Hindawi
Journal of Sensors
Volume 2022, Article ID 6703864, 22 pages
https://doi.org/10.1155/2022/6703864

Research Article

An Image-Based Deep Learning Approach with Improved DETR for Power Line Insulator Defect Detection

Yang Cheng and Daming Liu

College of Computer Science and Technology, Shanghai University of Electric Power, Shanghai, China

Correspondence should be addressed to Daming Liu; ldm@shiep.edu.cn

Received 4 April 2022; Revised 1 July 2022; Accepted 6 July 2022; Published 7 September 2022

Academic Editor: Mohamed Louzazni

Insulators are basic parts of high-voltage transmission, and detecting faults of insulators is a critical task. Most state-of-the-art methods contain two or more stages, including insulator detection and defect locating. Some also involve hand-designed components to improve the performance due to the complicated and misleading background of the wild. To automatically detect faults in UAV-captured insulator images, this paper presents a method that introduces DETR into insulator defect detection. With the self-attention mechanism in Transformer, the model can naturally exploit its advantage in focusing on the target area. However, training DETR requires large data sets and long training schedules to establish spatial relations in sparse locations, which makes it generally not feasible to train in small data sets. To explore the possibility of training a well-performing model with a data set that minimizes the cost of collecting insulator images, transfer learning techniques were applied to this process. To compensate for the disadvantage of DETR in detecting small objects at more precise scales, an improved loss was transplanted to this model. The results show that our proposed method can detect defects directly from UAV images without the need to locate the insulator first, while providing competitive performance with a lower cost of collecting training samples.

1. Introduction

Insulator, as a kind of special insulation control part, is widely used as an electrical component in high-voltage transmission. Insulators provide support and ensure insulation between the conductor and high-voltage wire connection tower. Due to various electromechanical stresses caused by changes in the environment and electrical load conditions, the damage could occur and the effect of insulators may be reduced, thus causing loss of reliability in the transmission of the entire line. Therefore, the detection of insulator defects is one of the most common and important tasks in the regular inspection and maintenance of the power grid.

Concerning the enormous amount of transmission towers standing in the wild, manual one-by-one inspection performed by human workers becomes impossible. With the help of unmanned aerial vehicles (UAV) and inspection programs on the computer, images of high-voltage line components can be collected and processed in a more labor-saving way. Then, there comes another problem, which is how to improve the accuracy and efficiency of detecting the insulator and defect. Huge amounts of effort have been applied to this issue.

There was a time when hand-designed classifiers and detectors took over this area. Image processing techniques using MPEG-7 EHD (edge histogram method) were applied [1], which is an algorithm to extract and calculate the edge characteristic of the insulator. Image segmentation by converting the color space of the image was proposed to extract insulators from aerial images [2]. Wu et al. [3] proposed a texture segmentation algorithm to partition aerial images into subregions based on the Principal Component Analysis (PCA) and Global Minimization Active Contour (GMAC) model. Another image processing method is proposed by Khalayli et al. [4], which uses co-occurrence matrices to extract texture features and classify these features by the polynomial classifier. A segmentation framework [5] using the active contour model to locate and extract insulators. Liao et al. [6] proposed a robust insulator detection algorithm to deal with the complex background in aerial images by local features and spatial orders. Zhai et al. [7] presented an algorithm based on saliency

and adaptive morphology in a unified framework with a limitation of insulator type. This was then resolved by Zhai et al. [8], which can detect defects of both glass and ceramic insulators based on spatial morphological features, while achieving high accuracy of over 91%. These methods widened the road to the detection of insulator defects, broadened our minds, and developed insulator detection methods. However, strong domain ability is needed when designing these approaches involving color and texture features or traits hidden in other forms, and hand-crafted feature extractors are prone to get disturbed by the complicated background and objects covering the insulators randomly.

With the development of GPU acceleration for computing, deep neural networks started to push the performance of the tasks in various areas to new state-of-the-art continuously and so it did to insulator defect detection. Several methods were proposed with segmenting insulators from the background before detecting the defect. Chen et al. [9] proposed a method that used a second-order full convolutional network (SOFCN) to extract the insulator area in the image and then obtain the fault region with the full convolutional network (FCN). Gao et al. [10] proposed a modified conditional generative adversarial network with a reconstructed generator and a discriminator composed of an FCN based on patchGAN to segment the insulator in pixel-wise. Sadykova et al. [11] presented a model based on YOLOv2 to detect faults using aerial images based on drones in various weather conditions. Tao et al. [12] presented a deep CNN cascading architecture that splits the entire detection task into two stages, including the localization of the insulator and the detection of defects, reaching a relatively high precision of 0.91 and a recall of 0.96. Zhao et al. [13] pushed the two-stage method further using Faster RCNN combined with improved FPN and color space adaptive algorithm. However, tasks were split into two or more stages to acquire higher accuracy and hand-crafted techniques are still needed to reduce the interference of complicated background in the aerial images.

This paper focuses mainly on simplifying the pipeline of insulator fault detection and improving the performance with a relatively low cost of collecting training data. Carion et al. [14] proposed DEtection TRansformer (DETR) as an end-to-end object detector which adopted the encoder-decoder architecture of transformer [15]. With the help of the attention modules, the network can generate attention-aware features from the image and focus more on the selected region, thus contributing to identifying the insulator defect from complicated backgrounds, and it can also produce representations with more discerning features to improve accuracy. However, the data sets are extremely large to train transformer models and converge to an acceptable level. Collecting and labeling data sets in this order of magnitude are not usually feasible in insulator defect detection, and data sets containing thousands of images are quite large in this area. To mitigate this, we propose an insulator fault detection model based on DETR and transfer learning methods with a simplified process compared to other methods and provide competitive performance. Also, an improved loss was grafted to this model to help solve the imbalance problem and suppress the disruption from useless backgrounds of the aerial images. The core contributions of this paper are as follows:

(1) We propose a new power line insulator defect detection method based on image recognition in a deep learning way leveraging the self-attention mechanism of transformer without the need of designing hand-crafted and expert-knowledge-based components

(2) We explored and tested several different layer clipping choices of fine-tuning to reduce the amount of training data and improve the performance of detecting defects

(3) We tested and verified the transplanted focal loss on DETR of alleviating the class imbalance problem on insulator defect data and improving accuracy

The remainder of this article is organized as follows. Section 2 introduces the structure of Transformer and DETR. Section 3 presents the insulator defect detection method. In Section 4, experiments and results are presented. The conclusions and discussions are in Section 5.

