sensors

Article

A Compact Convolutional Neural Network for Surface Defect Inspection

Yibin Huang [1,†], Congying Qiu [2], Xiaonan Wang [1,*], Shijun Wang [1] and Kui Yuan [1]

1 Institute of Automation, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100190, China; huangyibin2014@ia.ac.cn (Y.H.); wangsj_ucas2012@163.com (S.W.); kui.yuan@ia.ac.cn (K.Y.)
2 Civil Engineering & Engineering Mechanics Department, Columbia University, New York, 10024, USA; cq2192@columbia.edu
* Correspondence: wangxiaonan2012@ia.ac.cn
† Current address: No. 95 Zhongguancun East Road, Beijing 100190, China

Received: 9 February 2020; Accepted: 29 March 2020; Published: date

Abstract: The advent of convolutional neural networks (CNNs) has accelerated the progress of computer vision from many aspects. However, the majority of the existing CNNs heavily rely on expensive GPUs(graphics processing units). to support large computations. Therefore, CNNs have not been widely used to inspect surface defects in the manufacturing field yet. In this paper, we develop a compact CNN-based model that not only achieves high performance on tiny defect inspection but can be run on low-frequency CPUs(central processing units). Our model consists of a light-weight (LW) bottleneck and a decoder. By a pyramid of lightweight kernels, the LW bottleneck provides rich features with less computational cost. The decoder is also built in a lightweight way, which consists of an atrous spatial pyramid pooling(ASPP) and depthwise separable convolution layers. These lightweight designs reduce the redundant weights and computation greatly. We train our models on groups of surface datasets. The model can successfully classify/segment surface defects with an Intel i3-4010U CPU within 30ms. Our model obtains similar accuracy with MobileNetV2 while only has less than its 1/3 FLOPs(floating-point operations per second) and 1/8 weights. Our experiments indicate CNNs can be compact and hardware-friendly for future applications in the automated surface inspection (ASI).

Keywords: surface defect inspection; convolutional neural network; machine vision.

1. Introduction

For a long time, product quality control relies on manual examination in the manufacturing field. In the past several decades, automated surface inspection (ASI) have become more widespread as hardware and software technologies have developed rapidly. To lower labor costs and enhance examination efficiency, more factories start to employ embedded machines for product inspection. ASI platforms use special light sources and industrial cameras to capture the images of the product surface, and machine/computer vision technologies to filter out defective products, which can reduce labor greatly. High-performance cross-product ASI algorithms are urgently needed in manufacturing.

Common surfaces in the manufacturing industry include steel [1,2], glass [3], fabric [4], wood board [5,6], and so forth. Most proposed defect inspection algorithms for these surfaces are traditional classification or segmentation methods. For traditional classification methods, for example, Gabor [7], LBP(local binary pattern) [8], GLCM(gray-level co-occurrence matrix) [9] and HOG(histogram of oriented gridients) [10] features of the defects are extracted, then the defects are classified by machine learning classifiers,, such as SVM(support vector machines) [1], Random Forests [11], AdaBoost [10], and so forth. Traditional segmentation methods would partition pixels into regions, with the intention to emphasize the regions corresponding to surface defects. For example, Reference [6] firstly preprocesses images using convex optimization, then segments wood defects with Otsu segmentation method. Reference [12] segments fabric defects by the Sobel and the watershed algorithm.

These traditional methods involve image preprocessing, feature extraction, feature reduction, and classifier selection, which needs the experience of experts. Besides, most surface defects are tiny and very similar to their background, even the inspection for a single type of defect can be very difficult for these traditional methods. Although traditional algorithms are designed for specific surfaces, effective measures such as recall, precision, and accuracy can barely achieve the standard of industrial applications. This long-existing bottleneck has been restraining the growth of ASI.

Convolutional neural networks (CNNs) greatly stimulate the development of ASI [3,13–15] as deep learning methods in computer vision are achieving the state-of-the-art in recent years. As end-to-end solutions, CNNs also simplify the procedures of ASI. For example, a fully convolutional network (FCN) can segment surface defects by supervised learning [16], without any extra pre-process or post-process.

A notorious limitation of CNNs is computation. In many cases, expensive GPUs with high computing capabilities are necessary to support large workloads. However, in considering manufacturing cost, most core computing resources of ASI platforms are mainly the low-power CPUs of industrial personal computers (IPC), or even FPGAs (field programmable gate array). To take advantage of CNNs in ASI while continuing to use these low computing capability processor units, converting into compact CNNs is a workable option. Meanwhile, a CNN with high performance needs large datasets to train, while there exist few online datasets for surface defect images.

To address these limitations, this work introduces our novel lightweight CNN, the compact design remarkably reduces the number of weights and computation while still achieving a high performance. We also investigate the availability of a generic solution for ASI. That is, we apply our compact model to multiple types of surfaces. Following proper training strategies, our model has satisfactory results for different materials with small training sets. It is worth pointing out that the defect pixels only take a small proportion of the whole image, therefore, the backpropagation gradient of defects will be overwhelmed by the background gradient, and subsequently cause low accuracy. Moreover, the images with defects are usually far less than the non-defect, the networks do not have sufficient data to learn the defect features, resulting in accuracy significantly decreases. To tackle this problem, we apply the gradual training strategy where the defective regions are intentionally enlarged with specific scales. Our main contributions are:

- Propose an application-oriented deep neural network for cross-product ASI. Our model obtains the state-of-the-art in both classification and segmentation.
- Modify the residual bottleneck with small kernels and depthwise convolution, build a lightweight backbone and decoder. Our model significantly reduces the number of weights, computation and time cost.
- Obtain $\sim$5% true positive rate improvement on tiny defect detection, by training our model with gradual training and a fine-tuned strategy.

2. Related Work

2.1. Compact Design

Intuitively, increasing parameters, computation or the convolution layer may improve the accuracy of CNN models. Table 1 reveals the performance result of the well-known models trained on ImageNet (ILSVRC-2012-CLS) [17]. In the table, W is the number of weights (M, or $\times 10^6$); F is the FLOPs ($\times 10^6$) evaluated with TensorFlow [18], which accumulates the computation including convolution, pooling, regularization, batch normalization, and all other operations defined in the graph. The statistics seemingly indicate that a model with more FLOPs and weights is more likely to obtain higher accuracy. However, we observe some cases against this assumption—(1) Inception-V4

[19] still slightly outperforms ResNetV2-200 although it only has the 1/2 FLOPs and the 2/3 weights of ResNetV2-200 [20]. (2) MobileNetV2@1.4 has much fewer weights than VGG-16 while it achieves a higher accuracy by 4%.

Interestingly, many compact deep neural network (DNN) models can still achieve an excellent performance on image classification, such as SquezeNet [21], Compact DNN [22], MobileNetV1 [23], ShuffleNet [24], and MobileNetV2 [25]. Some popular applications of object recognition include gesture recognition [26], handwritten Chinese text recognition [27] and traffic sign classification [28]. Now, the demand for compact models on these applications prompts an increase. Compact models use specific convolution methods to reduce redundant weights, for example, 1×1 convolution, group convolution, and depthwise separable convolution. Compact models are also used in surface defect inspection, for instance, Reference [29] introduces a compact CNN to perform defect segmentation and classification simultaneously.

Table 1. The top-1 accuracy of the models on the ImagNet. F refer to FLoating-point OPerations (FLOPs) and W to Weights. **bolb** means the best performance.

