Deep Learning Applications in Image Classification with Imbalanced Classes, Noisy Labels, and Distributional Shift Issues

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.

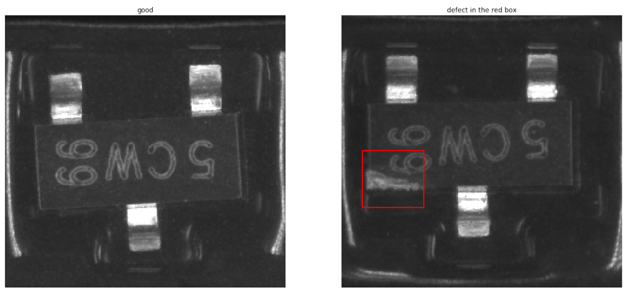


Figure 1. Left: a good semiconductor device; right: a defect device with defect area bounded by a red box

To achieve the goal, Nexperia has prepared three batches of datasets thus far, the first two of which can be found on the two Kaggle contest sites `https://www.kaggle.com/c/semi-conductor-image-classification-first` and `https://www.kaggle.com/c/semi-conductor-image-classification-second-stage`, respectively. The competitions aim 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 four major difficulties in the task:

1. The data is imbalanced. Digging into the data, we find nine major types of defects, though more sub-types can be further defined. Unlike the well studied public data such as ImageNet[1] or CIFAR10[6], the data we obtain are quite imbalanced, with mostly passed images and a few defect ones. Moreover, among the nine defect classes, data are not evenly distributed. Some defect classes have as few as single digit number of images in total, while the largest defect class contains thousands of 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.

2. A classification task as it is, the definition of some defect classes is more a regression (thresholding) problem with different assessment standard according to different human operators. Therefore, there is ambiguity in labelling for images lying on

the borderline between good and defect classes, adding more difficulty to the task. Chipping, foreign material, lead defect, and scratch are such typical classes.

3. Similar to ImageNet[1], Nexperia data fall prey to noisy labels, i.e., some defect images were mislabelled as good by human operators.
4. Different batches of data are collected from different machines that produce the devices at different time, thus subject to distributional shift. The distributional shift issue can be formulated as a causal invariant problem or an underspecification problem and remains unexplored.

To attack the problems, two deep neural network models, ResNet18 and VGG19 with back normalization both pretrained on ImageNet[1] together with their modifications to combine spatial transformer networks (STN) or attention networks are trained with the conventional cross entropy loss function. 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 ≤ 99.7%, despite the long training time and large memory the model consumes. However, FPR is very unstable when TPR is 100%. After using TenCrop technique on the test dataset, i.e., cropping each test image on the four corners and at the centre, and flipping it horizontally to do the same croppings again, feeding the data to the model and averaging the output of the 10 images cropped from each same one to evaluate the performance, FPR drastically drops when TPR is 100% for ResNet pretrained on ImageNet, making the model the best in terms of stability, performance, and cost-effectiveness.

Examining the wrongly classified images, the research finds that many of false positive ones (good devices with high defect scores) are indeed mislabelled 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. To be more specific, 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. Here good and defect scores are two numbers between 0 and 1 and summing up to 1. Interestingly, comparing the training behaviour of clean and mislabelled data, an early learning phenomenon on clean data is observed, in line with many experiments on noisy labels in public data[2][7][8]. Therefore, a reweighting and label modification scheme such as self-adaptive-training (SAT)[2] can be applied to enhance the learnability of images that converge in early epochs while weakening the influence of those that are hard to learn and more suspicious of being mislabelled. Nevertheless, probably due to imbalance of the data, training behaviour varies among classes. Thus, modifications of SAT algorithm are required to adapt to Nexperia data.

The research so far has the following contributions:

1. The pretrained deep learning models (e.g. ResNet) used in the research are doing well in accurate and efficient abnormal detection of semiconductor images. Furthermore, a simple TenCrop technique applied on test data preprocessing can suppress FPR by over one half to 10% on a pretrained ResNet50 model. For products that can bear a small underkill (failure to detect bad devices) or overkill (discard passed devices as defect) rate, quality control can be automated.
2. Noisy labels are found in underperforming data in both train and test sets. Currently the human error rate is estimated as above 1%. Fortunately, deep convolutional neural networks are robust against noisy labels in early learning as stated in many available literatures[8][7][2] in terms of public datasets[6], though learning patterns across classes vary. Thus, carefully designated re-weighting scheme and labels softened by model predictions can be applied on cross entropy loss function to avoid overfitting.
3. Models trained on one batch of data fail to achieve viable results on another batch collected from a different time period. A distributional shift issue arises. One hypothesis is that training on different batches of data could amplify invariant features and that online few shot learning could capture new features in input data.

The report is organized 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.

