# Use 3D Convolutional Neural Network to Inspect Solder Ball Defects

Bing-Jhang Lin(✉), Ting-Chen Tsan, Tzu-Chia Tung, You-Hsien Lee, and Chiou-Shann Fuh(✉)

Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan
lknight8631@gmail.com, fuh@csie.ntu.edu.tw

**Abstract.** Head-In-Pillow (HIP) is a solder ball defect. The defect can be caused by surface of solder ball oxidation, poor wetting of the solder, or by distortion of the Printed Circuit Board (PCB) by the heat of the soldering process. The current diagnosis of the HIP defects is difficult to find out the problems, and some destructive tests are not recommended for use. In this paper, we use different angles of 2D X-Rays images to reconstruct the 3D PCB volumetric data. We crop the 3D solder balls volumetric data to 46 × 46 × 46 pixels from the 3D PCB model. Because HIP problems do not happen often, we use the data augmentation method to expand our solder ball data. We propose a new 3D Convolutional Neural Network (CNN) to inspect the HIP problems. Our network uses convolutional blocks that consist of different convolutional paths and the dense connectivity method to connect the blocks. The network can learn various features through these convolutional blocks with different convolutional paths. Moreover, the features of each layer will be fully utilized in the following layers by the dense connectivity method, and also avoid some features lost in a deep convolutional path. In the last layer of our network, the global average pooling can let our network process more different sizes of solder ball data, and the normalized same size vector can be used to do the end-to-end learning. Compared with other classic models, our network not only has fewer parameters but also has faster training time.

**Keywords:** 3D object recognition · 3D CNN · Deep learning
Head-in-Pillow problems

## 1 Introduction

Head-in-Pillow (HIP) is a solder ball defect. The defect can be caused by surface of solder ball oxidation or poor wetting of the solder, or by distortion of the printed circuit board (PCB) by the heat of the soldering process [6]. The current diagnosis of the HIP defects usually uses 2D X-Ray inspection machine or Burn/In method to filter out boards with HIP. Since most X-Rays can only check from the top to the bottom, it is hard to find out the location of defects. The Burn/In method requires additional costs. In addition, the more reliable methods are destructive tests, but they are not recommended for use.

It is hard to find out the location of HIP defects from 2D X-Rays images, but we can reconstruct the 3D solder ball model from different angles of 2D X-Rays images. The 3D solder ball model can not only provide more information but also represent the location of HIP defects more clearly. In recent years, Deep Learning has been widely used in many computer vision tasks. In particular, the Convolutional Neural Network (CNN) has an outstanding performance in image object recognition. Different levels of features can be integrated by the deep network structure. Finally, the complex high-level features can be combined with an end-to-end network to predict the result.

In this paper, we reconstruct the 3D PCB model from different 2D X-Ray images (1472 pixels $\times$ 1176 pixels $\times$ 73 layers), and we crop the 3D solder balls to 46 $\times$ 46 $\times$ 46 pixels from the 3D PCB model. We propose a 3D CNN to inspect the HIP problems. Our network combines the advantages of many classic CNNs. We design two types of the convolutional blocks composed of many different convolutional paths and use the dense connectivity method to connect these blocks. Because the convolutional block has many different convolutional paths composed of different kernels of convolutions, the network can learn more features from the convolutional block. To process more different sizes of data, we use the global average pooling to adapt these data. In addition, our network is not only faster and more accurate than other classic CNN models, but also makes the speed of convergence faster during the training stage. Because the HIP problems do not happen often, so we use the data augmentation method to expand the 3D solder ball data. In our experiments, our network can achieve excellent results on testing. The size of our network is more refined, and the training and execution time is faster than other models.

## 2 Related Works

In recent year, many state-of-art CNN models are proposed in the image object recognition. The Visual Geometry Group (VGG) achieved 92.7% top-5 test accuracy in ImageNet in 2014 [8], the ImageNet is a dataset of over 14 million images belonging to 1000 classes. VGG proved that to increase the depth of the network can improve the accuracy of the image recognition, and many papers also used VGG to achieve their recognition tasks [7, 9, 10]. VGG is composed of many consecutive convolutional layers. Therefore, the parameters of the VGG model are not only quite large, but also need to take much time to train this model.