Model	**Layers**	F**(M)**	W**(M)**	**Top-1** $Acc(\%)$
AlexNet	8	1386	49.43	57.2
VGG-16	16	9040	138.35	71.5
ResNetV2	50	7105	25.57	75.6
ResNetV2	101	14631	44.58	77.0
ResNetV2	200	29165	64.73	79.9
Inception-V4	-	12479	42.65	**80.2**
MobileNetV2@1.4	53	**1189**	**6.11**	75.0

2.2. *Transfer Learning*

Transferring the knowledge from a large scale source dataset to a small one has proved to be a meaningful practice. Reference [30] tried to investigate the best approach for transferring knowledge. In their experiment, 13 ImageNet classifiers were trained via three different methods—1) random initialization; 2) fine-tuned from ImageNet-pretrained weights; 3) freeze CNN as fixed feature extractors. Then these classifiers were assigned to 12 small datasets. The study shows that fine-tuned models significantly outperform others. At the same time, those models that achieve higher accuracy on ImageNet also exceed others on different datasets. A considerable large source dataset is a key factor of using transfer learning to improve the target dataset. In DeeplabV3+ [31], the source dataset for model pre-training is JTF-300M [32], which is 300 times larger than ImageNet. Using this super large dataset, researchers pretrain a 65-layer Xception [33] model, then fine-tune a segmentation model based on PASCAL VOC 2012. The result suggests these strategies have remarkably leveraged performance, their model won the best-recorded mIOU (Mean Intersection over Union). In the practice of surface inspection [15], a fine-tuned ImageNet pretrained VGG-16 is used as a patch classifier on surface defect inspection. The experiment shows that fine-tuning all layers is the best valid way for models to achieve high accuracy. Training neural networks with the fine-tune strategy can gain much better results than with random initialization. Based on these studies, we opt to utilize the fine-tune strategy.

2.3. *Model Compression*

Deep neural networks are computationally expensive and memory expensive, while hardware resources are very limited in reality. For example, in mobile applications, the application size and graphics computing resources are constrained. To address this issue, model compression has been introduced to help models easier to deploy embedded systems by reducing redundant parameters and computation [34]. In early work, pruning [35] reduces the number of redundant connections; quantization [36,37] compresses the floating-point operations by quantifying 32-bit floating-point parameters to 16-bit, 8-bit or even 1 bit fixed-point; Low-rank decomposition [38] removes the redundancy in the convolution kernels through computing the low-rank tensor decomposition; Knowledge distillation [39] trains small student network with the knowledge of a large teacher network. These model compression methods can significantly reduce parameter size, computation cost and running time. More practical approaches and suggestions can be seen in Reference [40].

3. Architecture

The network architecture of our lightweight (LW) CNN consists of a LW bottleneck, classifier network, and segmentation decoder.

3.1. Depthwise Convolution

We call the regular convolution in deep learning as the standard convolution. Figure 1a shows the basic operations of standard convolution. In a standard convolution layer with C_{in} input channels and C_{out} output channels, each output feature map is the sum of the C_{in} input feature maps convoluted by C_{in} corresponding kernels. Many famous models such as Alexnet [41], VGG [42], and ResNet [20,43] use the standard convolution. The number of weights of standard convolution is:

$$W_{\mathrm{std}} = C_{in} \times k_W \times k_H \times C_{out}, \tag{1}$$

where $k_W \times k_H$ is the kernel size. To generate output feature maps with size $f_W \times f_H$, the computational cost is:

$$MACs_{\mathrm{std}} = C_{in} \times k_W \times k_H \times C_{out} \times f_W \times f_H, \tag{2}$$

where $k_W \times k_H$ is the kernel size and $f_W \times f_H$ is the output feature map size. MACs is Multiply-ACcumulate operations. The value of FLOPs evaluated by TensorFlow is roughly twice larger than of MACs.

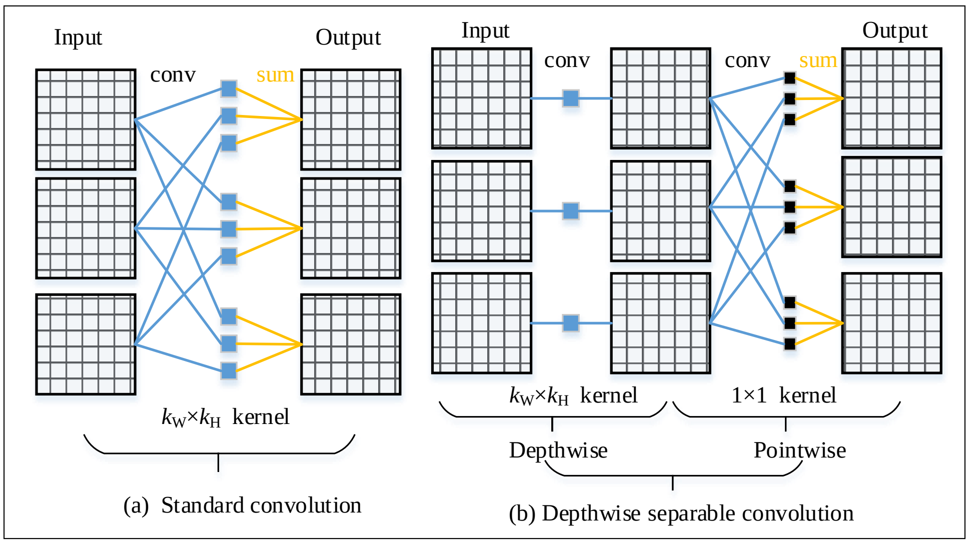

Figure 1. Standard convolution and depthwise convolution operations. A depthwise separable convolution is combined by a depthwise convolution and a 1×1 convolution (pointwise conv).

Figure 1b displays the basic operations of depthwise convolution and depthwise separable convolution. Distinct from standard convolution, each output feature map of depthwise convolution is only the result of an input feature map convoluted by a single convolution kernel. The number of weights of a depthwise convolution is:

$$W_{\mathrm{dw}} = k_W \times k_H \times C_{out}. \tag{3}$$

The computational cost of a depthwise convolution layer is:

$$MACs_{\mathrm{dw}} = k_W \times k_H \times C_{out} \times f_W \times f_H. \tag{4}$$

Depthwise convolution is the key component of our backbone. It reduces the weights and computational cost by C_{in} times. Depthwise separable convolution is the operation of depthwise convolution followed by a 1×1 convolution (pointwise convolution). In modern networks, depthwise separable convolution has been widely applied and achieved excellent performance, such as Xception [33] and MobileNets [23,25]. We also use the depthwise separable convolution in our decoder.

3.2. LW Bottleneck

The deep bottleneck architecture of ResNet is introduced in Reference [20,43]. Figure 2a shows the structure of ResNet bottleneck. It substantially increases the feature map number, while decreasing the number of weights and the computational cost. Layer dimensionality impacts the manifold of interest in neural networks. In ResNet, the number of channels experiences a series of changes. Firstly, the network has high-dimensional inputs. Afterward, the number of channels significantly shrinks in the first pointwise convolution operation, then stays the same in the following standard convolution layer, and finally expands back to the input size in the next pointwise convolution. This structural design effectively reduces the computational cost of 3×3 standard convolution. Figure 2b shows the basic structure of MobileNetV2 invert residual bottleneck. On the contrary, the invert residual bottleneck has low-dimensional inputs, and the number of channels firstly expands then shrinks. The depthwise convolution is much lighter than standard convolution, making the thicker 3×3 convolution possible, and this structure proved to be more efficient in running memory and slightly increased the model accuracy.

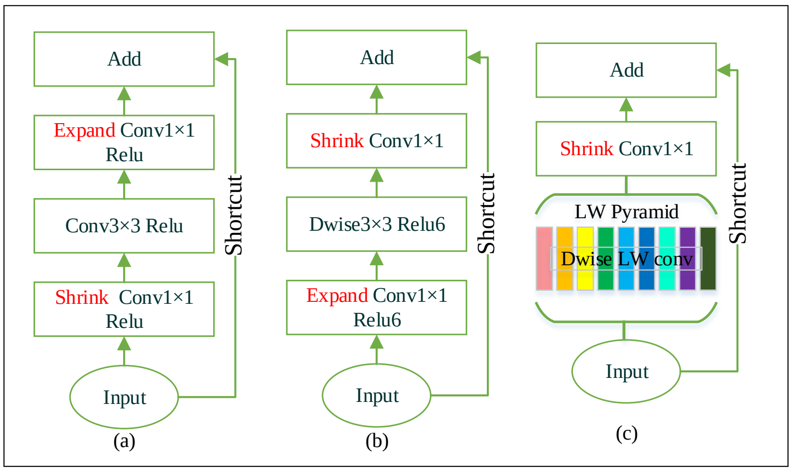

Figure 2. Our bottleneck structure (**c**) Vs ResNet (**a**) and MobileNetV2 (**b**). LW(LightWeight) Dwise(Depth-wise) convolution collecting rich features in difference size of receptive fields, building a LW pyramid. Different color in (**c**) means different convolution kernel size.