2 Methodology

2.1 Deep Neural Networks

Deep neural networks have been well studied in the past decade. Different structures such as ResNet**ResNet**, VGG[9] have been proven effective in training large datasets, e.g., ImageNet[1], CIFAR[6]. 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 fairly 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 as variations in different orientation, size, or position. A spatial transofrmer network (STN) is a network that is trained to transform each image through rotation, scaling, and translation so as to align the same objects in the same shape and position to be better recognized [3] (See Figure 2). In this sense, STN can be viewed as an alternative approach to data augmentation.

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
an STN contains 2 convolutional layers, each followed by a maxpool layer and a ReLu layer, and 2 fully connected layers, and outputs a linearized vector of the trainable parameters required in A_θ. A natural initialization is the identity transformation, i.e., $A_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$.

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 $A_\theta = \begin{pmatrix} \theta_1 & 0 & \theta_2 \\ 0 & \theta_1 & \theta_3 \end{pmatrix}$. 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 at any layer 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 the following layer.

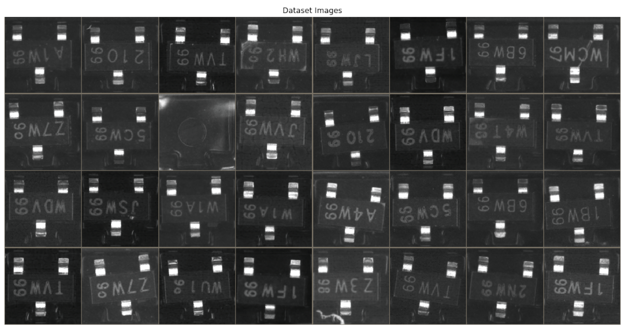
2.3 Attention Networks

An attention network is another modification of an existing CNN. It preserves features in the middle of the network by adding one or more attention layers in the middle, combining the outputs of the attention layers as a vector, and feeding the combined vector into the last fully connected layer to give the final score of each class[4].

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 vector g. For each layer previous to the attention layer s, $\mathcal{L}^s = (l_1^s, \ldots, l_n^s)$ denotes the corresponding output, 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

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



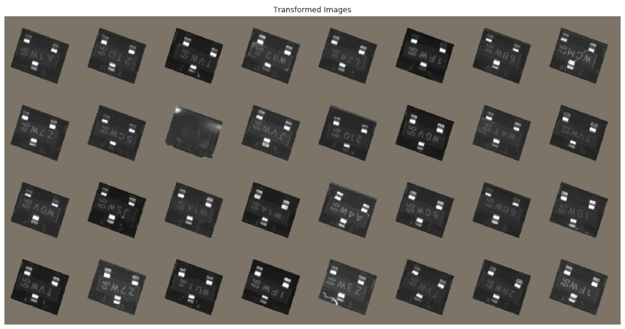


Figure 2. This shows how STN inserted to a VGG19 network with back normalization transform the images. Top: original images; bottom: STN added.

Note that the $+$ operator here denotes vector concatenation. u is a parameter to be trained. The operation is indeed feeding $l_i^s + g$ to a linear layer without bias term 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.

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.

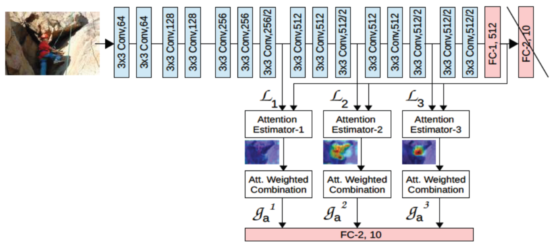


Figure 3. This is an example of 3 attention layers inserted after the last 3 convolutional blocks of a VGG network, respectively. 2 more convolutional layers are inserted between the last block and the first fully connected layer.[4]

2.4 Crop Styles

Since the images are usually of different sizes or not of the desirable size for model input, cropping is necessary for data preprocessing in computer vision. There are several techniques to crop images before feeding them into a model. The simplest and most common way to crop an image is to use a CenterCrop, which is to crop the image at the centre for a 224×224 size, size of ImageNet data. However, one can also use a FiveCrop technique to crop on the four corners and the centre to obtain five different crops to feed them all into a model and then obtain the output by averaging the five results. A more complicated cropping scheme is to do a FiveCrop on an image, flip it either horizontally or vertically, and then do another FiveCrop, and finally obtain the output of the image by averaging the outputs after feeding each of the ten crops into a model. Stabilization of model results is expected after smoothing out outliers of any crops by averaging.

2.5 Early Learning

An early learning phenomenon has been recorded by many previous works[2][7][8]. While training a deep neural network with noisy data, it has been observed that in early learning, the networks are usually robust against label noise, but that as time passes, the models tend to interpolate the train dataset but overfit on validation dataset. This can be seen on accuracy, loss, and margin error (the difference between the output on the given class and the largest output excluding the given class of an image) curves on training and validation data, where train accuracy tends to 100%, loss to 0, and margin error to a large value, while validation accuracy and margin error peaks at some point and then decrease, and that validation loss bottoms before rising. Therefore, it is essential to capture the turning points of the validation curves to avoid overfitting.