2. Revisiting Transformer

Transformer models [15] have been proposed in the field of Natural Language Processing (NLP) with astounding performance. It has now been widely applied in NLP domains such as machine translation [16] and many others. In recent years, Transformer models have attracted a lot of sights in the computer vision area and have been adapted into different tasks such as image classification [17] and object detection [14]. The structure of Transformer model is shown in Figure 1. The left part is the encoder and the right part is the decoder. The transformer is composed of several couples of encoders and decoders. Each encoder has the same structure which consists of a multi-head attention layer and a feed-forward layer. After each layer, there exists an add and norm operation which adds and normalizes the output of this layer and the residual connection of the previous layer. Then, the output of the encoder was sent as the input of the decoder's multi-head attention. The decoder looks much like the encoder but with an additional masked multi-head attention layer which takes the output embedding as input and sends its output to the decoder's multi-head attention. The input embedding and output embedding are processed by a positional encoding layer to mix the positional information into the embedding sequence. The output of the decoder is sent to a linear layer and softmax layer to generate output probabilities.

In Figure 1, some of the individual blocks can be easily found in the popular frameworks of deep learning. The FFN block is a simple feed-forward network and the Add&Norm block just adds the residual connections and performs layer normalization [18]. The Linear&Softmax block performs linear projections and produces the probabilities of prediction by softmax operation. The other blocks are discussed in the subsections.

2.1. Self-Attention. Self-attention can be taken as an aggregator which can calculate the association between each element

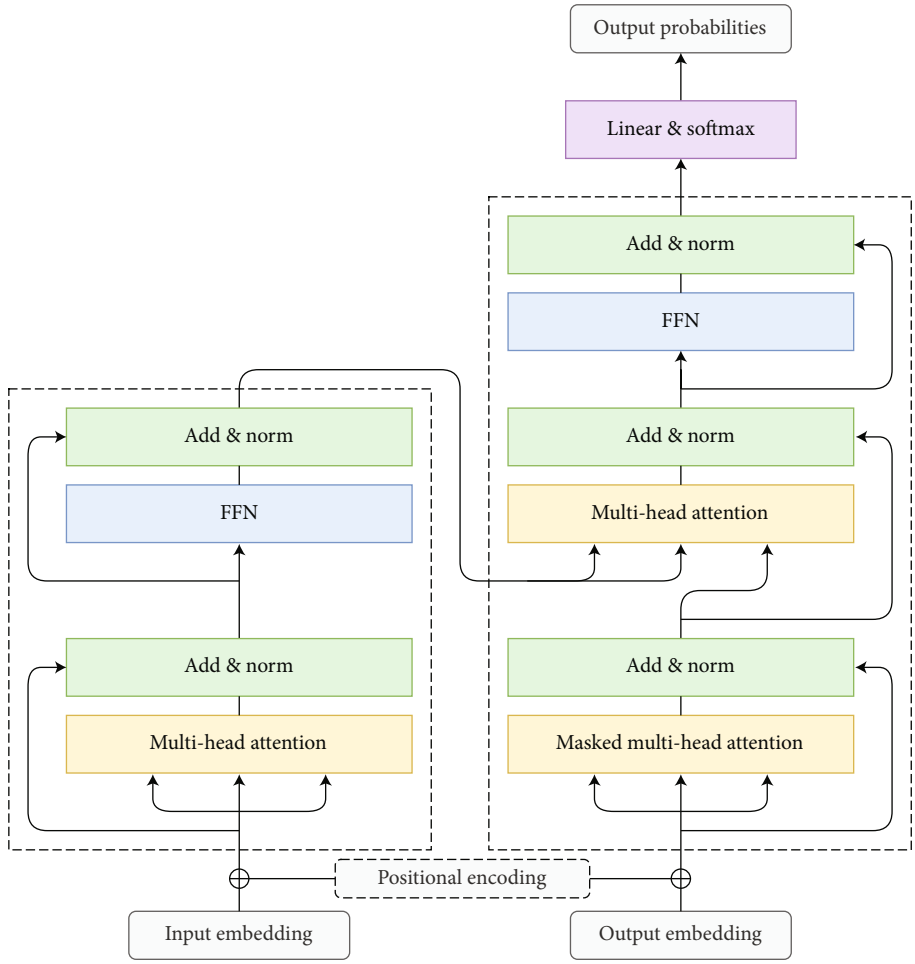


FIGURE 1: The model structure of Transformer [15]. The three arrows generated from one are part of the self-attention mechanism, which corresponds to Q, K, and V in Equation (1) generated from the same input by three different linear projections. That is why it is called "Self-Attention." The two arrows making from one are the K and V generated from the encoder and along with the Q from the previous layers of decoder as inputs to the multi-head attention which is also called "Cross-Attention."

and all other elements in the same context. In language-related tasks, these elements are the embedding sequences of words. When processing images, the elements are representations of pixels. An element more related to another will have a higher attention score. These attention scores can be converted to weighted representations to help the model make decisions. The computation of self-attention layer can be demonstrated as follows: given d as the dimension of the embedding and n as the number of the vectors, $X \in \mathbb{R}^{n \times d}$ as the vector sequences, three matrices W^q, W^k, and W^v to be adjusted through the learning process as suitable weights and another three matrices generated from the same input X by multiplying W^q, W^k, and W^v as $Q = XW^q$, $K = XW^k$, and $V = XW^v$, the attention score can be calculated as

$$\text{Attention} = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V. \quad (1)$$

The dot product of $Q \cdot K^T$ is divided by a scale $\sqrt{d_k}$ which is the dimension of a key vector to normalize the value. Then, use the Softmax operation to convert the result to a probability distribution and then multiply it by the matrix V to obtain the weighted summation. Elements with a higher attention score will receive more focus while processing the inputs.