Despite the depthwise convolution shrinking a remarkable number of weights, we attempt to reshape kernels for more performance improvement. In the LW bottleneck, the first layer has multiple parallel depthwise convolutions that will generate feature maps in different receptive fields, which is also equivalent to expanding channels to the next layer. Previous research has shown that the 5×1 and 5×1 convolutions can provide wider receptive fields and reduce weights with lager kernel techniques [44]. Based on the studies, some 3×3 kernels will be replaced with $1 \times k$ and $k \times 1$ to avoid redundant computation, and k will include 5. Specifically, the types of kernels are 5×1, 1×5, 3×1, 1×3, 2×1, 1×2, 2×2, and 3×3. Each kernel forms a depthwise convolution path, then the feature maps of the paths are concatenated. Afterward, a pointwise convolution is used to shrink the channels. The multiscale kernels build a pyramid with different receptive field sizes, which is structurally similar to the ASPP [31,45]. In a LW bottleneck, the number of weights is:

$$\begin{aligned} W_{\mathrm{LW}} =& [5+5+3+3+2+2+2\times 2+3\times 3] \\ & \times C_{in} + 8\times C_{in}\times 1\times 1\times C_{out} \\ =& (33+8\times C_{out})\times C_{in} \end{aligned} \tag{5}$$

and a MobileNetV2 bottleneck with a expansion factor 6 is:

$$\begin{aligned} W_{\mathrm{MB}} =& C_{in}\times 1\times 1\times 6\times C_{in} + 6\times C_{in}\times 3\times 3 \\ & + 6\times C_{in}\times 1\times 1\times C_{out} \\ =& (54+6\times C_{in}+6\times C_{out})\times C_{in}. \end{aligned} \tag{6}$$

In our network design, we hold the condition $C_{out} \leq 2\times C_{in}$, thus $W_{\mathrm{LW}} < W_{\mathrm{MB}}$.

Relu activation function is widely used for non-linear transformation and training acceleration [41]. Relu6 [46] is more robust when combining with low-precision computation. However, non-linearity will lead to significant information loss in the case of low-dimensional input [25]. Based on these studies, we only use Relu6 after the depthwise convolution but drop non-linear transformation after the pointwise convolution. We use stride rather than pooling in the depthwise convolution to enlarge the receptive field and reduce the feature dimension, since it is proved that fewer computational cost while no impact on the accuracy in using stride than pooling [47]. Residual shortcuts will be applied to prevent vanishing gradient once the input and output of the bottleneck have the same tensor shapes.

3.3. Backbone and Decoder

Table 2 describes our backbone implementation structure in detail. A block is the basic convolution unit, and it can either be a standard convolution or a bottleneck. In the table,*N* represents that the blocks are repeated by *N* times; *S* is the stride. *S* is used in the first depthwise convolution when the bottleneck blocks are stacked repetitively. Compared with MobileNetV2, our backbone is more compact in terms of the number of layers and blocks. For example, the MobileNetV2 has 53 convolution layers and 19 blocks, while our model has 21 and 12. Another improvement is on channel design, which leads to $\sim$1 M of weights reduction without hurting model performance. In our design, the output channels of the last layer are shrunk into 320, which is sufficient enough for surface defect inspection. We also use the width multiplier parameter for computation and accuracy tradeoff. Width multiplier is a tunable hyperparameter that is introduced by MobileNet and MobileNetV2 [25], when a multiplier is smaller than 1, the very last layer of the backbone will not be changed to maintain the accuracy performance. Width multiplier is used as a tunable hyperparameter for accuracy/performance tradeoffs, which can change network channels uniformly at each layer as MobileNets do.

Table 2. The backbone of light weight (LW) network with a $331^2\times 3$ input.

Input	Operator	C_{out}	*S*	*N*	Block
$331^2\times 3$	3×3 Conv2d	32	2	1	1
$166^2\times 32$	1×1 Conv2d	16	1	1	2
$166^2\times 16$	LW bottleneck	24	2	2	3–4
$83^2\times 24$	LW bottleneck	32	2	2	5–6
$42^2\times 32$	LW bottleneck	32	2	1	7
$21^2\times 32$	LW bottleneck	64	2	4	8–11
$11^2\times 64$	1×1 Conv2d	320	1	1	12

After the backbone, the operation procedure for classification and segmentation are different. In classification, the backbone is followed by a global average pooling layer, then a fully connected layer. For the segmentation task, a decoder is embedded, which is shown in Figure 3. The decoder consists of an Atrous Spatial Pyramid Pooling (ASPP) [31] and three pointwise convolution layers. In this decoder, the number of feature maps is fixed, and the output is the result of segmentation. The number of output channels equals the number of semantics, so we set channel number as 2—one for the foreground and another for the background. Specifically, the ASPP is composed of one pointwise convolution and three depthwise separable convolution layers. The kernels in depthwise separable convolution have the same size3×3 , but their atrous rates are different, which are 6, 12, and 18. The shortcut is from the 4th or the 5th block of the backbone, which corresponds to $1/4$ and $1/8$ of output size respectively.

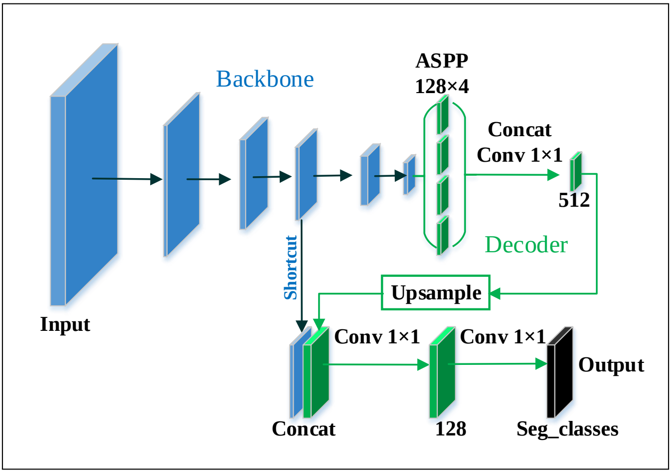

Figure 3. The proposed segmentation model. The decoder includes an atrous spatial pyramid pooling (ASPP), a bilinear upsampling, a series of concats and pointwise convolutions.

3.4. Implementation Details

3.4.1. Hyperparameter Settings

The network is implemented with TensorFlow. In both classification and segmentation tasks, we use softmax cross-entropy as the loss function and Adam optimizer as the adaptive gradient descent optimizer. The learning rate is 0.0001, which will exponentially decay 10% at every 10,000 steps. To speed up training, batch normalization layers are attached to each convolution layer, with decay = 0.9997 and epsilon = 0.00001. To avoid overfitting, we use L2 regularization in each convolution layer. The decay of L2 regularization is 0.0001.

3.4.2. Gradual Training

Training deep neural networks on the surface defect datasets may have the issue of data imbalance—first, the image data are difficult to collect, because there only exists a small amount of the products with surface defect; second, the target features is difficult to detect, because the number of surface defect pixels are much less than background pixels. These characteristics can cause the background overfitting and the foreground underfitting. In the worst case, the neural network may predict the defective product as a "perfect product" because all the surface defect pixels may be "ignored" by the model. Besides, the size of different types of defects varies greatly, the gradient of tiny defects will be overwhelmed by the gradient of large defects or background gradient, causing the low accuracy of tiny defects.

We propose a gradual training strategy to address data imbalance. In the early stage of network training, we only train the images of defective products and zoom into the defect areas. The model is trained with constant input size, while the zoom ratio is linearly reduced to 1. Later, the images of perfect products are fed to the model. This strategy has proved to be a valid way to prevent the network from overfitting. It also improves model accuracy. Details of this training strategy will be expounded in Section 4.4.

4. Experiments

4.1. Datasets

4.1.1. The Textures dataset

We collect a dataset of surface texture (8674 images), which contains 64 classes from 3 public datasets. It can be accessed by https://github.com/abin24/Textures-Dataset. The images are uniformly resized into 331 × 331. Notably, 11 classes (3194 images) are from KTH surface datast (including KTH-TIPS and KTH-TIPS2) [48], 28 classes (4480 images) from Kyberge [49] dataset, and 25 classes (1000 images) from the UIUC dataset [50].