In [11], GoogleNet Inception V1 was the champion on the ImageNet competition in 2014. The Inception V1 block is composed of 1 $\times$ 1, 3 $\times$ 3, and 5 $\times$ 5 convolutions along with a 3 $\times$ 3 max pooling. The network can choose the better features to learn. This architecture allows the model to extract both local feature via smaller convolutions and high abstracted features with larger convolutions. Besides, the 1 $\times$ 1 convolution is a good way to reduce the dimension of the feature maps, and it also represents the raw data well. Because the kernel of the 1 $\times$ 1 convolution has only one parameter, it is easy to scale the dimension of the feature to make it easier to increase the depth of network. Many papers also use the 1 $\times$ 1 convolution to increase the depth of their network [5, 13].

Although increasing the depth of the network can improve the accuracy of the recognition, it is easier for vanishing gradient problem to happen when the network is too deep. This situation will lead to decreased accuracy despite increasing the depth of the network. Simply increasing the depth of the network will not produce very good results, and it will even increase the error. The Deep Residual Neural Network (ResNet) improved the vanishing gradient problem of increasing the depth of the network and made it easier to increase the depth of the network [3]. ResNet proposed a residual method to create a short path to connect earlier layers with later layers. The input of the block not only performs two consecutive convolution processes but also adds to the output in each residual block. This way carries the important information features in previous layer to the later layers, and it also prevents the vanishing gradient problem. Many people do modify the network architecture according to the paper, so that the deep neural network is deeper and more accurate [2, 12].

In 2018, the Densely Connected Convolutional Network (DenseNet) was proposed [4] and used a simple method to increase the depth of the network. This network is one variant of ResNet, but it connects the result of each layer to make the accuracy of the network better. The dense connection makes features of each layer to be fully used, and it also avoids some important feature loss in deep network. The parameters of the network are less than other state-of-art CNNs.

## 3 Approach

### 3.1 3D Solder Balls Volumetric Data

The 3D PCB volumetric data are reconstructed by different angles of 2D X-Rays images. In order to obtain clearer 3D PCB volumetric data during the reconstruction, we control the camera to take a 2D X-Rays image every 2.81 degrees. Figure 1(a) shows the 3D PCB volumetric data reconstructed by 128 different angles of 2D X-Rays images, and Fig. 1(b) shows the data reconstructed by 16 different angles of 2D X-Rays images. In contrast to Fig. 1(b) reconstructed result, Fig. 1(a) result is more clear, and background noise is also less.

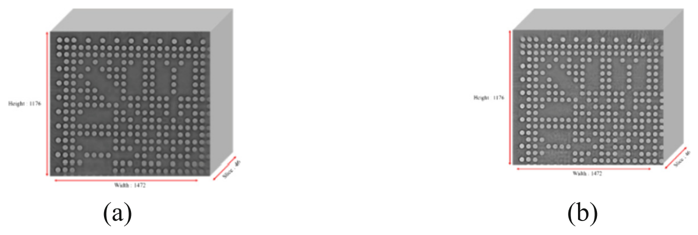


**Fig. 1.** (a) The 3D PCB model reconstructed with 128 different angles of 2D X-rays images. (b) The 3D PCB model reconstructed with 16 different angles of 2D X-rays images.

The 3D PCB volumetric data size is 1472 × 1176 × 46 pixels. Some solder balls on the edge of PCB will be cut, so we exclude those incomplete solder balls. We receive a total of 277 complete solder balls from Fig. 1(a) PCB. Since the volumetric pattern of solder ball is an elliptical sphere, the middle slice of the volumetric contains the biggest solder ball cross section. However, the maximum bounding box size of the solder ball from the middle slice of the PCB volumetric data is 46 × 46 pixels, so the size of each 3D solder ball is 46 × 46 × 46 pixels. Figure 2(a) shows the volumetric pattern of solder ball. Figure 2(b) shows the solder ball pattern in the middle slice for the solder ball volumetric data.

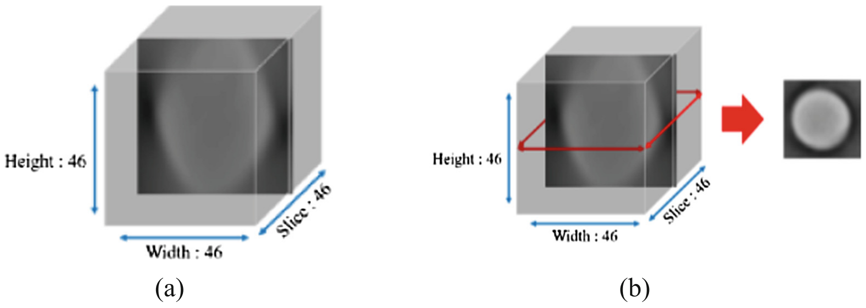


**Fig. 2.** (a) The volumetric pattern of solder ball. (b) Solder ball pattern in the middle slice of the solder ball volumetric data.

