Toward Fast and Accurate Semi-conductor Image Classification using Deep Convolutional Neural Networks

Tony C. W. Mok
Department of Computer Science
cwmokab@connect.ust.hk

Jierong Wang
Department of Computer Science
jwangdh@connect.ust.hk

Abstract

In this project, we aim to evaluate the image classification performance of the state-of-the-art deep learning-based methods with a mini semi-conductor dataset. Specifically, we compare three classic methods, namely, ResNet, DenseNet, and wide ResNet on the semi-conductor image classification competition on Kaggle. After that, we study the effect of data augmentation and the sampling strategy used in the training phase. Finally, we propose a novel spatial memory system on top of the classical methods and evaluate it comprehensively.

1 Introduction

During the production of semi-conductors, some unqualified devices are mixed with qualified semi-conductors. Conventionally, these unqualified units can be eliminated by human workers via visual checking. However, mass production makes it difficult for human workers to examine all of the devices. Recently, deep learning approaches gain a lot of attention in a variety of tasks, including but not limited to image classification, segmentation, and motion detection.

Nexperia [2], one of the biggest Semi-conductor companies in the world, provided a dataset for the Kaggle [1] in-class contest that aims to classify image of semi-conductor devices into two classes: good and defective. The Nexperia image dataset in the Kaggle contest contains 34457 training images, in which 27420 and 7039 are labeled as good and defect respectively. In this project, we aim to we aim to evaluate the image classification performance of the state-of-the-art deep learning-based methods with a mini semi-conductor dataset, which provided by Nexperia. The Nexperia image dataset imposes two critical challenges. First, the number of "good" semi-conductor devices is almost 4 times the number of "bad" semi-conductor devices, which makes the learned model bias to the "good" semi-conductor devices. Second, the dataset contained a misclassified ground truth label in the training dataset. Therefore, we address these issues by 1) evaluating multiple state-of-the-art image classification methods using the Nexperia image dataset; 2) pre-process the training data manually to clean the data. 3) propose a novel Spatial Memory Module to mimic the memory function of the human brain on top of the existing convolutional neural network.

1.1 Data visualization

Figure 1 shows the example images from the Nexperia image dataset. A defected devices often has scratch/artifact on the surface. A minority of defected devices are absent in the image. On the contrary, a "good" devices should show no scratch on the surface and no damage on the metal parts. During the visual checking, we found around 400 "good" devices are misclassified as "defect" in the training set. As modern deep learning models are capable to generalize well on noisy dataset. We assume this problem is minor during the training. The result in table 1 supports our assumption.

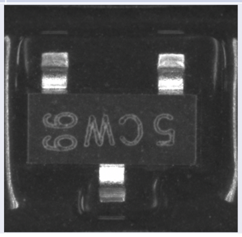
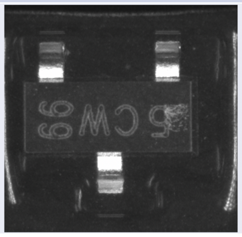
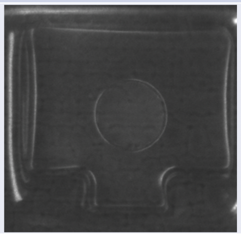
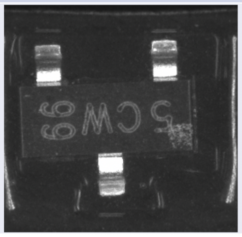
File name	WEA938001D4A_01-5CW-ITISA49-1_54_3	WEA938001H2A_56-5CW-ITISA49-1_34_1	2_387_1	WEA938001H2A_21-5CW-ITISA49-1_321_3
label	Good	Defect	Defect	Good (misclassified)
Image				

Figure 1: Example images from the Nexperia image dataset.

2 Methods

3 State-of-the-art image classification models

Since the number of training data in the Nexperia image dataset is limited, we decide to leverage existing pre-trained deep learning models for image classification. Specifically, we utilize three different state-of-the-art pre-trained models, namely, ResNet [3], DenseNet [4], and wide ResNet [7] as the baseline in this project. Each model is pre-trained on the ImageNet dataset, which contains images with 1000 classes. For each model, we change the last layer (i.e. fully connected layer) of the model to 2 classes to fit our task. Then, we will finetune the whole model with the Nexperia image dataset. The detailed methodology of each method will be discussed below.

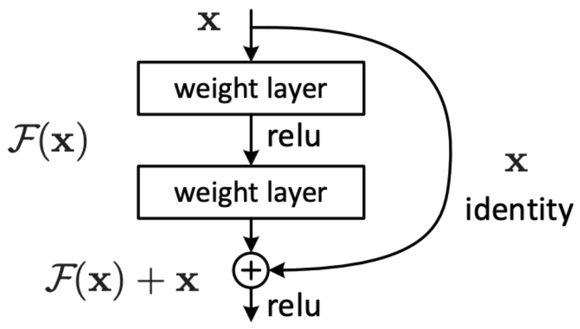

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2: An example of ResNet block[3].