Images in the Textures datasets include wood, blanket, cloth, leather, and so forth, these textures are very commonly seen in surface inspection problems. So, we believe that it can be used as a source dataset for network pretraining, and also can be used to evaluate the capacity of defect inspection models. Samples of this dataset are displayed in Figure 4.

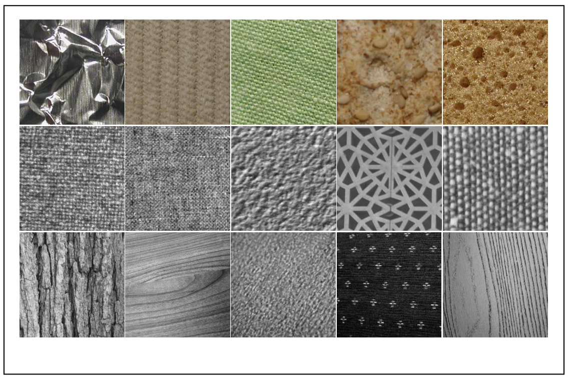

Figure 4. The textures dataset. Each image is a sample of one texture class, first row is from the KTH, second row from Kyberge, and third from UIUC.

NEU surface defect dataset [1] is a hot-rolled steel strip surface defect dataset including 6 classes (1800 images) of defects. Images are grayscale with uniform size 200 × 200. The steel defects include crazing (Cr), inclusion (In), patches (Pa), pitted surface (PS), rolled-in scale (RS), and scratches (SC), as shown in Figure 5. Images are captured in different exposure conditions, their intra-class defects exist large differences in appearance while the inter-class defects share some similar aspects, making the classification task challenging.

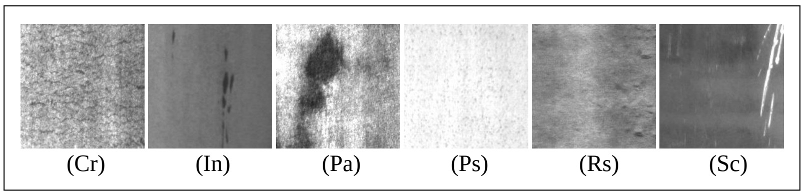

Figure 5. The 6 classes surface defects in NEU dataset.

4.1.2. The DAGM dataset

The DAGM 2007 dataset [51] has 6 classes of textured surfaces, it is evenly divided into training and test sets. As shown in Figure 6, defects are coarsely labeled with elliptical masks. In each class of the 6 surfaces, there are 575 images in the training set, about 80 with a defect and the rest are normal, and the test set shares this distribution.

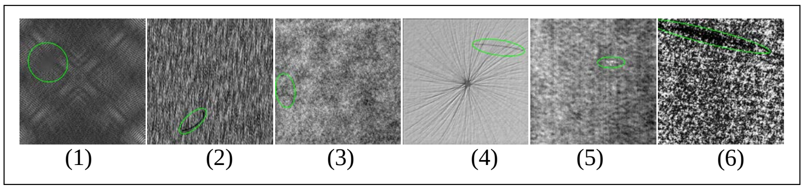

Figure 6. The DAGM dataset, the green ellipses are the provided coarse ground truths.(1)-(6) are the 6 classes of the surface in the dataset.

Wood defect dataset [5] is a surface defect dataset of dried lumber boards, containing 18 types of defects (839 images with size 488 × 512). Figure 7 shows defects and the corresponding ground truths with a set of bounding boxes. Some defects only appear in 2 or 3 images, such a severe imbalance of image distribution brings many difficulties to our work.

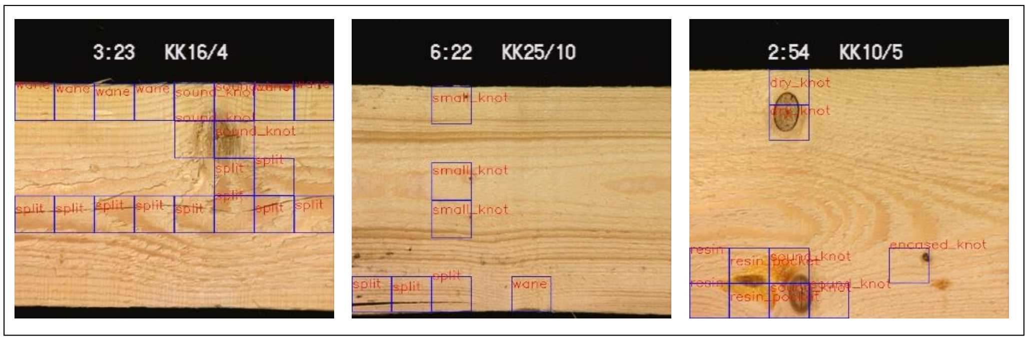

Figure 7. Samples in wood defect dataset, blue bounding boxes are the provided ground-truth, notice that the bounding box are not mapped well with the defects.

4.2. Classification on Textures

4.2.1. Experiment Settings

We split the Textures dataset into two halves randomly, one for the training set, and another for the validation. We implement strong data augmentation to balance the image number of each class and avoid the effect of overfitting. In the training set, we generated 200 images for each class. Each augmented image is generated from a random combination of multiple data augmentation methods provided by imgaug library [52]. These data augmentation approaches include flipping, affine translation, crop, blur, Gaussian noise, illumination perturb, elastic transform, perspective transform, and piece-wise affine. The augmented images are mixed with the original images to build our final training set.

We compare our models with ResNetV2-50 [20], ShuffleNet [24], and MobileNetV2. The hyperparameter setting is the same as those described in Section 3.4. The input size is 331 × 331 × 3, and training for 90,000 steps with a batchsize of 64. Table 3 shows the experiment result, where @ represents Width multiplier which is mentioned at Section 3; t is the running time of a forward operation on an IPC that is equipped with an Intel i34010U CPU(1.7 GHz) and 4 GB memory; Acc is the test accuracy, defined as:

$$Acc = \frac{C}{N} \times 100\%, \qquad (7)$$

where C is the number of correct predictions in the test set and N is the number of all predictions in the test set.

Table 3. Model performance on the Textures dataset. F refer to FLOPs, W to Weights, t to running-time, and Acc to Accuracy.

Model	**Pretrain**	F(M)	W(M)	t(ms)	Acc(%)
ShuffleNet@0.25	-	196	0.19	254	97.02
MobileNetV2@0.35	-	267	0.48	259	97.72
MobileNetV2@1.4	-	2684	4.43	695	97.56
ResNetV2-50	-	16082	23.65	893	**99.03**
ours@0.35	-	**76**	**0.06**	**75**	97.51
ours@0.75	-	246	0.14	140	98.71
ShuffleNet@0.25	ImageNet	196	0.19	254	99.33
MobileNetV2@0.35	ImageNet	267	0.48	259	99.63
MobileNetV2@1.4	ImageNet	2684	4.43	695	**99.93**
ResNetV2-50	ImageNet	16082	23.65	893	99.60
ours@0.35	ImageNet	**76**	**0.06**	**75**	99.29
ours@0.75	ImageNet	246	0.14	140	99.52
ShuffleNet@0.25	Flower102	196	0.19	254	99.27
MobileNetV2@0.35	Flower102	267	0.48	259	99.50
MobileNetV2@1.4	Flower102	2684	4.43	695	**99.73**
ResNetV2-50	Flower102	16082	23.65	893	99.58
ours@0.35	Flower102	**76**	**0.06**	**75**	98.48
ours@0.75	Flower102	246	0.14	140	99.33

4.2.2. Impact of Parameter Size

A model with more weights is more likely to obtain higher accuracy. For example, MobileNetV2@1.4 performs better than MobileNetV2@0.35; ours@0.75 performs better than ours@0.35. However, ours@0.75 obtains higher accuracy than MobileNetV2@1.4 with fewer weights when the models are trained from scratch (random initialization).

In the experiments, the fine-tuned MobileNetV2@1.4 obtains the highest accuracy. Regardless of the good accuracy performance of the MobileNetV2@1.4 and ResNetV2-50, these models contain a large number of parameters (and operations) which require GPUs. However, an ideal ASI model should not only perform well inaccuracy but also takes less computational resources, because many ASI platforms usually are not equipped with such expensive hardware devices. After exploring the trade-off between accuracy and efficiency, we give up on embedding these large-parameter models.