In this case, we just receive a total of 277 complete solder balls. There are fewer solder balls with HIP problem. For an image object recognition task, the amount of data is not sufficient, so we also use the Data Augmentation method to obtain more data. The 3D solder ball is a cube-like object whose size is 46 × 46 × 46 pixels. We rotate to expand the data 6 times. In more detail, we have achieved the rotation method by converting between the *x*, *y*, and *z* axes. Figures 3(a) and (b) show the *xy* plane results. Figures 3(c) and (d) show the *yz* plane results. Figures 3(e) and (f) show the *xz* plane results. Besides, we also normalize each solder ball volumetric before inputting to the network for processing.

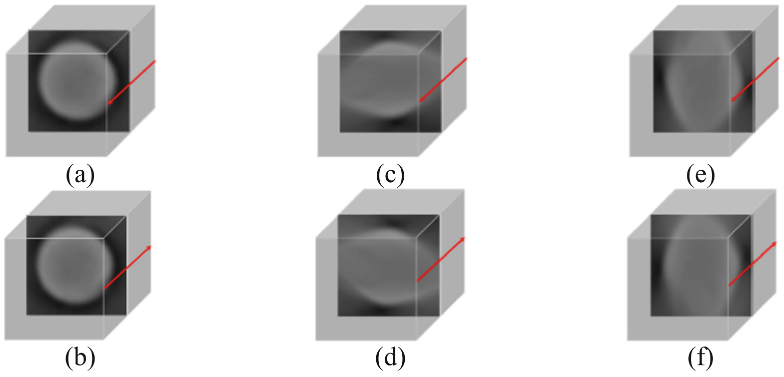


**Fig. 3.** (a) Combination of positively arranged *xy* images. (b) Combination of reversed *xy* images. (c) Combination of positively arranged *xz* images. (d) Combination of reversed *xz* images. (e) Combination of positively arranged *yz* images. (f) Combination of reversed *yz* images.

## 3.2 3D Convolutional Neural Network

In this paper, we reference four state-of-art CNN models (VGG, GoogleNet Inception V1, ResNet, DenseNet) in our solder ball HIP problems [3, 4, 8, 11]. These models have a significant impact on the domain about deep learning. We not only convert these four kinds of 2D CNN models into 3D CNN models to inspect our solder ball HIP problems, but also combine the advantages of these models to propose a new model to inspect the problem in our task. We use dense connectivity method to connect our convolutional blocks. Because the feature maps of a dense connection block will have multiple growths, we use the 1 × 1 × 1 convolution with stride 2 to reduce the dimension of the feature maps. The 1 × 1 × 1 convolution can represent the raw data well and compress the features to reduce the dimension, so that the following layer can process these concise features more quickly and efficiently. Besides, the global average pooling is used to normalize the input tensor to the same size, because we want to let our network adapt to the different sizes of the input. Figure 4 shows our 3D CNN structure to inspect the solder ball HIP problems.