3.1 ResNet

Empirically, the depth of neural networks is essential for the model performance. After adding more layers, the network can extract more complex features and thus can have better results on various tasks. It is true for narrow neural networks. However, for deep neural networks, degradation problem will be exposed when the network depth increases, resulting in larger errors on both training set and test set. It is an optimization problem, which reflects the difficulty in optimizing models with similar structures is different and does not increase linearly with depth.

ResNet was proposed [3] to solve above problem by adjusting the network structure, which essentially change the error surface. A deep residual learning framework was presented and a basic unit, named block, was constructed by stacking few nonlinear layers. For each block, let $F(x)$ denote the function it can fit and $H(x)$ denote the desired underlying map. Instead of training the network to let $F(x)$ fit $H(x)$ directly, ResNet explicitly let the block fit a residual mapping of $F(x) := H(x) - x$. The original mapping is then recast into $F(x) + x$, which can be realized by feedforward neural networks with short connection. The short connection is named as shortcut in ResNet, which is an identity mapping and their outputs are added to the outputs of the stacked layers by element-wise addition, as shown in Fig. 2. The element-wise addition requires the sizes of $F(x)$ and x should be matched, therefore a 1×1 layers can be added at the top of stack, which is used for reducing and then increasing dimension. Such block is named as bottleneck design in the original paper.

By adding shortcut, the error surface of ResNet is more smooth and its gradient can be more predictable (Fig. 3).Therefore, the training of ResNet can be easier and the degradation problem is solved. In the experiments, we use ResNet-50.

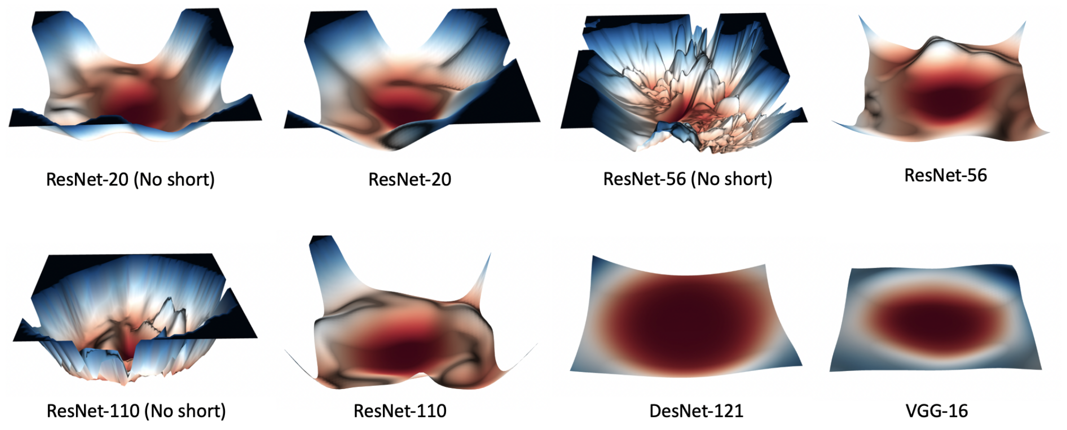


Figure 3: Error surface. "No short" means no shortcut in the network. Each error surface figure is generated from http://www.telesens.co/loss-landscape-viz/viewer.html

3.2 DenseNet

The core idea of DenseNet [4] is similar to the ResNet, both of them create short paths from early layers to later layers.. Instead of using shortcut to establish the connection of among two adjacent blocks, DenseNet connects the current layers with all previous layers, including the input layer (Fig. 4). By using such dense connection, each layer can directly accept the gradients from the loss function and the input signal, resulting in an implicit deep supervision and relieving gradient vanish. Moreover, feature re-usage is achieved by concatenation among different blocks.