There are two arrows making from one and along with another one as inputs of the multi-head attention in Figure 1. That is part of the cross-attention mechanism pretty much the same as self-attention but with K and V replaced by linear projection results from another input which is the output from the encoder. Unlike the self-attention modeling the relations between the elements in a sequence, the cross-attention module can calculate the similarity of the vectors, which can be taken as the degree of relationship between the elements and queries.

2.2. Multi-Head Attention. There exists a concern that only one self-attention head may not focus on the most related element. The stochastic initialization operation may put the start point to a place that will lead the self-attention head to focus on a local relevant element, not the global one. This situation can be improved by adding multiple self-attention heads, which are called Multi-Head Attention. With multiple heads to scan the elements, the randomness of the

initialization turns into an advantage that can distribute the heads to different start points that converge to different locations of the elements. Then, the output of the self-attention head $h_1 h_n$ can be concatenated as follows:

$$\text{MultiHead} = \text{Concat}(h_1, \cdots, h_n) W', \tag{2}$$

where W' is a matrix of learnable weights to project the output to the final multi-head attention values.

2.3. Masked Self-Attention. In certain aspects, an encoder-decoder couple is designed to predict the probabilities of the next element in the sequence, which means that it should not peep at the ground truth elements in the output embeddings. In other words, the module can learn nothing from cheating. But the self-attention layer can participate in the processing of all elements. Then, a mask was added to the self-attention layer to make it blind for the future sequence, and this is called Masked Self-Attention. It is implemented by calculating the Hadamard product [19] of the attention value and an upper triangular matrix M:

$$\text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}} \circ M\right). \tag{3}$$

Then, the masked self-attention layer can be used in the decoder to process the output embeddings without the concern of seeing the future.

3. Method

3.1. Structure of DETR. DETR [14] was an adaption of the transformer model to simplify the general object detection pipeline. Several hand-designed components were widely used in traditional models relying on prior knowledge in certain fields, setting anchors or non-maximal suppression steps. With the transformer encoder-decoder modules and the well-designed architecture of DETR, end-to-end object detection became more than possible. The detection task was treated as solving a set prediction problem based on two parts, which are the self-attention module and the bipartite matching loss. The relationships of the elements in a sequence can be discovered and established by the self-attention architecture and meanwhile can help to generate a set of bounding boxes. The loss function can measure the differences between the predicted box and ground truth boxes based on the bipartite graph matching so that to make the position and category of the predicted box closer to the ground truth.

The simplified overall architecture of DETR is illustrated in Figure 2. It contains three parts to work: a CNN backbone to generate activation maps with lower resolution from the initial images, several encoder-decoder couples to process the activation maps, and feed-forward networks (FFN) to predict the classes and bounding boxes. Positional encoding was added to the activation maps to retain positional information. Unlike the original transformer architecture proposed in [15], the DETR transformer decoder module processes all object queries at the same time without masking some of them out. In other words, to take advantage of the ability of the self-attention module that can globally reason the relationship between all objects, the context of the whole image shall be preserved for the transformer decoder. The final predictions were made by the feed-forward networks. Three layers of perceptrons were used to predict the coordinates, heights, and widths of the bounding boxes, while the class labels were predicted by a single linear layer.

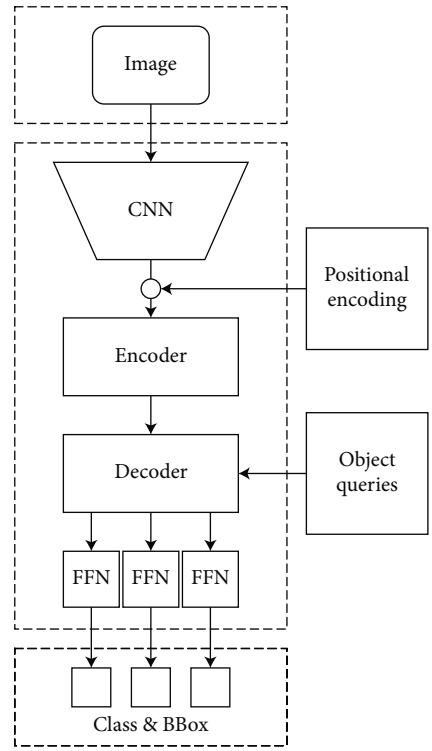


Figure 2: The simplified representation of DETR structure [14].

To find the best prediction for the class labels and bounding boxes, the matching loss based on the Hungarian algorithm [20] can be described as:

$$\ell_h(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + 1_{\{c_i \neq \varnothing\}} \ell_{\text{box}}\left(b_i, \hat{b}_{\hat{\sigma}}(i)\right) \right], \tag{4}$$

where y represents the ground truth objects, $\hat{y}$ denotes the predicted ones, and $\hat{\sigma}(i)$ can be taken as the index of the predicted box. And N needs to be set as a number larger than the number of actual objects that could appear in the image. To match the numbers, a class label $\varnothing$ was added to the classes of objects to represent the case "No object." The loss for the bounding boxes is defined as follows:

$$\ell_{\text{box}}\left(b, \hat{b}_\sigma\right) = \lambda_{\text{iou}} \ell_{\text{iou}}\left(b, \hat{b}_\sigma\right) + \lambda_{L1} \left\| b - \hat{b}_\sigma \right\|_1. \tag{5}$$

When the matched object falls into the label $\varnothing$, the corresponding loss value becomes zero. Otherwise, the loss will be determined by the probability of being in the same class as the predicted class and the ground truth, and by the difference between the predicted box and the actual one. The loss of boxes is calculated by the intersection over union (IoU) and the $L1$ norm of the coordinates.

Table 1: Treatment of layers.