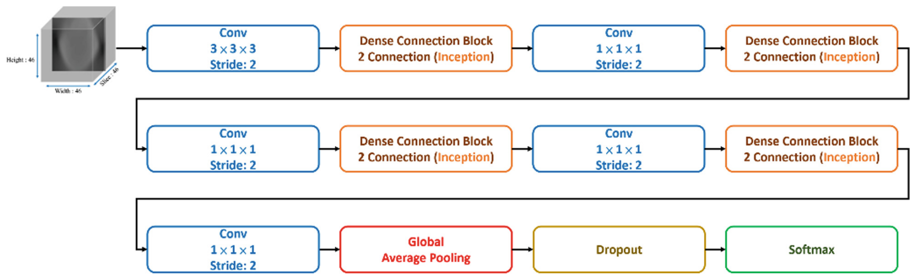
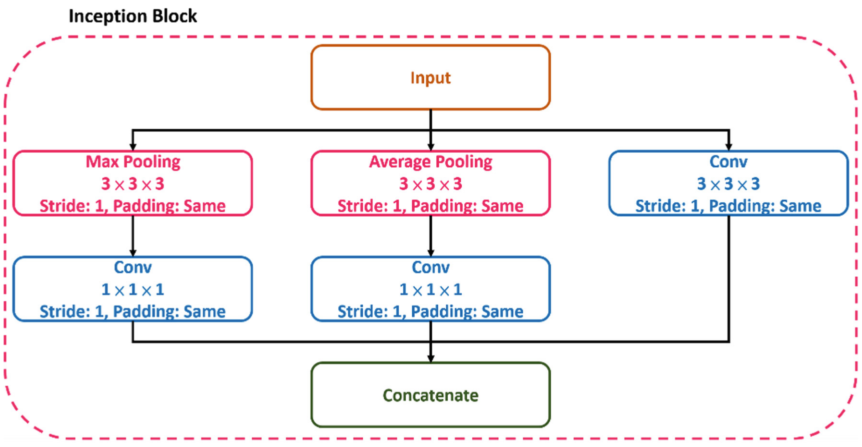

Height : 46
Width : 46
Slice : 46
Conv 3×3×3 Stride: 2
Dense Connection Block 2 Connection (Inception)
Conv 1×1×1 Stride: 2
Dense Connection Block 2 Connection (Inception)
Conv 1×1×1 Stride: 2
Dense Connection Block 2 Connection (Inception)
Conv 1×1×1 Stride: 2
Dense Connection Block 2 Connection (Inception)
Conv 1×1×1 Stride: 2
Global Average Pooling
Dropout
Softmax

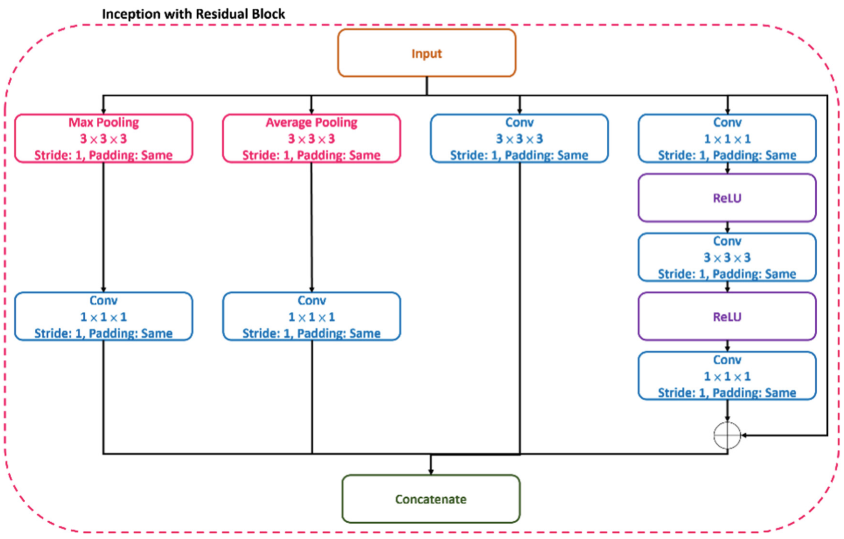
**Fig. 4.** (a) Our network has four dense connection blocks. The amount of each dense connection block is 2, and the convolutional block type is the "Inception". We add a 3D max polling layer with stride 2 with 2 × 2 × 2 kernel and a 3D convolutional layer with ReLU activation function with 1 × 1 × 1 kernel between the middle dense connection block. Last, the global average pooling layer is used to adapt the different sizes of the input and do the end-to-end learning with the following layers.

