left: lead defect in the red box; right: scratch in the red box

There are several hypotheses about and potential solutions to the failures.

1. The defects are too minor to reach a threshold for models to detect. In this case, models to attack aleatoric uncertainties[4] might help.
2. The failure is attributed to lack of information (epistemic uncertainty), so more data with similar defects might explain it away[4].
3. There are a significant amount of noisy labels that affect judgement of the models. Human-machine interaction to correct the labels might help.

5 future studies

Future studies will aim to attack the model failures mentioned in Section 4.3. Moreover, we will study domain shift problems in deep learning so that a model trained with only one batch of data can be applied to data in different environments, which are in this case devices manufactured by different machines or even different types of semiconductor devices.

mean (std) %			Resnet18	Resnet18 (corrected)
AUC			99.83 (0.00)	99.91 (0.00)
TPR$\geq$ %	99.1	FPR	1.25(0.08)	0.55(0.14)
	99.3		2.21 (0.41)	0.98 (0.25)
	99.5		3.60 (0.31)	1.28 (0.31)
	99.7		4.71 (0.40)	3.25 (0.40)
	100		36.80 (2.06)	35.87 (2.05)

Table 3. Comparison between the test result of Resnet18 and that after correcting 46 wrongly labelled defect images with the highest defect scores.

6 Conclusion

On this stage, we experiment 5 deep learning models derived from Resnet18 and VGG19_bn and compare their performances, i.e., FPRs when TPRs are closer or equal to 100%. It turns out that all the models can detect most of the defects with Resnet18 topping the rank. Furthermore, the models are able to spot noisy labels due to human errors. A human-machine interaction is suggested to correct the labels for better training. However, there are defects (see Figure 4) all the models fail to detect. The reasons for the failures remain unexplored. On the next stage, we aim for attacking these failures and dive into domain shift problem to adapt the models to more diverse datasets.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] M. Jaderberg, K. Simonyan, and A. Zisserman, “Spatial transformer networks,” in *Advances in neural information processing systems*, 2015, pp. 2017–2025.

[3] S. Jetley, N. A. Lord, N. Lee, and P. H. Torr, “Learn to pay attention,” *arXiv preprint arXiv:1804.02391*, 2018.

[4] A. Kendall and Y. Gal, “What uncertainties do we need in bayesian deep learning for computer vision?” In *Advances in neural information processing systems*, 2017, pp. 5574–5584.

[5] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.