Transferred weights	Backbone Transformer	ResNet Encoder/decoder layers
Clipped and retrained layers	Class embedding	$\text{Linear}(256 \times 80) \longrightarrow \text{Linear}(256 \times 2)$
	Bbox embedding	$\text{Linear}(256 \times 256) \longrightarrow \text{Linear}(256 \times 128)$
		$\text{Linear}(256 \times 256) \longrightarrow \text{Linear}(128 \times 128)$
		$\text{Linear}(256 \times 4) \longrightarrow \text{Linear}\ (128 \times 4)$

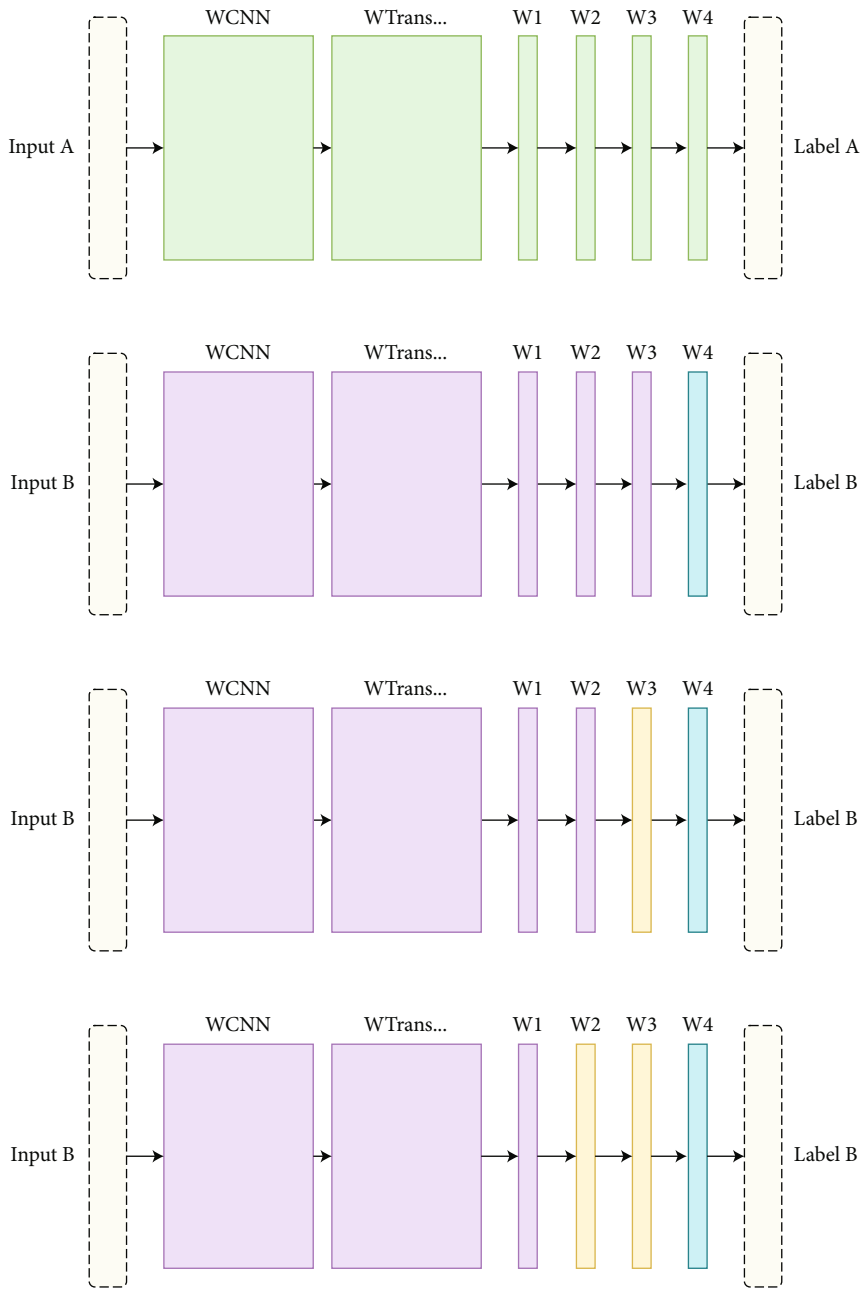


Figure 3: Overview of the treatment and control of the pre-training and fine-tuning of this model. The model was pre-trained on domain A and transferred to domain B. Rectangles colored with green represent weights obtained from pre-training of domain A, the purple ones represent weights adjusted in fine-tuning, the blue ones represent the layer of weights for classification which are clipped and retrained on domain B, and the yellow ones represents the linear layer weights for bounding box regression which are retrained on domain B.