**Dense Connection Block**

In this paper, we propose two types of convolutional blocks for our network. These two blocks all combine the Inception concept [11], and these blocks are composed of many different convolutional paths. The first convolutional block called "Inception Block" has three convolutional paths. The first path includes a 3D max pooling layer with 3 × 3 × 3 kernel and a 3D convolutional layer with 1 × 1 × 1 kernel. The second path includes a 3D average pooling layer with 3 × 3 × 3 kernel and a 3D convolutional layer with 1 × 1 × 1 kernel. The third path includes a single convolutional layer with 3 × 3 × 3 kernel. Last, we concatenate each result of the convolutional path. We also use the residual method [3] in the second convolutional block. This block is called "Inception with Residual Block". This convolutional block has four convolutional paths, and the first three convolutional paths are the same as our "Inception Block". Besides, we add the fourth convolutional path. This added path consists of a 3D convolutional layer with ReLU activation function with 1 × 1 × 1 kernel, a 3D convolutional layer with ReLU activation function with 3 × 3 × 3 kernel, and a 3D convolutional layer with 1 × 1 × 1 kernel. We add the input to the last of the fourth path to achieve the residual method. Finally, we concatenate each result of the convolutional path. The residual method is a good way to prevent vanishing gradient problem, so we can more easily increase the depth of the network. Adding different convolutional paths allows the following layer to learn various features, so the network can choose better features to adapt the data and give us better results on testing data. Figure 5(a) shows our "Inception Block", and Fig. 5(b) shows our "Inception with Residual Block".


Inception Block
Input
Max Pooling 3×3×3 Stride: 1, Padding: Same
Average Pooling 3×3×3 Stride: 1, Padding: Same
Conv 3×3×3 Stride: 1, Padding: Same
Conv 1×1×1 Stride: 1, Padding: Same
Conv 1×1×1 Stride: 1, Padding: Same
Concatenate

(a)


Inception with Residual Block
Input
Max Pooling 3×3×3 Stride: 1, Padding: Same
Average Pooling 3×3×3 Stride: 1, Padding: Same
Conv 3×3×3 Stride: 1, Padding: Same
Conv 1×1×1 Stride: 1, Padding: Same
ReLU
Conv 3×3×3 Stride: 1, Padding: Same
ReLU
Conv 1×1×1 Stride: 1, Padding: Same
Conv 1×1×1 Stride: 1, Padding: Same
Conv 1×1×1 Stride: 1, Padding: Same
Concatenate

(b)

**Fig. 5.** (a) Our "inception block". (b) our "inception with residual block".

In addition, we use the dense connectivity method [4] to connect our convolutional block. The output of previous layer will be connected with the current output of the convolutional block and do the ReLU activation function. The features of each layer will be fully utilized in the following layers, and also avoid some features lost in a deep convolutional path. Figure 6 shows our dense connection block.

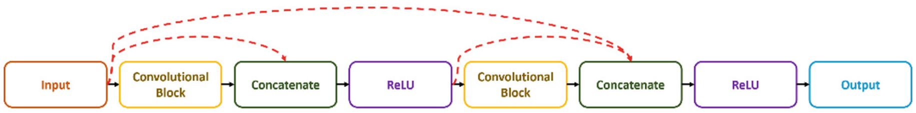


**Fig. 6.** Our dense connection block. The result of current convolutional block will concatenate with the result of each previous convolutional block, then concatenated result will do the ReLU activation function before inputting the next convolutional block.

## 4 Experiments

In order to ensure the accuracy and efficiency of our network, we compare our network with other state-of-art CNN models. We also compare computational acceleration to understand what methods can speed up the time of data preprocessing.