4.2.3. Impact of Fine-Tune

All the backbone networks in this paper are either trained from scratch (random initialization) or fine-tuned from pretained parameters. As shown in Table 3, we pretrain our model on Oxford Flower102 [53] and ImageNet. We obtain higher accuracy improvement using fine-tune strategy than simply stacking more weights or layers. Compared with the random initialized ResNetV2-50 and MobileNetV2@1.4, ours@0.75 model (ImageNet pretrained) has much fewer weights and FLOPs, but higher accuracy. The fine-tune strategy can enhance accuracy by 1$\sim$2 %.

Fine-tune can also accelerate convergence. As shown in Figure 8, the Cross-Entropy losses of pretrained models decrease much faster than the random initialization ones. The ImageNet pretrained MobileNetV2@1.4 obtains the highest validation accuracy (99.93%) with only 4000 training steps, which is 22.5 times faster than the random initialization. Our experiment also shows that transferring the features of the larger scale source dataset is more generalization. As shown in Figure 8, ImageNet pretrained model has higher validation accuracy than Flower102 pretrained model at the same training loss. The ImageNet pretrained model has also higher accuracy improvement as seen in Table 3. However, it takes days to weeks to pretrain a designed model on ImageNet with a high-performance computer equipped with exceptionally strong GPUs and over 130 GB disk storage. Since the accuracy improvement of Flower102 pretraining is competitive to ImageNet pretraining, a much smaller source dataset can be used as a compromise when computing resources are limited.

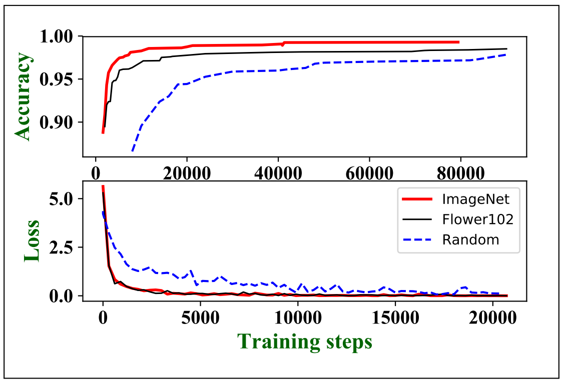

Figure 8. The accuracy and loss of different parameters initialization methods (our@0.35).

Our model (ours@0.75) is ∼4.9 times faster than MobileNetV2@1.4, with only ∼1/10 weights, ∼1/4 FLOPs and a slight (∼ 0.41%) accuracy decrease. Compared to the models with similar FLOPs size, our models also have competitive accuracy to ShuffleNet@0.25 and MobileNet@0.35, but much faster speed on low power CPU.

4.3. Classification on NEU

Each class of NEU dataset has 300 200 × 200 images, they are randomly split into 150 training images and 150 test images. We use accuracy defined in Equation (7) as evaluation metric, and the classification result can be seen in Table 4.

SCN + SVM [1] is the first model using the NEU dataset. In this model, a translation and scale invariant wavelet method called scattering convolution network (SCN) is used to extract texture features; support vector machine (SVM) is used as the classifier. Their experiments proved that SCN performs better than other methods on extracting features, and SVM performs better than the Nearest Neighbor Classifier on this dataset. This method achieves 98.60% accuracy. Reference [2] introduces an end-to-end CNN for this surface defect classification task. Their work proves that data augmentation on this dataset is crucial for performance improvement, their model gets a better accuracy at 99.05%. Ruoxu, Ren, et al. [54] used an ImageNet pretrained CNN named Decaf [55] as a fixed peculiarity extractor, they remove the fc7, fc8 layer, freeze the layers before fc6, then trained a multinomial logistic regression (MLR) classifier. This transfer learning method helps them reach a higher accuracy at 99.27%. Larger-size models can fit the dataset better, mentioned in Reference [56], Wide Residual Networks (WRN) [57] are used, and the WRN-28-20 gets the best-reported result at 99.89%. However, as shown in the Table 4, WRN has so many weights to train that results in a massive computation cost during training, as long as 2 days as their report, even if the input images were resized to 32 × 32.

Table 4. The accuracy of the NEU dataset.

Model	**Pretrain**	F (M)	W(M)	t(ms)	Acc(%)
SCN + SVM	-	-	-	-	98.60
CNN(Yi Li)	-	**400**	**0.46**	-	99.05
Decaf + MLR	ImageNet	1078	28.58	-	99.27
WRN	-	115557	144.8	-	**99.89**
ShuffleNet@0.25	-	52	0.19	126	99.89
MobileNetV2@0.35	-	102	0.40	116	99.89
ours@0.35	-	**28**	**0.04**	**29**	**100**
ShuffleNet@0.25	ImageNet	52	0.19	126	99.89
MobileNetV2@0.35	ImageNet	102	0.40	116	**100**
ours@0.35	ImageNet	**28**	**0.04**	**29**	**100**

We apply strong data augmentation to generate sufficient data for network training. Each class contains 9000 images after data augmentation, all the augmentation methods mentioned in Section 4.2.1

is used in this case. We train ShuffleNet@0.25, MobileNetV2@0.35 and ours@0.35 at this augmented dataset with 200 × 200 input size and a batch size of 64, losses and hyperparameters are the same as those described in Section 3.4.

Both MobileNetV2@0.35 and the ours@0.35 achieve exceptionally high performance with the best record of 100% test accuracy. However, compared to the minor improvement on accuracy, it is more worth pointing out that the pretrained models accelerate convergence. For instance, the ImageNet pretrained MobileNetV2@0.35 and ours@0.35 takes only 1.2K-1.6K training steps to converge (99% + test accuracy). In the setting of random initialization, the training time of ShuffleNet@0.25, MobileNetV2@0.35 and the ours@0.35 is almost identical, which takes around 4 hours (60K steps) on a GTX 1080 GPU, much faster than the 48 hours of WRN-28-10. Our model has the least parameter, computational cost and running time but the highest accuracy in the NEU dataset.

4.4. Segmentation on DAGM

4.4.1. Comparison Models

We use the $mAcc$ (mean Accuracy) in Reference [14] as our evaluation metric, which is defined as:

$$mAcc = \frac{TPR + TNR}{2} \tag{8}$$

and TPR (True Positive Rate), TNR (True Negative Rate) are defined as:

$$TPR = \frac{TP}{TP + FN}, \qquad TNR = \frac{TN}{TN + FP}. \tag{9}$$

$mAcc$ is an image-level evaluation metric, without considering pixel-level segmentation accuracy. Specially, the $mAcc$ in Table 5 is the mean $mAcc$ of all the 6 type textures of the test set.

Table 5. The mean accuracy of the DAGM dataset.

Model	**Pretrain**	***F*(M)**	***W*(M)**	***t*(ms)**	***mAcc*(%)**
FCN+CNN	-	-	-	-	95.99
CA-CNN	-	-	-	-	96.44
CNN + SW	-	-	-	-	99.20
C-CNN	-	**2280**	**1.27**	-	99.43
VGG	ImageNet	9024	134.31	-	**99.93**
ShuffleNet@0.25+ours decoder	-	1126	0.64	371	98.99
MobileNetV2@0.35+ours decoder	-	1630	1.40	591	98.99
ours@0.35	-	**923**	**0.55**	**217**	98.86
ours@0.75	-	1312	0.60	369	**99.46**
ShuffleNet@0.25+ours decoder	ImageNet	1126	0.64	371	99.40
MobileNetV2@0.35+ours decoder	ImageNet	1630	1.40	591	**99.90**
ours@0.35	Textures	**923**	**0.55**	**217**	99.40
ours@0.35	ImageNet	**923**	**0.55**	**217**	99.43
ours@0.75	Textures	1312	0.60	369	**99.79**
ours@0.75	ImageNet	1312	0.60	369	99.67

In Reference [13], 32 × 32 image patches and their 64 × 64 context patches are randomly extracted from the training images, and fed to a context-aware CNN(CA-CNN) classifier together. All test images are classified into 12 classes(6 background textures and 6 defects) with the 32 × 32 patches. Context information is critical to increasing the $mAcc$ this sliding window classifier. In Reference [14], the problem of segmentation is also treated the defect segmentation as a classification task in 32 × 32 patches, with CNNs and sliding windows (CNN + SW). There also exist some attempts to select an optimal CNN structure among different depths and widths between operation time and accuracy trade-off. In Reference [58], the segmentation strategy includes two stages: first, an FCN is used to segment defects; second, the 32 × 32 or 64 × 64 patches are cropped around the defect detected by the first stage and a CNN is used as a foreground-background classifier to reduce false alarms. These three approaches use very small patches. However, the elliptical ground contains tremendous non-defective areas (as shown in Figure 6(2) and (4)). To extract positive samples, more accurate manual re-label ground truths are necessary. The ever-changing texture background makes a universal set of background patches to train the network, in Reference [14] 324,000 training patches are cropped to ensure model accuracy. So these sliding window based models are unstable and hard to train.