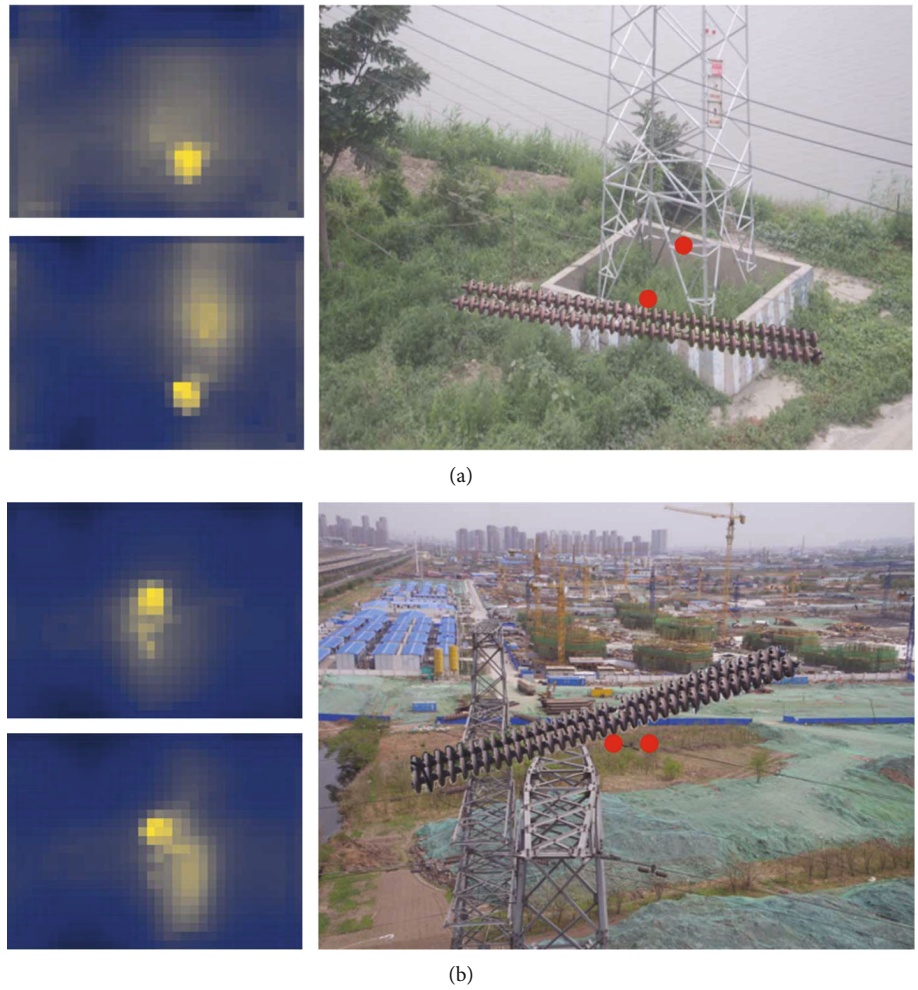


FIGURE 4: Visualization of the weights of the last encoder layer. Take two points around the defect as reference points, the weights of self-attention shows a high correlation with defect areas. The images of (a) and (b) are two samples randomly selected from the data set.

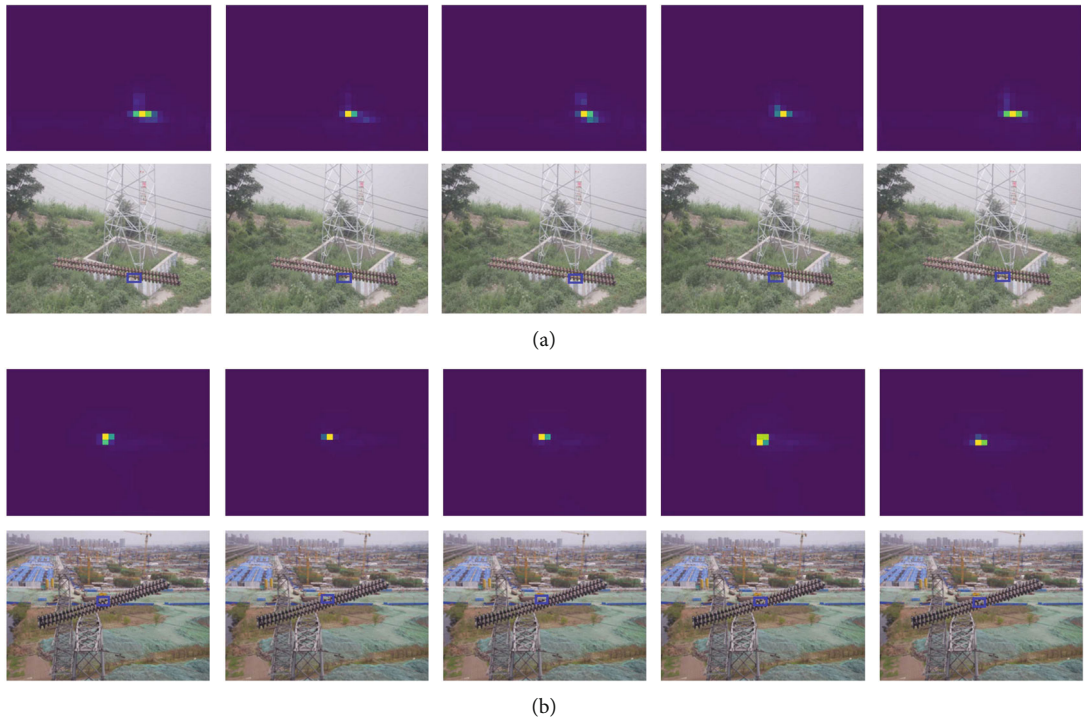
The λ_{iou} and λ_{L1} are super parameters to determine the proportion of these parts.

3.2. Transfer Learning. Models based on Transformer can exert the advantage of self-attention mechanism to exploit the spatial relations of the elements in a sequence and help to acquire higher accuracy. However, the difficulty [21] of training transformers hinders us from further exploration in this area. Models with Transformer structures usually need a huge amount of data and more training epochs to converge compared to traditional networks. That is partly derived from the amount of the necessary information to establish the correct relations of the elements. [22] have provided several observations on this issue to help get better results, but the amount of data in insulator defect detection is still quite hard to reach the requirement of Transformer structures. Thus, transfer learning methods are transferred into this area to help change the model from untrainable to well-performing.

Transfer learning [23] is an efficient framework in machine learning. A normal training process is based on the assumption that training and testing data are distributed independently in an identical way. When the assumption cannot be satisfied, transfer learning can be applied to tackle this problem. Studies [24, 25] have been pushed upon this diagram and showed the ability to improve performance through learning information from other data sets. Basic notations of transfer learning are briefed as follows:

Given $\mathscr{D}_S$, $\mathscr{T}_S$, $\mathscr{D}_T$, and $\mathscr{T}_T$ as the source domain, task and the target domain, task satisfying $\mathscr{D}_S \neq \mathscr{D}_T$ or $\mathscr{T}_S \neq \mathscr{T}_T$. Transfer learning utilizes the knowledge from $\mathscr{D}_S$ to help accomplish the $\mathscr{T}_T$ or achieve better performance for it.

As have been stated in [26], learned features in different layers in a deep neural network contribute differently to the target learning task. The model in this paper was pre-trained on COCO [27] which is a well-known large-scale data set for the purpose of common object detection, segmentation, and captioning. There are over 330000 images for training, 210000 of which are labeled, and there are 1.5 million instances of objects and 80 categories. To maximize the advantage of the learned features of general objects, most layers of the pre-trained model are preserved, and linear layers on the top are clipped to adapt to the target domain. Then, the model was fine-tuned on a relatively small-sized data set. The preserved layers also participated in the fine-



(a)

(b)

Figure 5: Visualization of the attention weights for several query ids of the last decoder layer. The corresponding decoder attention weights of the predicted bounding boxes show that the weights are clustered on the defect areas. The images of (a) and (b) are the corresponding images of Figure 4.

tuning process and are adjusted to fit the new task. The classification layer was clipped and retrained during the fine-tuning process and two layers for bounding box regression are clipped progressively to explore the transferable ability of these layers and how do they produce boost effects on generalization and specification ability when transferring and fine-tuning on a new data set.