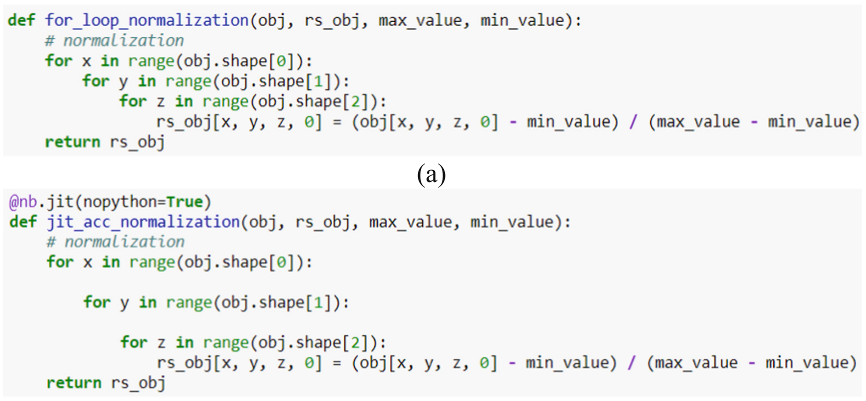
### 4.1 Solder Ball Data

We currently have 277 balls of 3D solder, of which 38 are labeled data and the rest are unlabeled data. The volumetric size of each solder ball is 46 $\times$ 46 $\times$ 46 pixels. Half of labeled data (19) are defective solder balls, and the other are normal solder balls. We split the labeled data into two parts: training and testing. The training part has 32 solder balls and the testing part has 6 solder balls. Because the defective solder ball do not often appear on the PCB, so that the 239 unlabeled data are temporarily considered as normal solder balls in evaluating stage. In addition to cross comparing the results of different models (VGG, GoogleNet Inception V1, ResNet, DenseNet, Our Network), we also provide the data to the experienced experts to check. We can know which solder ball is defective and the accuracy of the different models. The data are the important part in deep learning. The sampling range is wide enough and the sampling density is dense enough that the results of the model can approach the true distribution. Therefore, before we input the data to our network to inspect the HIP problems, we will augment the data to increase the amount of the labeled data. We rotate to expand the labeled data 6 times, so the training part has 192 (=32 $\times$ 6) solder balls, and the testing part has 36 (=6 $\times$ 6) solder balls.

### 4.2 Preprocessing Acceleration Comparison

Although the deep learning tasks are accelerated by the GPU device, the data need to be arranged by the CPU. Therefore, we also pursue the efficiency of CPU computation during preprocessing. Because every pixel in the solder ball volumetric needs to be processed for normalization, our program needs many for-loops to achieve preprocessing. It is a good opportunity to improve the computational efficiency. For speeding up the efficiency, we choose Numba acceleration library [1], and we also use some vectorization techniques to speed up.

Numba can speed up the program with high performance functions written directly in Python. With a few annotations, array-oriented and mathematics-heavy Python code can be just-in-time compiled to native machine instructions, so we do not have to drastically change the architecture of the program. The code section with for-loops is shown in Fig. 7(a), and the code section with Numba is shown in Fig. 7(b).



```python
def for_loop_normalization(obj, rs_obj, max_value, min_value):
    # normalization
    for x in range(obj.shape[0]):
        for y in range(obj.shape[1]):
            for z in range(obj.shape[2]):
                rs_obj[x, y, z, 0] = (obj[x, y, z, 0] - min_value) / (max_value - min_value)
    return rs_obj
```

(a)

```python
@nb.jit(nopython=True)
def jit_acc_normalization(obj, rs_obj, max_value, min_value):
    # normalization
    for x in range(obj.shape[0]):

        for y in range(obj.shape[1]):

            for z in range(obj.shape[2]):
                rs_obj[x, y, z, 0] = (obj[x, y, z, 0] - min_value) / (max_value - min_value)
    return rs_obj
```

(b)

**Fig. 7.** (a) The code section with for-loops. (b) The code section with Numba.

The comparisons of normalization of execution times are shown in Table 1. We compare the execution times for these methods for normalization under different quantities of data. In the small data, there is not much difference between Numba and for-loops, but the difference increases as the data increase. Numba is slightly faster than vectorization in performance. Table 1 also shows total execution time of normalization of three methods. We can get approximately about 30 times (= 15.60453/0.56150) acceleration by Numba, and about 50 times (= 15.60453/0.35498) acceleration by vectorization.

**Table 1.** The comparisons of normalization of execution times with different quantity of data.

| Quantity of data | For-loops (sec.) | Numba (sec.) | Vectorization (sec.) |
|---|---|---|---|
| 4566 (testing 6 balls) | 0.34192 | 0.32136 | 0.01253 |
| 24352 (training 32 balls) | 1.80130 | 0.02757 | 0.04110 |
| 181879 (testing 239 balls) | 13.46030 | 0.21156 | 0.30033 |
| Total time | 15.60453 | 0.56150 | 0.35498 |

The comparisons of data augmentation of execution times are shown in Table 2. Because we only augment data on labeled data, we only use training and testing data for comparison. Table 2 shows the total execution time of data augmentation of three methods. We can get approximately about 10 times (= 4.03928/0.35697) acceleration by Numba, and about 30 (= 4.03928/0.12433) times acceleration by vectorization.

**Table 2.** The comparisons of data augmentation of execution times with different quantities of data.

| Quantity of data | For-loops (sec.) | Numba (sec.) | Vectorization (sec.) |
|---|---|---|---|
| 4566 (testing 6 balls) | 0.68535 | 0.20957 | 0.01904 |
| 24352 (training 32 balls) | 3.35342 | 0.14690 | 0.10528 |
| Total time | 4.03928 | 0.35697 | 0.12433 |

## 4.3 Experimental Result

We use the Tensorflow 1.5 and CUDA 9.0 to construct our deep learning development environment, and Python 3.6 as our development language. In addition, we use Nvidia GTX 1060 6G as our GPU device, and Intel i7-7700HQ as our CPU processor.