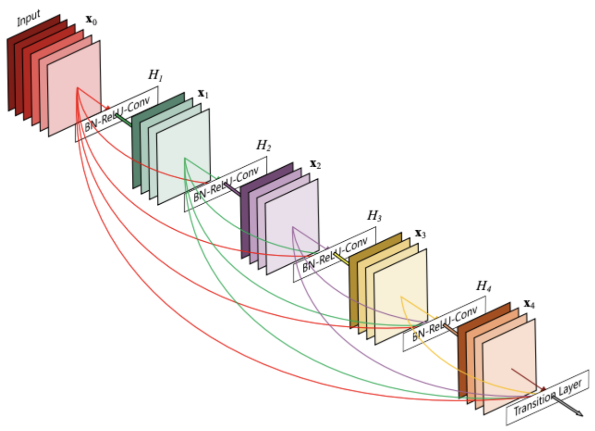


Figure 4: A sketch of DenseNet. This figure is captured from [4]

Denote x_l is the output of l-th layer, H_l is a nonlinear transformation. For ResNet, the output of l-th layer is obtained by adding the output of $l-1$-th layer to the nonlinear transformation of the output of $l-1$-th layer,

$$x_l = x_{l-1} H_l(x_{l-1}). \tag{1}$$

As for DenseNet, denote $[x_0, x_1, ..., x_{l-1}]$ as the concatenation of feature maps produced in 0-th, 1-th, ..., $l-1$ layers, the output of l-th layer is given by,

$$x_l = H_l([x_0, x_1, ..., x_{l-1}]), \tag{2}$$

where H_l consists of a batch norm unit, a ReLU unit and a 3×3 convolution layer. Similar to ResNet, the bottleneck design can be added to dense block to reduce the computation load. To further improve model compactness, DenseNet also introduces a transition layers between two dense block, which is a 1×1 convolution layer. We use DenseNet-121 during our experiments.

3.3 Wide ResNet

Despite the success of ResNet, there are some problems existing in such structure. During back-propagation, gradient can flow through the networks by shortcut, which is an identity mapping. That may result in only limited layers can learn the knowledge, while many blocks share little information and make small contribution to the final goal. This problem is also known as diminishing feature reuse.

Wide ResNet [7] tries to solve the above issue by widening the neural network, that is, by increasing the number of output channels. In the original paper, a factor k is used to determine the network width, which appears in each layer except for the first layer. When $k = 1$, it becomes the original ResNet.

However, when the network width increases, the number of parameters will increase exponentially. To avoid overfitting, wide ResNet inserts dropout between two convolution layers within the residual block. As shown in Fig. 5,(a) is the basic residual block used in ResNet, (b) is the block using bottleneck design - the 1×1 convolution layer. (c) is the widen residual block of wide ResNet and (d) is the widen residual block using dropout. In the experiments, we also use 50 layers for wide ResNet.

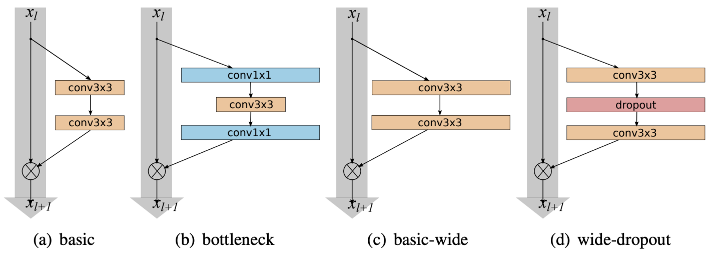


Figure 5: Various blocks. This table is captured from [7]

4 Spatial Memory Module

Inspired from the memory system of human brain, we propose a spatial memory module, which able to learn memorize the error spatially during the training phase. Specifically, we concatenated the 3-channel input image with a learnable 2D tensor, which called spatial memory module, and form a 4-channel input. During the training phase, the error can be directly back-propagate from the last layer to the spatial memory module.

5 Experiments

5.1 Implementation

5.1.1 Visual checking

Since the Nexperia image dataset contains some misclassified semi-conductor devices in the training dataset, we clean the training data manually by visual checking to eliminate the noisy images in the training data. Eventually, we are able to locate 426 misclassified images in the training data. Unfortunately, we found that the training deep learning model with the cleaning data doesn't gain any improvement toward the public score on the test set in Kaggle. The result on validation data using cleaned training data is reported in table 1. We doubt that the reason for this phenomenon is because the testing set is also noisy or the standard of classifying the defective devices is ambiguous by visual checking.

5.1.2 Standard pre-processing

In order to fit our data with the existing pre-trained models, we carry out the standard pre-processing pipeline. The standard pre-processing contains resizing the image to 256×256 pixel, zero-mean 1-std normalization, gray-level to RBG transformation. In particular, we utilize the built-in function provided by *torchvision* to achieve online pre-processing during the training.

5.1.3 Optimization

We implement our method using Pytorch [5] and train our models with a Titian RTX GPU. During the training phase, we finetune the pre-trained models by minimizing the cross-entropy loss between the predicted result and the ground truth label using Adam optimizer [5]. The training dataset is split into a training set (80%) and validation (20%). We select the models with the highest validation score and predict results for the assigned testing set after the training. The source code of this project can be found in `https://github.com/wingwing518/MATH6380p_finalproject`.