The treatment of the layers is presented in Table 1. The weight of the backbone and transformer part is transferred and the weight of class embedding and bbox embedding are retrained during fine-tuning. The size of class embedding and bbox embedding layers are clipped as in Table 1. Note that during our experiments one more layer clipped of bbox embedding, the performance dropped one more step and all bbox layers clipped will lead to unacceptable performance. So in Figure 3, we only presented two clipping conditions. We will talk about it in case study of failure in Section 4. The full detailed structure of DETR is shown in the Appendix.

3.3. Focal Loss. Lots of models predicting bounding boxes are based on the process of proposing unbiased initial boxes and performing regression based on these boxes. DETR [14] predicts a set of bounding boxes among those N initializations where N is predefined as a number much larger than the actual number of possible target boxes. This feature introduces an imbalance between the actual targets and the useless predictions. The widely used loss that DETR adopts is the standard cross-entropy loss, which can be calculated as follows.

$$CE(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases} \tag{6}$$

Here, y is the predicted binary for the class label and p is the predicted probability. If we simplify the branch for p and $1 - p$ as p_t, then we get:

$$CE(p_t) = -\log(p_t). \tag{7}$$

The function indicates that the cross-entropy loss treats the predictions of all the samples equally, and the optimization process will measure the steps in the same way. Lin et al. [28] proposed focal loss to help the model descend in a way focusing on the samples that are harder to learn:

$$FL(p_t) = -(1-p_t)^{\gamma} \log(p_t), \tag{8}$$

where γ is an adjustable parameter between 0 and 5 that can control the degree of neglect or focus. As the p_t of the easy samples grow close to 1, the loss values of these samples are ignored to a certain extent while those hard examples are naturally emphasized and reflected in the overall loss. This improved loss compared to the canonical focal loss can relieve the imbalance problem during the training process and help to improve the model's accuracy.

3.4. Defect Detection. Compared to the two-stage methods, our proposed method can detect the defect location directly from the images. The image to be tested is fed into the backbone part based on a convolutional neural network to generate feature maps with concentrated representations. ResNet [29] is used as the backbone in this part. Degradation problems start to become insurmountable when normal deep networks become deeper, and the deeper layers will look like starting to just copy the features learned by shallower layers.

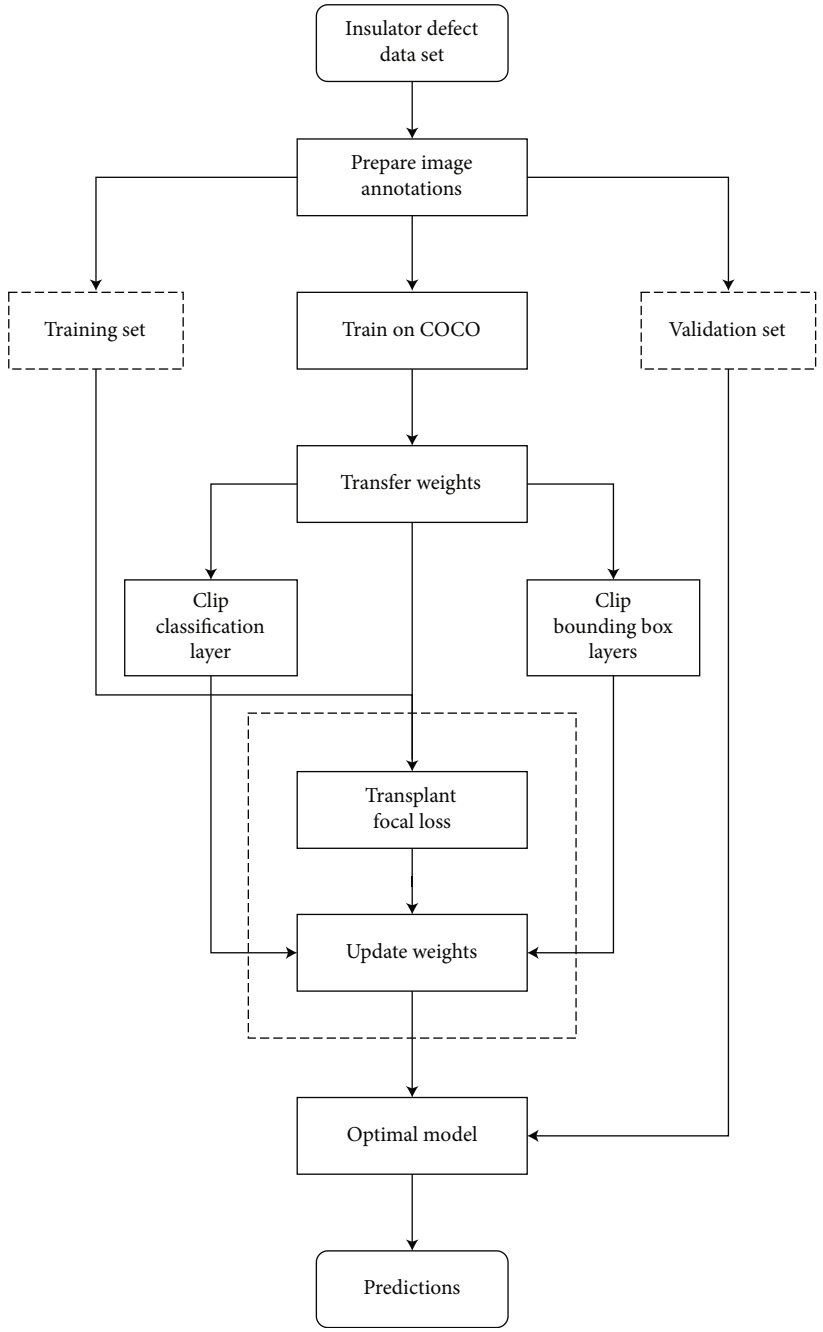


Figure 6: The overall process of the method.