In training stage, we set the batch size to 5 to train models, because the volumetric size is too large that we cannot set higher batch size at one time. The batch size 5 is the most suitable for our development environment and device. Besides, we also discover that some deep networks have to use the lower learning rate for training. Because it has to pass more layers to back-propagate the error, if the learning rate is too high, it may fall into a local optimal solution and achieve poor results. However, the dense connectivity method makes each layer connected to other layers in each dense connection block, so the gradient will not disappear in the deep network. Table 3 shows the information of each model. VGG16 has the largest model size, because it uses many continuous convolutional layers. These layers also have larger amount of the kernels such as 64, 128, 256, and 512. In 3D convolution the $3 \times 3 \times 3$ kernels have 27 parameters that are 3 times more than 2D convolution kernel, so the size of the model parameter also shows significant growth. The time of the training model is proportional to the size of the model. The larger size of model, the more time it takes to train. Table 3 shows the information of each model.

**Table 3.** The information of each model.

| Model | Depth | Training (sec.) | Learning rate | Size of model (KB) |
|---|---|---|---|---|
| ResNet50 | 50 | 1099.3151 | 0.00001 | 540568 |
| GoogleNet inception V1 | 22 | 1286.3257 | 0.00001 | 460280 |
| VGG16 | 16 | 5149.8747 | 0.00001 | 910582 |
| DenseNet | 21 | 406.5984 | 0.0001 | 6945 |
| Our network (inception block) | 18 | 574.8570 | 0.0001 | 45318 |
| Our network (inception with residual block) | 26 | 848.4092 | 0.0001 | 70449 |

Table 4 shows the results on testing data. We use the cross entropy as the loss function in this experiment. Our network has the best performance on testing data, and the processing time of the network is shorter. Because the network can learn various features from different kernels of the convolution, it can choose the better feature to adapt the data to achieve better results. GoogleNet Inception V1 also has good accuracy by using different kernels of convolution in network. Besides, the dense connectivity method is a good way to avoid some important features lost in deep network, and the network which uses this way also has good accuracy and speed. Although depth of ResNet50 is very deep, its result is not ideal. Perhaps each residual block using the single shortcut path to connect previous layer with later layer, the network only learns the features from the previous layer, so it is hard to let the network to learn more features to adapt data. VGG16 has decent results, but its size and execution time are quite large compared with other networks.

**Table 4.** The result on testing data which uses cross entropy as loss function.

| Model | Execution time (sec.) | Loss | Accuracy |
|---|---|---|---|
| VGG16 | 1.52800 | 1.805191 | 86% |
| ResNet50 | 0.47715 | 1.939934 | 83% |
| DenseNet | 0.23764 | 0.955081 | 83% |
| GoogleNet inception V1 | 0.61803 | 0.000997 | 100% |
| Our network (inception block) | 0.30885 | 0.000093 | 100% |
| Our network (inception with residual block) | 0.44718 | 0.000000 | 100% |

Table 5 shows the results on unlabeled data. In the unlabeled data, our network still achieves the best accuracy and faster execution time. The networks use the dense connectivity method will improve the performance. The important features will be preserved by this way, and those features are closer to real data.

**Table 5.** The results on 239 unlabeled solder balls.

| Model | Execution time (sec.) | Accuracy |
|---|---|---|
| VGG16 | 13.86638 | 84% |
| ResNet50 | 5.99745 | 80% |
| DenseNet | 2.33119 | 85% |
| GoogleNet inception V1 | 6.98257 | 83% |
| Our network (inception block) | 2.86511 | 86% |
| Our network (inception with residual block) | 4.29642 | 87% |

When the patterns of these solder balls are skewed near PCB border, we find that the models may misclassify the normal solder balls into defect. The normal solder ball does not have unfused part in the pattern, but we can easily find the overlapping patterns on the defect solder ball in Fig. 8. The skew situation often happens on the PCB border, and the pattern of skew solder ball is blurry which may cause misclassification.