2.6 Self-Adaptive-Training (SAT)

Self-adaptive training developed by [2] is a method to alleviate overfitting after the turning points.

In multi-class deep learning tasks, a cross-entropy (CE) loss function

$$\mathcal{L}(X,Y,f) = -\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij} \log \sigma(f(x_i)_j),$$

is commonly used, where $X = (x_1, \dots, x_n)$ denotes n input sample data, $Y_i = (y_{i1}, \dots, y_{iC})_{i=1,\dots,n}$ the one-hot label for C classes of the corresponding sample i, f the output function, and σ an activation function, usually $Softmax$ or $Sigmoid$. Unfortunately, when input labels are corrupted, deep learning models with CE are doomed to overfitting despite observed robustness against noisy labels in early training.

Many available literatures[2][8][7] confirm the early learning phenomenon and add regularization to address the noisy labels issue, among which [2] neatly adds a weight obtained from early training results to the CE loss function, naming it a self-adaptive-training (SAT) alogorithm, incurring no calculation cost and achieving performance close to that without label noise. The SAT loss function is formulated as follows:

$$\mathcal{L}(X,Y,f) = -\sum_{i=1}^{n} w_i(\sum_{j=1}^{C} t_{ij} \log f(x_i)_j),$$

where $t_i(\tau) = \alpha t_i(\tau-1) + (1-\alpha)f(x_i)(\tau)$ is the soft label at the τ-th iteration for some momentum $\alpha \in [0,1]$ with the initialization same as the assigned label y_i, $w_i = \max_j t_{ij}$ the sample weight.

By observation, in early training, w_i is relatively small for noisy labels, thus suppressing the influence of the corrupted labels on model training. In the extreme case, the corrupted labels could eventually be corrected by soft labels t_i, thus boosting w_i equal to the score of the corrected class. Thus, SAT has the ability to both auto-correct noisy labels as well as checking the sanity of labels by ranking sample weights.

2.7 Weighted Binary Cross Entropy Loss

Another comparison is observed between measures over 10 classes and that over the binary class (good and defect). The latter is obtained by summing up the 9 defect scores to 1 defect score as opposed to the good score. Comparing the validation curves, we have found that the extreme points of accuracy/loss validation curves differ from that of the binary accuracy/loss curves. For example, when the accuracy validation curve plateaus, the binary accuracy curve might indeed in a downturn trend, for defect images tend to be predicted as good ones rather than a wrong defect class. To counter this downturn, a binary cross entropy loss function should be included to ensure that even if a defect image is misclassified, it will only be predicted as another defect class instead of the good one. Therefore, the loss function can be written as

$$\mathcal{L}_{\text{weighted binary CE}}(x_i, y_i, f) = \lambda L_{i,CE} + (1-\lambda) L_{i,binary},$$

where $L_{i,CE}$ is the cross-entropy loss function over 10 classes of sample i, while

$$L_{i,binary} = -(1 - y_{good}) \log(\sum_{\text{all defect classes } j} \sigma(f(x_i)_j)) - y_{good} \log \sigma(f(x_i)_{good}),$$

where $y_{good} = 1$ if image i is a good one and 0 if it is defect.

3 Experiment Settings

3.1 Data Preprocessing

In batch 1 dataset, there are $30,000$ images in total, among which $27,000$ are good, and $3,000$ bad. Batch 2 dataset, with $38,302$ images in total, $30,523$ good and $7,779$ bad, sees a more detailed and imbalanced classification, where the defect images are further divided into 9 types listed as below (the number in the parenthesis following each type indicate the number of images): chipping ($3,681$), device flip (135), empty pocket (203), foreign material ($1,952$), lead defect (269), lead glue (625), marking defect (175), pocket damage (5), and scratch (749), among which 15 out of 135 device flip images also suffer lead defect. Thus, the data are divided into 10 classes in total. The batch 3 dataset has a similar classification scheme to batch 2 except that a significant number of foreign material defect images bear tin particles on the surface, thus allocated to a new class, and that very rare but attention-worth images are subjected to camera glitches and cropping issues, thus classified as a new type neither could be passed nor discarded as defect.

The article focuses on experiments on the batch 2 dataset but toys with batch 3 data for the purpose of addressing the distributional shift issue. We split the batch 2 data into a training set of size $31,025$ (good $24,724$, bad $6,301$), a validation set of size $3,447$ (good $2,747$, bad 700), and a test set of size $3,830$ (good $3,052$ 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, the pocket damage class with less than 10 images in total. Therefore, only one image is randomly picked out as the test image.
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 for 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 types 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 $34,457$ train images ($27,420$ good and $7,037$ bad) and $3,830$ test images ($3,003$ 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

ResNets (with 18, 34, and 50 layers, respectively) and VGG19 with back normalization (we will call it VGG19_bn for simplicity from now on) and their modifications are experimented. The baseline models ResNets and VGG19_bn are downloaded from Pytorch with parameters pretrained on ImageNet. 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

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

We apply SGD as the optimization algorithm, with learning rate$=0.001$, decaying every 14 epochs at the rate of $\gamma = 0.1$ and 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_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 Cropping

CenterCrop, FiveCrop, and TenCrop are experimented either only on the test dataset or on all the train, validation, and test datasets

3.6 Loss Reweighting and Label Softening

It has been observed that the validation accuracy and loss curves for good class plateaus at around 90th epoch (slightly increasing for the loss curve thereupon) and finally stabilize at around 150th epoch (see Figures 4 good and 5 good). Therefore, the warm-up epochs where the models are trained using the conventional cross entropy loss end at between these two points.