With the residual connection, ResNet can learn more complicated feature maps with deeper construction.

The learned feature maps are then collapsed to one feature map of $d \times HW$ after reducing the dimension of feature maps by a 1×1 convolution layer, where d denotes the reduced dimension and HW represents the height and width of the collapsed feature map. Then, the feature map shall be sent to the encoder layers with position information added by the positional encoding step to supplement the permutation-immutability of transformer architecture. Sine and cosine functions are used to adjust the frequencies according to the dimension of the sequence. Given pos as the position of the input sequence and i as the dimensional serial in the sequence, the positional encoding can be computed like below:

$$PE_{(\text{pos},2i)} = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right), \tag{9}$$

$$PE_{(\text{pos},2i+1)} = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right). \tag{10}$$

Based on the supplemented inputs, the encoder layers forward the sequences to establish the attention relations of the elements. In these layers, global information of the whole scene participated in the process of computing the

(a) (b)

Figure 7: Samples of the images from data set B and bounding box predictions with confidence score.

(a) (b)

Figure 8: Box predictions for the 20 epoch on CPLID. (a) and (b) are two samples to illustrate the prediction results of the early learning process.

attention score. The elements in the input sequence will be modeled to find the matching parts in the image without disturbance from the background. Instances are separated progressively in an approximate way. As shown in Figure 3, the weights of the last encoder layer focus on a few parts of the image around the defects. Self-attention is a fairly large matrix that contains the attention scores for every element over the others, so we randomly selected two reference points near the defect location to visualize the attention weights.

The attention weights are then processed by the decoder layers with the attending of the object queries. The object queries are learned positional encodings as an embedding layer to project the position information of the objects into a higher dimension and are trained altogether with the model. The decoder aggregates the output of encoder layers and the object queries to learn the local defect information given the baseline of the separated area. Visualizing the weights of the last encoder layer is shown in Figure 4. Visualizing the weights of the last decoder layer for different queries is shown in Figure 5. The output of the decoder layer is then sent separately to two feed-forward networks to produce the final predictions: box coordinates and class labels.

The overall pre-training and fine-tuning process of the proposed method is shown in Figure 6. The data set containing images of insulator defects is annotated with the defect class labels and bounding box positions. Then, the data set is divided into the training set and validation set. The model is pre-trained on the COCO [27] data set which is a public large-scale data set for object detection. The weights of the pre-trained model are saved and prepared to load into the next learning phase of the method. Different layers of the pre-trained weights including the classification layer and bounding box layers are clipped. Focal loss is transplanted to the model during the fine-tuning phase, and the weights are adjusted and updated through more like a full fine-tuning process, which means that all the pre-trained weights but the clipped layers are involved in the training process on the insulator defect data set. The weights of the clipped layers are discarded and relearned during fine-tuning. After fine-tuning, the optimized model is tested on the validation set to verify the performance. Finally, the optimal model aligned and compared among the different clipping strategies is used to detect the insulator defects.

The key point that makes the model competitive is the multi-head self-attention (MSA). MSAs are basic components