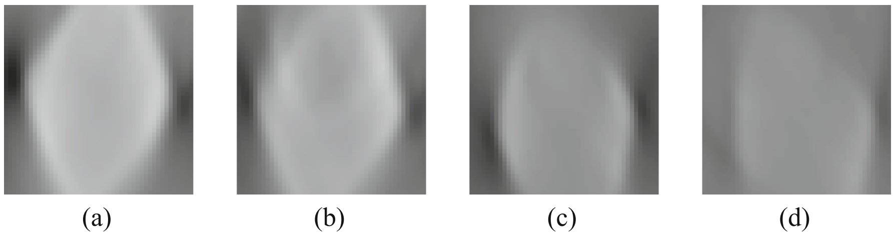


**Fig. 8.** (a) Pattern of the normal solder ball. (b) Pattern of the defect solder ball. (c) Skew normal solder ball near PCB border. (d) Skew defect solder ball near PCB border.

## 5 Conclusions and Future Work

In this paper, we propose a 3D CNN to inspect the HIP problems. Our network combines the advantages of many state-of-art CNN models. We design two types of the convolutional block, and these blocks are composed of many different convolutional paths. Because the network can learn various features from different convolutional paths, so it can choose better features to adapt the data to achieve the better result on testing data. Besides, we reference the dense connectivity method to connect the output of each layer. The previous layers will be connected to the latter layers in each dense connection block, so the features can be fully used in the network. It is a good way to avoid the important features loss in the deep convolutional paths. The global average pooling layer can help our network to adapt many different sizes of data, so our network can use the normalized same size vector to do the end-to-end learning to process more different data. In our experiments, our network has almost the fewest parameters compared with other classical models, and the convergence speed of our network is also the fastest. The accuracy of our network can achieve 100 % on the testing data, and the accuracy of our network is also the best on the unlabeled data.

Data are very important in deep learning. If the sampling range is wide enough and the sampling density is dense enough, then the model can reflect the true distribution of the real situation. Although our network can achieve the astonishing accuracy on the testing data, the amount of the 3D solder balls is not enough. Therefore, we use data augmentation to expand our data before training the network, but these data are still insufficient to summarize the real situation. HIP problems seldom happen. Solder balls on PCB border are often skew and may cause misclassification. In the future, we will not only continue collecting more information on solder balls, but also focus on using those data reconstructed by different angles of the 2D X-Rays images to train the network. We will still endeavor to simplify and accelerate our network to inspect HIP problems.

**Acknowledgement.** This research was supported by the Ministry of Science and Technology of Taiwan, R.O.C., under Grants MOST 104-2221-E-002-133-MY2 and MOST 106-2221-E-002-220, and by Test Research, Jorgin Technologies, III, Egistec, D8AI, and LVI.

## References

1. Numba, A.: https://numba.pydata.org/ (2018)
2. Bi, L., Kim, J., Ahn, E., Feng, D.: Automatic skin lesion analysis using large-scale dermoscopy images and deep residual networks. arXiv preprint arXiv:1703.04197 (2017)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
4. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. arXiv preprint arXiv:1608.06993 (2018)
5. Iandola, F.N., Shen, A., Gao, P., Keutzer, K.: DeepLogo: hitting logo recognition with the deep neural network hammer. arXiv preprint arXiv:1510.02131 (2015)
6. Seelig, K.: Head-in-Pillow BGA Defects. AIM Solder, Montreal (2008)
7. Sercu, T., Puhrsch, C., Kingsbury, B., LeCun, Y.: Very deep multilingual convolutional neural networks for LVCSR. arXiv preprint arXiv:1509.08967 (2016)
8. Simonyan, K., Zisserman, Z.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2015)
9. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.: Multi-view convolutional neural networks for 3D shape recognition. arXiv:1505.00880 (2015)
10. Sun, Y., Liang, D., Wang, X.G., Tang, X.O.: DeepID3: face recognition with very deep neural networks. arXiv preprint arXiv:1502.00873 (2015)
11. Szegedy, C., et al.: Going deeper with convolutions. arXiv preprint arXiv:1409.4842 (2014)
12. Targ, S., Almeida, D., Lyman, K.: Resnet in Resnet: generalizing residual architectures. arXiv preprint arXiv:1603.08029 (2016)
13. Zhong, Z.Y., Jin, L.W., Xie, Z.C.: High performance offline handwritten chinese character recognition using GoogLeNet and directional feature maps. arXiv preprint arXiv:1505.04925 (2015)