Then, self-adaptive-training (SAT) is applied, though, in addition to that in [2], two modified SAT schemes are experimented, one is called SAT bad 1, which only applies SAT on good data, for the learn curves of defect classes usually lag behind in terms of plateauing probably due to small numbers and that all the noisy labels spotted by now are in good class, and the other is called SAT bad boost, which applies only on good data as SAT bad 1 and boost the weights of lead defect, pocket damage, marking defect, and scratch while calculating losses so as to alleviate the imbalance issue.

3.7 Weighted Binary Cross Entropy Loss

By checking the learning curve of validation set by class, one can come up with a turning point for each class where the corresponding validation curve reach its extremum. There upon a weighted average of cross entropy loss by 10 classes and binary cross entropy loss between good and defect classes are calculated for each image of the class. (See Figures 4 and 5)

For both SAT and weighted binary cross entropy loss methods, ResNet34 is trained from scratch for 200 epochs with initial learning rate at 0.01 and a cosine annealing scheme on learning rate decaying throughout the training.

3.8 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 and Analysis

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.

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.

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.

Furthermore, FiveCrop and TenCrop are applied on test data to evaluate the performance of ResNets (18, 34, and 50 layers, respectively). It is observed that ResNet50 with TenCrop on test data gives the best result, cutting the FPR at TPR$=100$% to 10% (see Table 3).

# of layers			18			34			50		
Crop Style			CenterCrop	FiveCrop	TenCrop	CenterCrop	FiveCrop	TenCrop	CenterCrop	FiveCrop	TenCrop
AUC% (std 10^{-2})			99.83 (0.00)	99.78 (0.01)	99.70 (0.01)	99.84 (0.00)	99.83 (0.00)	99.80 (0.00)	99.84 (0.00)	99.81 (0.00)	99.86 (0.00)
TPR$\geq$ %	99.1	FPR	1.25(0.08)	2.26 (0.29)	7.21 (0.31)	1.32 (0.09)	1.20 (0.12)	4.96 (0.43)	1.50 (0.10)	1.07 (0.08)	1.91 (0.43)
	99.3		2.21 (0.41)	2.89 (0.48)	8.33 (0.80)	2.15 (0.23)	2.00 (0.18)	5.83 (0.41)	1.73 (0.09)	1.24 (0.06)	3.10 (0.54)
	99.5		3.60 (0.31)	10.26 (2.14)	11.94 (0.04)	4.01 (0.25)	2.93 (0.21)	7.45 (0.55)	2.26 (0.15)	1.50 (0.08)	3.91 (0.42)
	99.7		4.71 (0.40)	12.21 (2.00)	13.84 (0.04)	4.42 (0.42)	15.84 (1.65)	9.15 (0.48)	3.61 (0.36)	3.34 (0.46)	7.79 (0.94)
	100		36.80 (2.06)	62.18 (7.09)	24.62 (0.07)	46.69 (1.92)	38.49 (1.14)	15.59 (1.35)	59.85 (1.86)	52.86 (1.58)	10.65 (0.41)

Table 3. The models are ResNets pretrained on ImageNet, now trained for 50 epochs on our Nexperia train dataset. The crop style are only applied on test dataset, where CenterCrop is to crop each image at the centre for size 224×224, FiveCrop is to crop an image at the four corners and the centre for the same size, and TenCrop is to do a FiveCrop, flip the image horizontally and do another FiveCrop. For FiveCrop and TenCrop, we average the results of each crop for one image to obtain the final output.

Then, training with FiveCrop and TenCrop is inspired. one can find that all FiveCrop, TenCrop with both a horizontal and a vertical flip stabilize the FPR when TPR is 100% in later epochs when finetuning a ResNet34 network, among which a FiveCrop achieves the best result (See Figure 6).

The results of SAT and weighted binary CE schemes are shown in Table 4. One can see that both the schemes can achieve similar results on test data as pretrained ResNets with TenCrop on test data. Moreover, comparing the validation curves, one can find that both SAT and weighted binary CE schemes have stabilizing effects (see Figures 7, 8, and 9), hypothetically attributed to the moving averaged softened labels on SAT and weighted averaged losses on the latter scheme. Another promising phenomenon is that with SAT only applied on good data, the loss and accuracy validation curves improve comparing to their counterparts using a conventional cross entropy loss function, from which one can deduce either that noisy labels in the training dataset probably have negative impact on model learnability or that SAT on good-labelled images assigns smaller weights on them, thus defect-labelled images have heavier weights on the loss, to some degree alleviating the imbalance of the dataset. The former is corroborated by the fact that the average loss of foreign material images diminishes for all three SAT schemes, while the latter is corroborated by the fact that SAT bad boost that assign five times the weights on some defect classes, thus helping nearly eliminate the losses on certain classes (e.g. pocket damage).