In Reference [15] a input image is cropped to 9 224 $\times$ 224 sub-images, and an ImageNet-pretrained VGG-16 [42] is fine-tuned. The whole image is classified by the vote result of 9 sub-images. Rich features learned from ImageNet push the VGG-16 well-fit to the DAGM and get the best $mAcc$(99.93%) overall. However, the parameters of a VGG-16 are 134.31 M, which causes very large computational cost — 9024M FLOPs in one sub-image, 9 times FLOPs are needed in an entire input image.

In Reference [29], an end-to-end segmentation-classification multi-task compact CNN(C-CNN) is used. Different from the ordinary segmentation model, they use a mean squared error (MSE) loss to treat the defect segmentation as a regression problem and use heat maps as segmentation results. The training process has two stages: first, only the regression FCN is trained; second, the layers related to FCN are frozen and the classifier is trained only. This training strategy enables the model to fit a good classifier without affecting the segmentation accuracy. The $mAcc$ of this model is 99.43% with small computational cost and very few parameters.

4.4.2. Proposed Model

The DAGM dataset contains 6 classes of defects. Noticeably, the full input image size is 512 $\times$ 512, but the width of the smallest defects only consists of 3-4 pixels. We treat this defect inspection as a binary image segmentation task and the elliptical mask as the supervise ground truth. We build a segmentation neural network with the encoder-decoder illustrated in Figure 3, in which input size = 512 $\times$ 512, output stride = 8, and output size = 64 $\times$ 64. The encoder backbone is our LW backbone, ShuffleNet or the MobileNetV2, and the decoder shortcut is the 5th layer output of the backbone. The training batch size is set to 12, losses and hyperparameters are the same as those described in Section 3.4. The outputs of these semantic segmentation model are two 64 $\times$ 64 binary images, one for the foreground (defect) and another for the background. When we evaluate the $mAcc$ in the test set, if any white pixel appears in the predicted foreground, the image will be treated as a positive sample, otherwise, it will be predicted as normal (negative). The training set is augmented by methods described in Section 4.2.1, since some spatial augmentation (affine/flip/crop/etc.) will change the locations of the defects, the same spatial augmentation transform will be applied to the ground truth masks as well. Each type of these texture surface will be augmented to 6000 images.

The proportion of training images with defects and non-defects is 1/7, and some tiny defects only consist of 3-4 pixels width while the image size is 512 $\times$ 512. Such severe imbalance is highly likely to cause low accuracy of tiny defects because of foreground under-fitting. We use the gradual training strategy mentioned above (Section 3.4), to address the issue of the severe imbalance distribution of DAGM. First, only the images with defects are trained at the beginning, and these images are zoomed to the defect area. Figure 9a presents an example of the process. The zoom ratio is linearly reduced from 3.6 to 1 at the first 20K steps, then the model continues to train with the original input size. The images without defects are fed to the model after 80K steps. These setups are empirically designed, and our experiments prove that this training strategy can help obtain better performance on surface defect inspection.

As shown in Table 6, gradual training improves TPR_2 of the tiny defect (the True Positive Rate of the second type texture surface), but cannot improve the $mAcc$. This conservative training strategy enables the network to learn the feature of tiny defects, to avoid foreground underfitting and result in predict all images as the background. Together with ImageNet pretraining, 4.77% TPR_2 and 0.61% $mAcc$ improvement can be achieved. Pretraining is the key process accuracy improvement, pretraining on small datasets like Flower102 or Textures is competitive to large scale dataset(ImageNet).

Table 6. Impact of the gradual training and pretraining on tiny defect.

Model	Gradual Training	Pretrain	$TPR_2(\%)$	$mAcc(\%)$
ours@0.35			90.47	98.82
ours@0.35	✓		94.05	98.86
ours@0.35	✓	✓	**95.24**	**99.43**

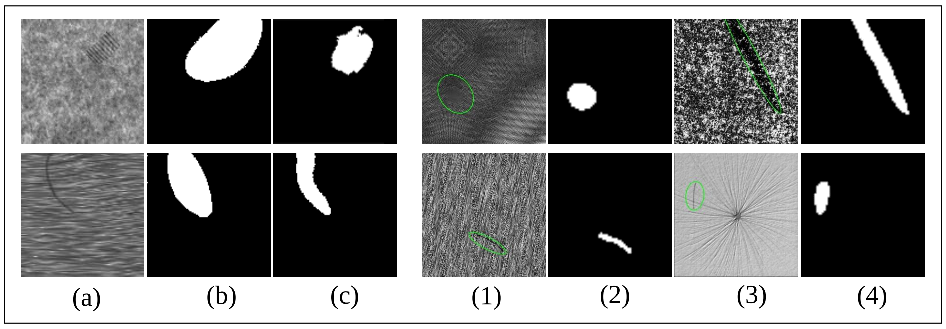

Figure 9. Results of DAGM dataset. (**a**): At the early training steps, training images are zoomed to the defect, (**b**) are the label of (**a**), (**c**) are the predicted images of the model. (1), (3) are the full-scale image, (2), (4) are the prediction of the well-trained model, predict masks are more accurate than the ground truths sometimes.

Table 5 shows the statistic of model performance on DAGM dataset. Ours@0.35 model costs the least computation with the least weights, while still achieves a $mAcc$ of 99.43%. ImageNet pretrained MobileNetV2@0.35 with our LW decoder get a very high $mAcc$ (99.90%). In contrast to the FCN+CNN [58], CA-CNN [13], CNN [14] and other patch-based models, our model does not require finer ground truth. Universe set of background patches is also no longer needed, and the end-to-end prediction does not require any post-processing steps. As shown in Figure 9, our model makes a fine prediction of elliptical defect masks, and sometimes the outputs are more accurate than the ground truths. This phenomenon indicates that the model has learned the feature of defects rather than overfitting the training set.

Ours@0.35 is 3 times faster than the MobileNetV2@0.35 on a low-cost IPC, which make $\sim$5 fps surface defect inspection is possible, with only 0.47% $mAcc$ drop. Ours@0.35 has same $mAcc$ with the ShuffleNet@0.25 but 1.7 times faster on the IPC. When using a GPU, ours@0.35, ours@0.75, ShuffleNet@0.25+ours decoder and MobileNetV2@0.35+ours decoder can inference a 512×512 image within 8 ms.

4.5. Segmentation on Wood Defects

Each image in the Wood defect dataset has a set of bounding boxes of the same size, and a label indicating the defect type inside these bounding boxes (shown in Figures 7 and 10). These evenly distributed bounding boxes sometimes do not match the defects well. We use ours@0.75 as the backbone, which also pretrained from DAGM dataset, as the encoder. The output stride is 8, the input size is $488 \times 512 \times 3$, and the defect bounding boxes as the ground truth. Half of the images are used as the training set and the rest as the test set, each of the 18 type defect is augmented to at least 6000 images with the same method used in DAGM dataset, to reduce the effects of data imbalance. We also adopt all the hyperparameters and gradual training strategy the same as we used in the DAGM dataset.

We adopt the Accuracy and Error escape rate defined in Reference [54], the Accuracy is defined as:

$$A = 1 - \frac{(N_{\text{lab}} - N_{\text{det\_lab}}) + (N_{\text{det}} - N_{\text{det\_lab}})}{N} \tag{10}$$

And Error escape rate is defined as:

$$E = 1 - \frac{N_{\text{det\_lab}}}{N_{\text{lab}}} \tag{11}$$

where the N_{lab} is the number of defect patches (bounding boxes), N_{det} is the number of detected defect patches, $N_{\text{det\_lab}}$ is the correctly detected defect numbers, and N is the total number of patches. High accuracy (A) and low Error escape rate (E) is expected.