5.2 Measurement

To measure the performance of different models on the Nexperia image dataset, area under ROC curve (AUC) is employed. ROC (reciever operating characteristic curve) is a curve using true positive rate (TPR) as y-coordinate and false position rate (FPR) as x-coordinate, in which,

$$TPR = \frac{TP}{TP+FN}, \quad \text{and} \quad FPR = \frac{FP}{TN+FP} \tag{3}$$

For multiple models comparison, if their ROCs do not have any intersection, then it can be clear to make decision on which model is better by finding the corresponding ROC closest to the left-up point in the coordinate. However, if intersection appears, the decision is not easy to make. By counting the area of ROC, we can have a quantitative measurement to compare the performance of different models,

$$AUC = \frac{1}{M*N}\left(\sum_{i\in positiveClass} rank_i - \frac{M(1+M}{2}\right), \tag{4}$$

where $rank$ is the order of samples when they are arranged in a sorting list from low probability to high probability of being classified as true label, M is number of positive samples and N is the number of negative samples. We can also compute the AUC graphically,

$$AUC = \frac{1}{2}\sum_{i}^{m-1}(x_{i+1}-x_i)(y_{i+1}+y_i), \tag{5}$$

where (x_i, y_i) is the point on ROC. Since ROC is usually over the line $y = x$, the value of AUC is from 0.5 to 1. Being closer to 1 means that the model is better.

5.3 Ablation study

5.3.1 Augmentation

Data augmentation [6] is a popular technique to alleviate the overfitting issues in learning-based methods. We follow this convention when finetuning our models. Specifically, during each training step, we flip the input image horizontally and vertically with probability p in both directions. We set p to 0.5 in our experiment. Although this strategy is simple, the result in table 1 shows a significant gain in accuracy when adopting this strategy.

Table 1: Ablation study of visual checking, data augmentation, and Sampling strategy. The resulting score with validation set is reported in this table.

Model	Visual Checking	Augmentation	Sampling strategy	Validation Score
ResNet-50				0.97421
ResNet-50			✓	0.97834
ResNet-50		✓	✓	0.98378
ResNet-50	✓	✓	✓	0.97011

5.3.2 Sampling strategy

Since the number of "good" image and the number of "defect" image are highly imbalance, we propose a sampling strategy to address this issue. During training, we will randomly draw one sample from "good" images or "defect" images with probability 0.5. With this sampling strategy, we are able to guarantee the number of "good":"defect" images ratio within a mini-batch is roughly equal to 1:1. The effectiveness of this sampling strategy is shown in table 1.

5.4 Result on Kaggle Competition

Table 2: Final scores of each selected model on Kaggle. The public score of this competition is AUC, area under ROC curve. ResNet w/ memory: ResNet with the proposed spatial memory module.

Model	Pre-trained	num. of layer	Public Score
ResNet	✓	18	0.97945
ResNet	✓	50	0.98378
ResNet		50	0.97722
Wide ResNet	✓	50	**0.98608**
DenseNet	✓	121	0.98347
ResNet w/ memory	✓	50	0.98042

Table 2 depicts the results of different approaches to the Nexperia Kaggle competition. The leaderboard of the competition can be accessed in `https://www.kaggle.com/c/semi-conductor-image-classification-second-stage/leaderboard` and our team name is *MATH6380p_Tony*. From table 2, the difference between different models is very small. All approaches can achieve a decent classification performance in this task. Interestingly, the result of wide ResNet (50 layers) outperformance DenseNet (121 layers). Also, we can achieve a decent classification performance even without pre-training as shown in the table. Apart from that, our proposed spatial memory module is not particularly useful for this task. It may because it included extra learning parameters, which may potentially exacerbate the overfitting issue in this case.

6 Conclusion

This research project studied the image classification performance of the state-of-the-art deep learning-based methods with a semi-conductor dataset. Among the three classic methods, we found that wide ResNet is the most robust method in classifying semi-conductor images. Moreover, we studied the data augmentation and sampling strategy in the training phase for ablation study. Experiments show that both techniques are useful for improving the model performance. Finally, we also propose a novel technique named spatial memory system on top of the classic models, by which the training error can be flow from the last layer to the spatial memory module directly.

7 Individual Contribution

- **Tony C. W. Mok**: Report writing; ResNet implementation; Wide ResNet implementation; Data augmentation; Spatial Memory System

- **Jierong Wang**: Report writing; DenseNet implementation; Sampling strategy and data preprocessing;Spatial Memory System

References

[1] Kaggle's homepage. `https://www.kaggle.com/`. Accessed: 2020-12-11.

[2] Nexperia's homepage. `https://www.nexperia.com/`. Accessed: 2020-12-11.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[5] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[6] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[7] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.