Loss Function			Cross Entropy	WBCE	SAT
AUC%			99.82	99.87	99.88
TPR$\geq$ %	99.1	FPR	0.98	1.18	1.38
	99.3		1.05	1.21	1.47
	99.5		1.47	9.08	2.79
	99.7		1.64	12.91	4.95
	100		65.43	26.41	31.62

Table 4. ResNet34 is trained from scratch for 200 epochs using the trhee afore-mentioned loss functions (WBCE means weighted binary cross entropy loss introduced in section 3.6). Unfortunately, neither FiveCrop nor TenCrop is doing the trick here to further suppress the FPR when TPR is 100%.

4.2 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 10 on the left), while the others with obvious defects escape the models' attention for unknown reasons (see Figure 10 on the right). It is unfortunate that with all the afore-mentioned methods applied, these few images always succeed in escaping the investigative power of our models.

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[5] might help; and
2. the failure is attributed to lack of information (epistemic uncertainty), so more data with similar defects might explain it away[5].

Reproducible codes can be found at the following website:

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

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 5. Comparison between the test result of ResNet18 and that after correcting 46 wrongly labelled defect images with the highest defect scores.

5 Future Studies

Several directions remain unexplored concerning this project:

1. A combination of different cropping styles, weighted binary CE loss and SAT schemes with different early stopping timing and sample weights for different classes can be experimented to obtain the optimal setting for the Nexperia data.
2. The model failures mentioned in Section 4.2 requires further studies.
3. The distributional shift issue will be studies in the near future so that a model trained with only one or a few batches of data can be applied to data in more environments, which are in this case devices manufactured by different machines at different time or even different types of semiconductor devices.

6 Conclusion

On this stage, we experiment 7 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 ResNet50 with TenCrop on test data topping the rank.

Different cropping styles during training can help stabilize FPR.

Furthermore, most deep learning models studied are able to spot noisy labels. An SAT scheme is empirically proven useful in alleviating the negative impact of noisy labels on model performance. However, there are defects (see Figure 10) all the models fail to detect. The reasons for the failures remain unexplored. On the next stage, we aim for attacking these failures, further improving current models, and diving into the distributional shift problem to adapt current models to more diverse datasets.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[2] L. Huang, C. Zhang, and H. Zhang, “Self-adaptive training: Beyond empirical risk minimizatio,” *arXiv preprint arXiv:2002.10319*, 2020.

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

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

[5] 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.

[6] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[7] M. Li, M. Soltanolkotabi, and S. Oymak, “Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 4313–4324.

[8] S. Liu, J. Niles-Weed, N. Razavian, and C. Fernandez-Granda, “Early-learning regularization prevents memorization of noisy labels,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.