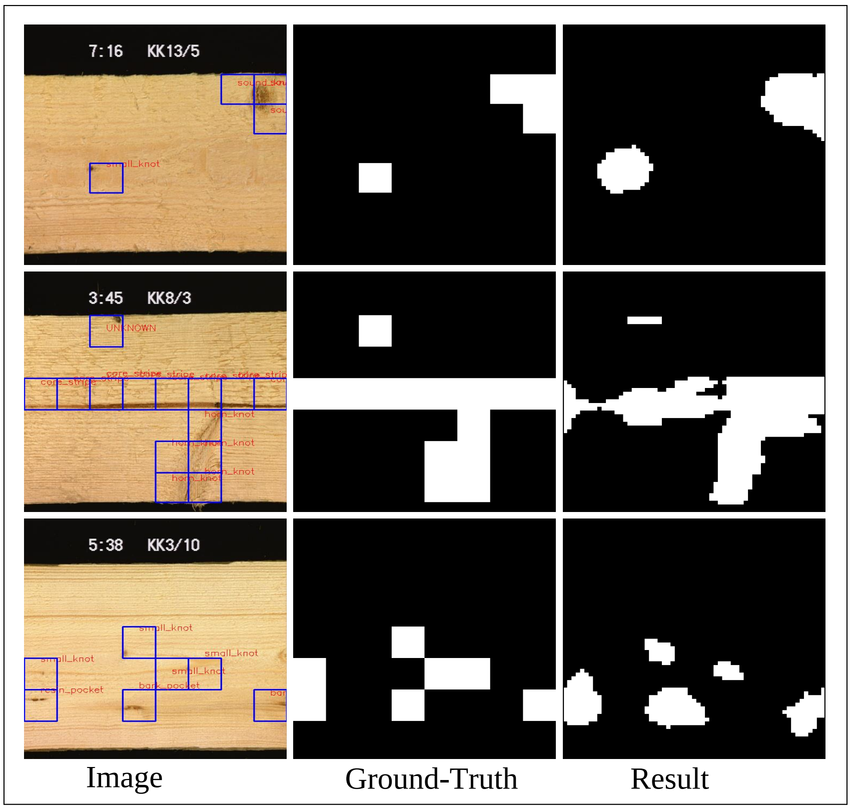

Figure 10. Segmentation results of Wood defect dataset.

In Reference [54], an ImageNet pretrained Decaf is used as the fixed feature extractor, and a multinomial logistic regression (MLR) classifier is trained with patches. Heatmaps are generated from the predict probabilities of the MLR, and Otsu's method together with Felzenswalb's segmentation is used to generate binary segmentation results. They found that patch size and patch stride will be a greatly affect the result, small patches will not include enough information, and small stride will lead to high computational cost, while large patch size or stride will lead to imprecise segmentation result. And in this dataset, they choose a patch size roughly the same as the size of the defect (50×50) and a stride of 10. They got the best-reported result in all the defect types, which has the highest A, and the lowest E.

We also treat this task as a binary segmentation, the segmentation result can be seen in Figure 10, our model fits the defect better than the provided bounding boxes. The A and E of our model and the best-reported result of Reference [54] is shown in Figure 11. Our model has much higher accuracy than the Decaf and similar Error escape rate. Our model is much faster than the reported segmentation time in Reference [54], they use a 24 core workstation and 2 mins to generate heatmaps per image, our model tests an image in 397 ms with an Intel i3-4010U(1.7 GHz) CPU or 27 ms with a GTX1080 GPU. Our model has no patch size selection process and no post-process, differ from the segmentation method in Reference [54].

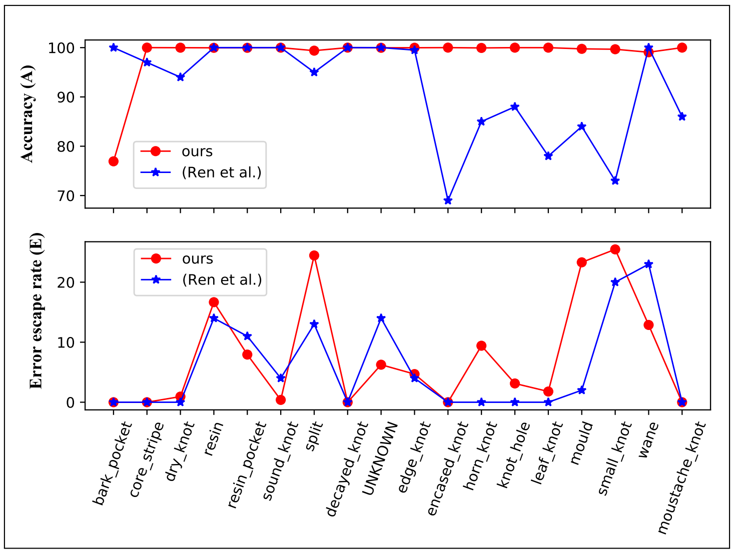

Figure 11. Accuracy and Error escape rate of Wood defect dataset.

5. Conclusions

In this paper, we introduce a novel compact CNN model with low computation and memory cost as well as explore the availability that if one model can be applied on cross-product surface defect inspection. We modify classic backbone architecture into the LW backbone via switching channel expansion order and customizing small kernels. Meanwhile, we observe that some strategies, e.g., transfer learning and zoom-in training, are critical to building the networks of tiny defect inspection. By evaluating the tradeoff between accuracy and computation cost, our model outperforms other advanced models with the least computation on four defect datasets. In contrast to the traditional approaches, our model gets rid of hand-crafted features and patch size selection. Compared with modern CNNs, the LW model has less than 1 M weights, which is no longer needed to run on expensive GPUs.

We record the experiment results on an IPC equipped with an Intel i3-4010U CPU (1.7 GHz). The inference time of a 200 $\times$ 200 image in the NEU dataset is $\sim$29 ms, a 331 $\times$ 331 image in the Textures dataset is $\sim$140 ms (@0.75, 99.33% Acc) and a 512 $\times$ 512 image in DAGM dataset takes 217 ms (@0.35, 99.43% mAcc). The inference time will be <30 ms if using a GPU. This lightweight, end-to-end compact model is promising to be used in low power consumption real-time surface inspection systems or smartphone apps.

Author Contributions: Conceptualization: Y.H., C.Q., and X.W.; methodology, formal analysis Y.H. and C. Q.; writing–original draft preparation, Y.H., C.Q.; writing–review and editing, S.W. and X.W.; supervision K.Y.; funding acquisition X.W. and K.Y. All authors have read and agreed to the published version of the manuscript 'A compact convolutional neural network for surface defect inspection'.

Funding: This study is supported by National Key R&D Program of China(Grant No.2018YFB1306500) and National Natural Science Foundation of China (Grant No.61421004).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Song, K.; Yan, Y. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Appl. Surf. Sci.* **2013**, *285*, 858–864.
2. Yi, L.; Li, G.; Jiang, M. An End-to-End Steel Strip Surface Defects Recognition System Based on Convolutional Neural Networks. *Steel Res. Int.* **2017**, *88*, 1600068.
3. Ye, R.; Chang, M.; Pan, C.S.; Chiang, C.A.; Gabayno, J.L. High-resolution optical inspection system for fast detection and classification of surface defects. *Int. J. Optomech.* **2018**, *12*, 1–10.