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

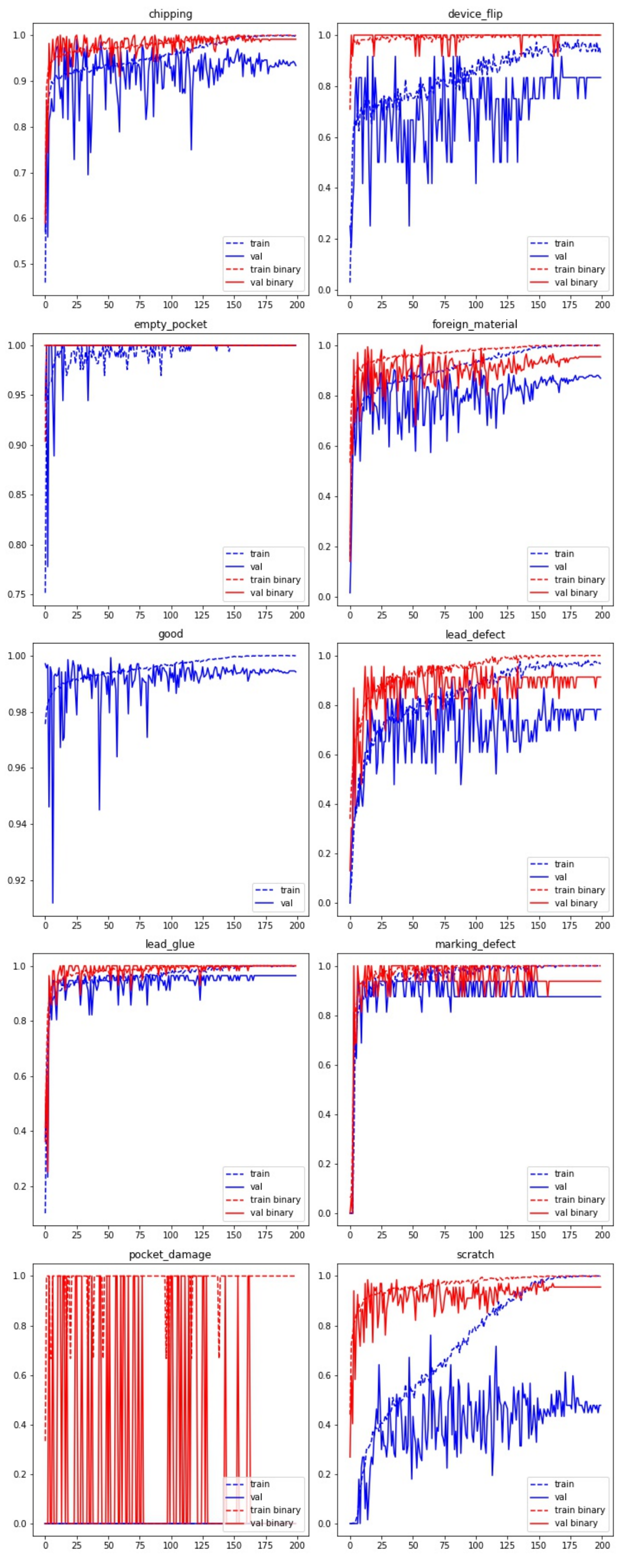


Figure 4. train and validation curves of averaged accuracy rate over 10 classes and binary accuracy of good and defect classes throughout 200 training epochs using CE loss are displayed for each of the ten classes, calculated by dividing the correctly predicted number of images over all images of the corresponding class.

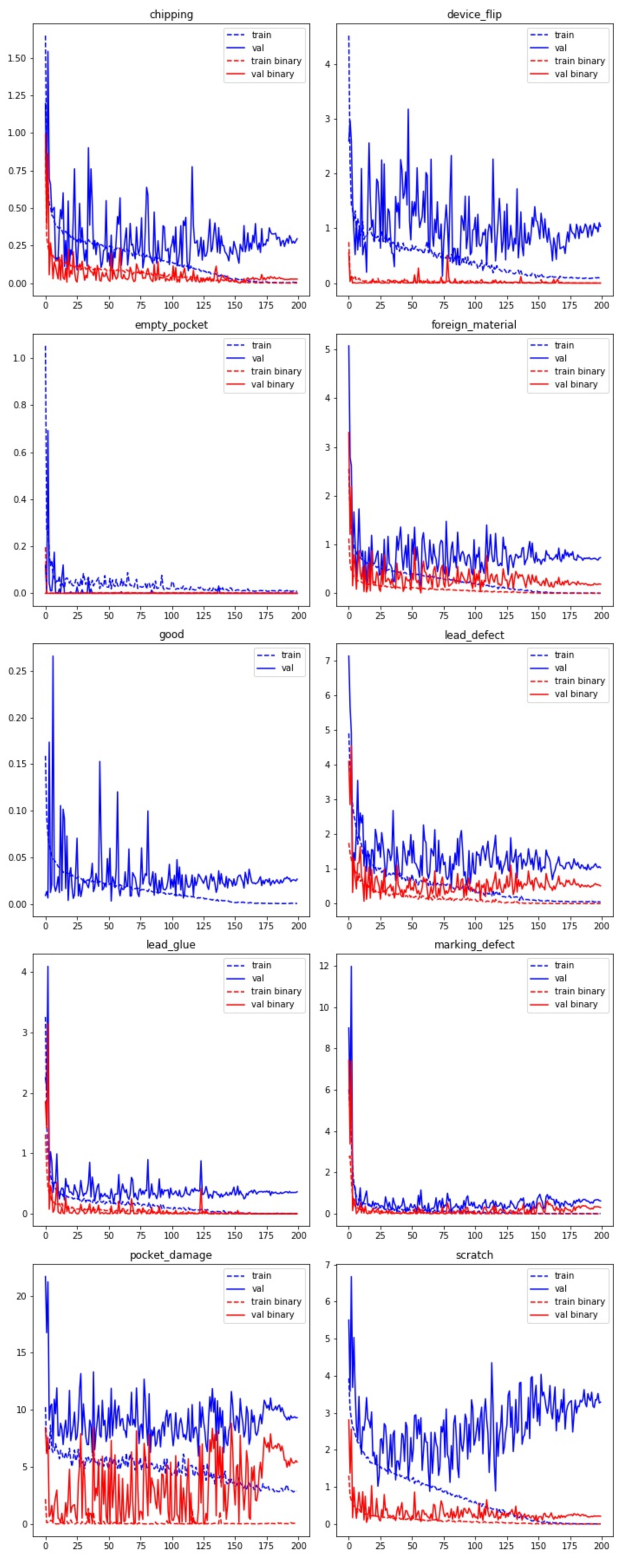


Figure 5. train and validation curves of averaged loss over 10 classes and binary loss of good and defect classes throughout 200 training epochs are displayed for each of the ten classes.