4. Cao, J.; Zhang, J.; Wen, Z.; Wang, N.; Liu, X. Fabric defect inspection using prior knowledge guided least squares regression. *Multimed. Tools Appl.* **2017**, *76*, 4141–4157.
5. Silvén, O.; Niskanen, M.; Kauppinen, H. Wood inspection with non-supervised clustering. *Mach. Vis. Appl.* **2003**, *13*, 275–285.
6. Chang, Z.; Cao, J.; Zhang, Y. A novel image segmentation approach for wood plate surface defect classification through convex optimization. *J. For. Res.* **2018**, *29*, 1789–1795.
7. Medina, R.; Gayubo, F.; González-Rodrigo, L.M.; Olmedo, D.; Gómez-García-Bermejo, J.; Zalama, E.; Perán, J.R. Automated visual classification of frequent defects in flat steel coils. *Int. J. Adv. Manuf. Technol.* **2011**, *57*, 1087–1097.
8. Chu, M.; Gong, R. Invariant Feature Extraction Method Based on Smoothed Local Binary Pattern for Strip Steel Surface Defect. *Trans. Iron Steel Inst. Jpn.* **2015**, *55*, 1956–1962.
9. Raheja, J.L.; Kumar, S.; Chaudhary, A. Fabric defect detection based on GLCM and Gabor filter: A comparison. *Optik* **2013**, *124*, 6469–6474.
10. Ding, S.; Liu, Z.; Li, C. AdaBoost learning for fabric defect detection based on HOG and SVM. In Proceedings of the International Conference on Multimedia Technology, Hangzhou, China, 26–28 July 2011; pp. 2903–2906.
11. Kwon, B.K.; Won, J.S.; Kang, D.J. Fast defect detection for various types of surfaces using random forest with VOV features. *Int. J. Precis. Eng. Manuf.* **2015**, *16*, 965–970.
12. Sarkar, A.; Padmavathi, S. *Image Pyramid for Automatic Segmentation of Fabric Defects*; Springer: Cham, Switzerland, 2018.
13. Weimer, D.; Benggolo, A.Y.; Freitag, M. Context-aware Deep Convolutional Neural Networks for Industrial Inspection. In Proceedings of the Australasian Conference on Artificial Intelligence, Canberra, Australia, 30 November–4 December 2015.
14. Weimer, D.; Scholz-Reiter, B.; Shpitalni, M. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Ann.* **2016**, *65*, 417–420.
15. Kim, S.; Kim, W.; Noh, Y.K.; Park, F.C. Transfer learning for automated optical inspection. In Proceedings of the International Joint Conference on Neural Networks, Anchorage, AK, USA, 14–19 May 2017; pp. 2517–2524.
16. Bian, X.; Lim, S.N.; Zhou, N. Multiscale fully convolutional network with application to industrial inspection. In Proceedings of the 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Placid, NY, USA, 7–10 March 2016; pp. 1–8.
17. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the CVPR 2009. IEEE Conference on Ieee Computer Vision and Pattern Recognition, Miami, Florida, USA, 20–25 June 2009; pp. 248–255.
18. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. *Tensorflow: A System for Large-Scale Machine Learning*; OSDI:USA 2016; Volume 16, pp. 265–283.
19. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In Proceedings of the Thirty-first AAAI Conference on Artificial Intelligence, Phoenix, Arizona USA, 12–17 February 2016.
20. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision*; Springer: Germany 2016; pp. 630–645.
21. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv* **2016**, arXiv:1602.07360.
22. Wu, C.; Wen, W.; Afzal, T.; Zhang, Y.; Chen, Y.; Li, H. A Compact DNN: Approaching GoogLeNet-Level Accuracy of Classification and Domain Adaptation. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 21–26 July 2017; pp. 761–770.
23. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv***2017**, arXiv:1704.04861.
24. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017.

25. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
26. Mullick, K.; Namboodiri, A.M. Learning deep and compact models for gesture recognition. In Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 17–20 September 2017; pp. 3998–4002.
27. Chen, K.; Tian, L.; Ding, H.; Cai, M.; Sun, L.; Liang, S.; Huo, Q. A Compact CNN-DBLSTM Based Character Model for Online Handwritten Chinese Text Recognition. In Proceedings of the Iapr International Conference on Document Analysis and Recognition, Kyoto, Japan, 9–12 November 2017; pp. 1068–1073.
28. Wong, A.; Shafiee, M.J.; Jules, M.S. MicronNet: A Highly Compact Deep Convolutional Neural Network Architecture for Real-Time Embedded Traffic Sign Classification. *IEEE Access* **2018**, *6*, 59803–59810.
29. Racki, D.; Tomazevic, D.; Skocaj, D. A compact convolutional neural network for textured surface anomaly detection. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 12–15 March 2018; pp. 1331–1339.
30. Kornblith, S.; Shlens, J.; Le, Q.V. Do Better ImageNet Models Transfer Better? *arXiv* **2018**, arXiv:1805.08974.
31. Chen, L.C.; Zhu, Y.; Papandreou, G.; Schroff, F.; Adam, H. Encoder-decoder with atrous separable convolution for semantic image segmentation. *arXiv* **2018**, arXiv:1802.02611.
32. Sun, C.; Shrivastava, A.; Singh, S.; Gupta, A. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 843–852.
33. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1610–02357.
34. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
35. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv* **2016**, arXiv:1611.06440.
36. Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; Narayanan, P. Deep learning with limited numerical precision. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1737–1746.
37. Xu, T.B.; Yang, P.; Zhang, X.Y.; Liu, C.L. Margin-aware binarized weight networks for image classification. In Proceedings of the International Conference on Image and Graphics, Shanghai, China, 13–15 September 2017; pp. 590–601.
38. Tai, C.; Xiao, T.; Zhang, Y.; Wang, X. Convolutional neural networks with low-rank regularization. *arXiv* **2015**, arXiv:1511.06067.
39. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
40. Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Process. Mag.* **2018**, *35*, 126–136.
41. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In Proceedings of the International Conference on Neural Information Processing Systems, Sydney, Australia,12-15 December 2012; pp. 1097–1105.
42. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
43. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, 26th June 2016; pp. 770–778.
44. Peng, C.; Zhang, X.; Yu, G.; Luo, G.; Sun, J. Large kernel matters—improve semantic segmentation by global convolutional network. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Hawaii, UAS, 21th July 2017; pp. 1743–1751.
45. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *40*, 834–848.
46. Krizhevsky, A.; Hinton, G. Convolutional deep belief networks on cifar-10. *Unpublished Manuscript* **2010**, *40*, 1–9.

47. Springenberg, J.T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv* **2014**, arXiv:1412.6806.
48. Caputo, B.; Hayman, E.; Mallikarjuna, P. Class-Specific Material Categorisation. In Proceedings of the Tenth IEEE International Conference on Computer Vision, Beijing, China,17th October 2005; Volume 2, pp. 1597–1604.
49. Kylberg, G. The Kylberg Texture Dataset v. 1.0. In *External report (Blue series) 35, Centre for Image Analysis*; Swedish University of Agricultural Sciences and Uppsala University: Uppsala, Sweden, 2011.
50. Lazebnik, S.; Schmid, C.; Ponce, J. A sparse texture representation using local affine regions. *IEEE Trans. Pattern Anal. Mach. Intell.* **2005**, *27*, 1265–78.
51. DAGM. Weakly Supervised Learning for Industrial Optical Inspection. DAGM Dataset. 2013. Available online: https://hci.iwr.uni-heidelberg.de/node/3616 (accessed on 26th March 2020).
52. Jung, A.B.; Wada, K.; Crall, J.; Tanaka, S.; Graving, J.; Yadav, S.; Banerjee, J.; Vecsei, G.; Kraft, A.; Borovec, J.; et al. imgaug. Available online: https://github.com/aleju/imgaug (accessed 25 September 2019).
53. Nilsback, M.E.; Zisserman, A. Automated Flower Classification over a Large Number of Classes. In Proceedings of the ICVGIP. Sixth Indian Conference on Computer Vision, Graphics & Image Processing, Bhubaneswar, India,16th December 2008; pp. 722–729.
54. Ren, R.; Hung, T.; Tan, K.C. A Generic Deep-Learning-Based Approach for Automated Surface Inspection. *IEEE Trans. Cybern.* **2017**, *5*, 1–12.
55. Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; Darrell, T. DeCAF: A deep convolutional activation feature for generic visual recognition. In Proceedings of the International Conference on Machine Learning, 2013; Volume 50, p. I–647.
56. Chen, W.; Gao, Y.; Gao, L.; Li, X. A New Ensemble Approach based on Deep Convolutional Neural Networks for Steel Surface Defect classification. *Procedia CIRP* **2018**, *72*, 1069–1072.
57. Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv* **2016**, arXiv:1605.07146.
58. Yu, Z.; Wu, X.; Gu, X. Fully convolutional networks for surface defect inspection in industrial environment. In Proceedings of the International Conference on Computer Vision Systems, Shenzhen, China, 10th July 2017; pp. 417–426.