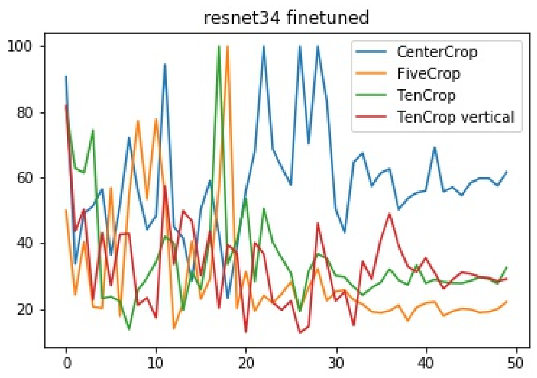


Figure 6. Different cropping styles are experimented during finetuning ResNet34. When TPR is 100%, FPR is very unstable and as high as nearly 60% for a simple CenterCrop, while both FiveCrop and TenCrop help suppress and stabilize the FPR, where a FiveCrop drags the FPR to as low as around 20% after around 35 epochs.

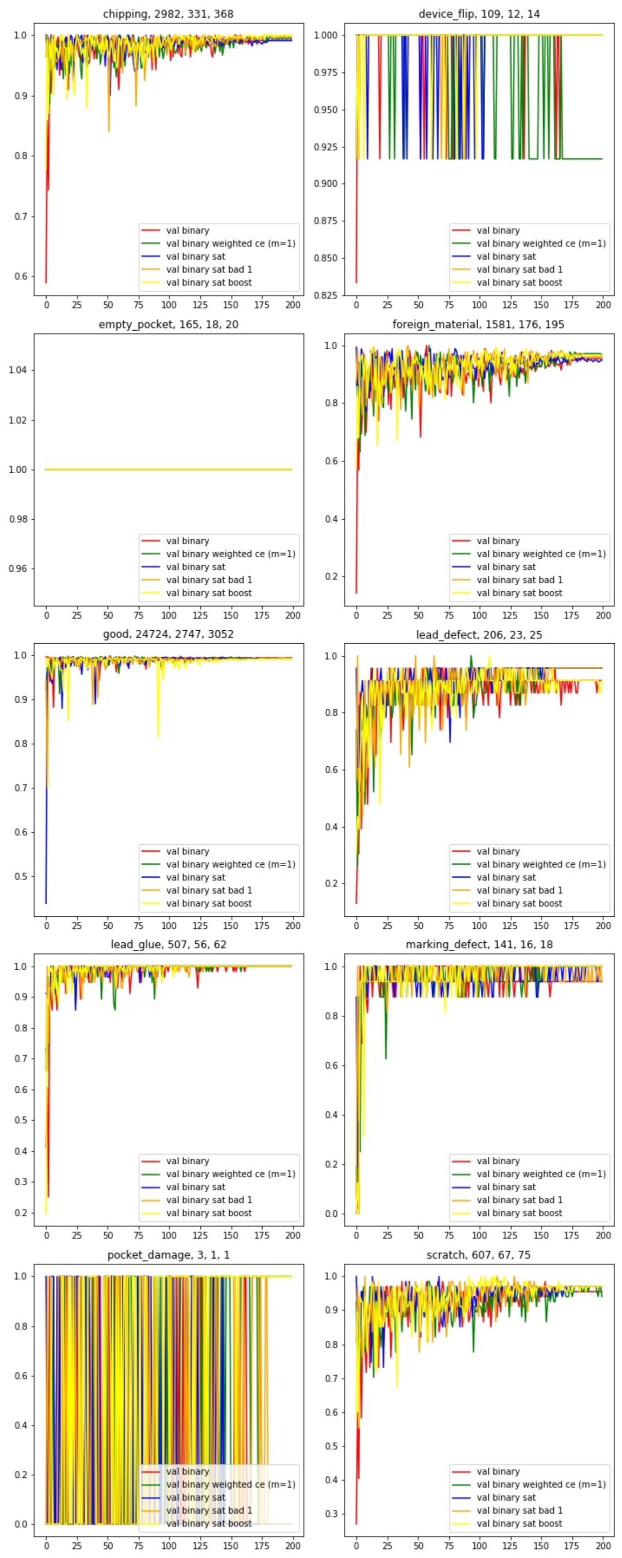


Figure 7. validation curves of averaged binary accuracy of good and defect classes throughout 200 training epochs using CE loss, weighted binary CE loss, SAT, SAT bad 1, and SAT boost, respectively, are displayed for each of the ten classes, calculated by dividing the correctly predicted number of images over all images of the corresponding class. Each of the three numbers next to each class indicates the number of images in train, validation, and test datasets of the corresponding class, respectively.

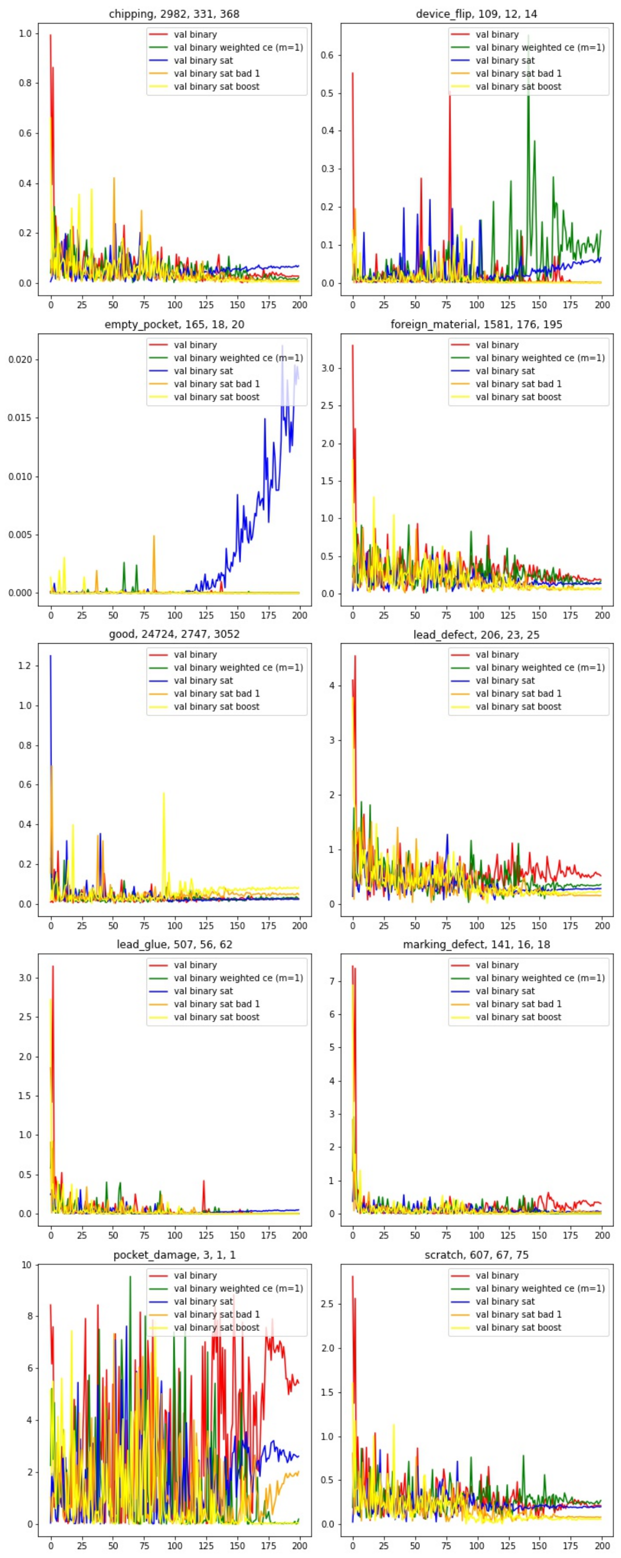


Figure 8. validation curves of averaged binary loss of good and defect classes throughout 200 training epochs using CE loss, weighted binary CE loss, SAT, SAT bad 1, and SAT bad boost, respectively, are displayed for each of the ten classes. Each of the three numbers next to each class indicates the number of images in train, validation, and test datasets of the corresponding class, respectively.

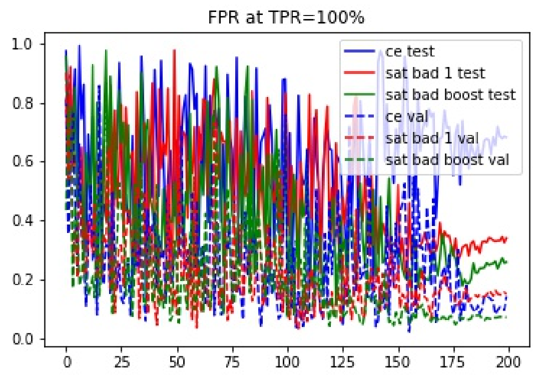


Figure 9. FPR for validation (solid lines) and test (dotted lines) datasets when TPR is 100% throughout the training epochs for each of the CE, SAT bad 1, and SAT bad boost schemes. Both SAT schemes achieve better results than the plain CE scheme, while SAT bad boost gives the best result.

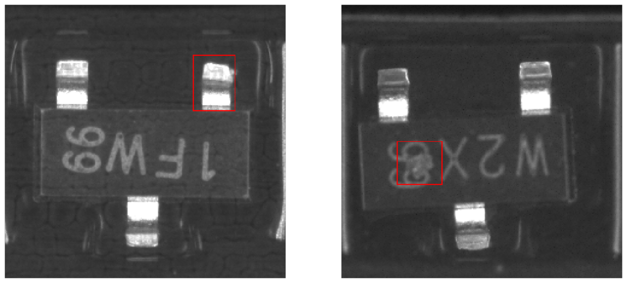


Figure 10. Left: lead defect in the red box; right: scratch in the red box