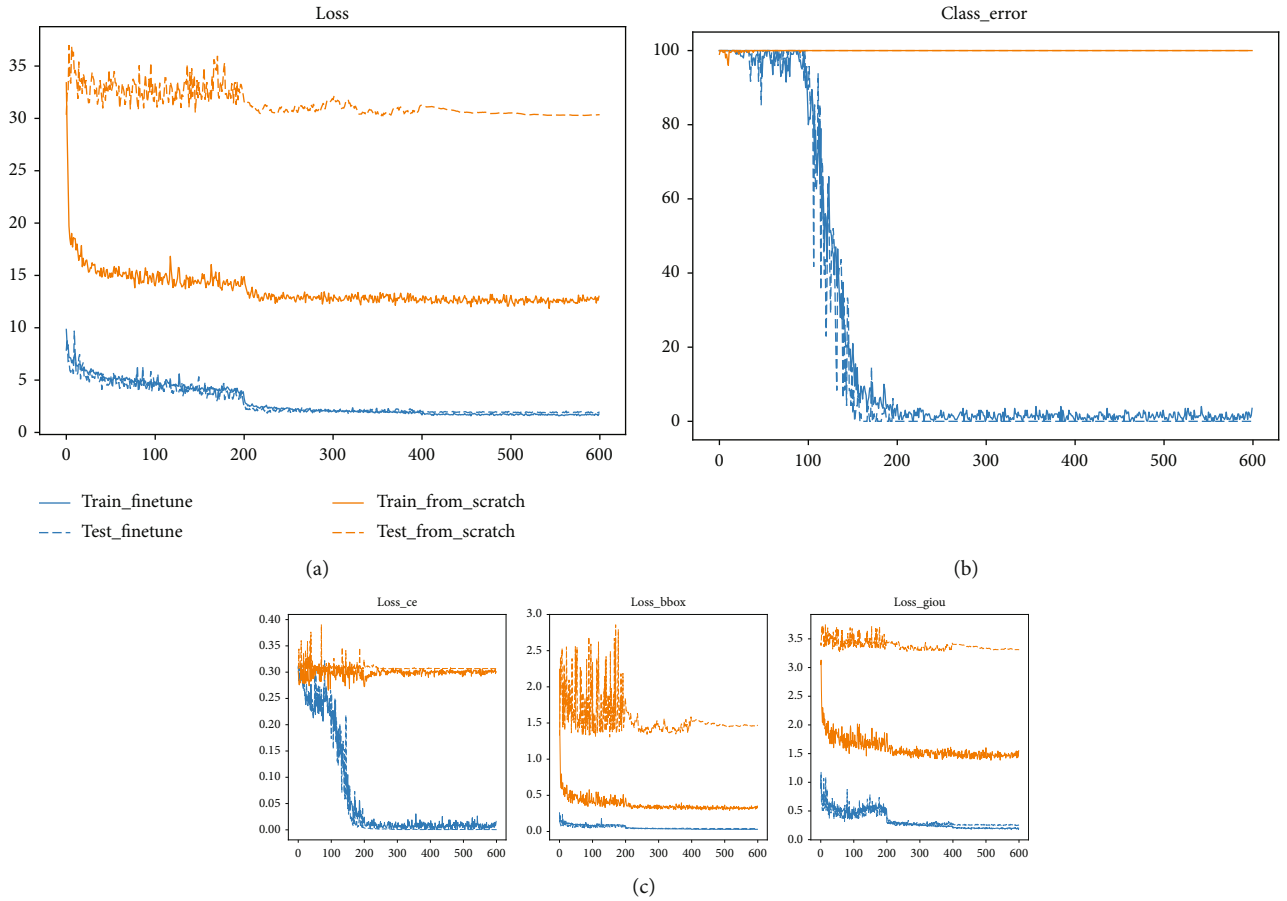


FIGURE 9: The training and testing loss and error of training on CPLID are plotted. The x axis represents the number of training epochs and the y axis denotes the value of loss or class error. Lines colored orange are outputs of the model trained from scratch, and blue lines belong to the fine-tuned model after pre-training. (a) The total loss. (b) The class error indicating the extent of misclassification. (c) The loss for class error, bounding boxes, and GIoU. The overall loss is composed of these three losses. Note that the loss curves are only for illustrating purpose, and the actual precision curves are presented in Figure 11.

of Transformer models. Some research helps us to explore further the intrinsic properties of the Transformer. Tuli et al. [30] show that vision models based on Transformer-like structures have a lower level of inductive bias but a higher gain in shape bias, which can contribute to stronger stability. Naseer et al. [31] analyze the intriguing properties of Transformers in vision tasks by tests on fifteen data sets and conclude that the Transformer models have higher robustness to occlusion, distributional shifts, and patch permutations thus can help to focus and separate the defect target in the images. However, along with the advantages come the disadvantages: the higher algorithm complexity of $O(n^2 \cdot d)$ where n is the sequence length and d is the representation dimension and the model becomes harder to converge when training on small data sets, which is a common scene in the defect detection area. The complexity problem gets relieved by the early processing of CNN which generates smaller-sized feature maps and transfer learning plays the role to resolve the learning problem. The defect detection task comes with an imbalance of defect types and the bounding box mechanism of DETR exacerbates the imbalance. Focal loss with the elaborate balance factor helps to relieve this problem. The experiments are analyzed and shown in the next section.

4. Experiments and Analysis

The platform for the experiments is an Ubuntu 16.04 machine equipped with a GeForce RTX 2080 SUPER and an Intel Core i5-8500 CPU @ 4.1GHz (32G RAM). The program runs on Python 3.9.7 and the framework used for deep learning is PyTorch 1.9.1.

4.1. Data Set and Evaluation Metrics. The data set we are using is the one called "CPLID" (Chinese Power Line Insulator Dataset) [12], providing normal insulator images captured by UAVs and defect insulator images augmented by synthetic methods. The data set was provided by State Grid Corporation of China and has been used in several studies [12, 13, 32, 33] in this area and the data set is currently available online. The data set contains 600 images of normal insulators and 248 images of defective insulators. The images are labeled in VOC [34] format which are XML files

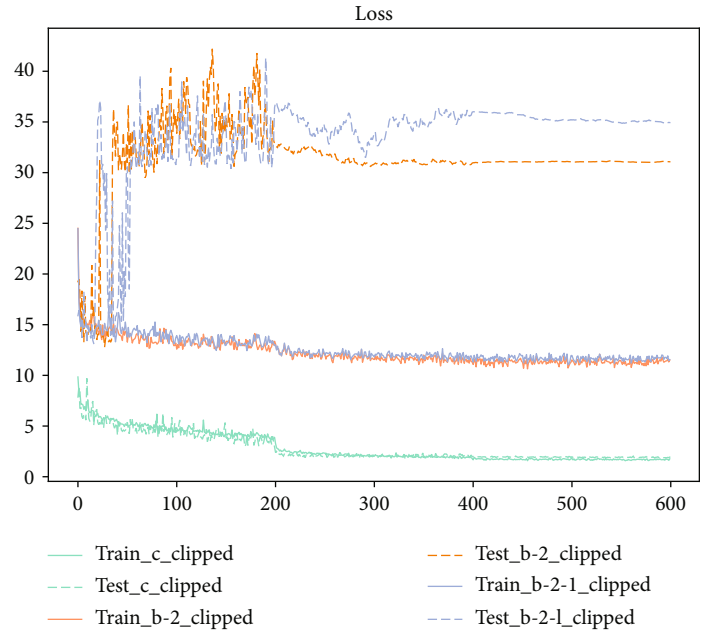


Figure 10: The loss of training and testing of models on CPLID with different clipped layers. The x axis represents the number of training epochs and the y axis denotes the value of overall loss. The green lines are from the model of the classification layer clipped. Orange lines and red lines are from the models of bounding box layers clipped.

Table 2: Performance of contrastive models on CPLID.

Model	Backbone	$AP_{\text{IoU=.50}}$	$AP_{\text{IoU=.75}}$	$AP_{\text{IoU=.50:.05:.95}}$	Speed (ms)
RetinaNet	ResNet101	0.992	0.953	0.762	80.6
RetinaNet	ResNet50	0.993	0.956	0.777	60.2
YOLOv5x	CSPDarknet53	0.995	0.960	0.794	26.6
YOLOv5l	CSPDarknet53	0.995	0.969	0.795	14.2
YOLOv5m	CSPDarknet53	0.995	0.960	0.795	**8.2**
TOOD	ResNet101	**1.000**	0.969	0.764	86.2
TOOD	ResNet50	**1.000**	0.964	0.787	65.7
Sparse R-CNN	ResNet101	**1.000**	0.981	0.758	86.9
Sparse R-CNN	ResNet50	**1.000**	0.992	0.785	67.1
DETR	ResNet101	**1.000**	0.965	0.752	63.3
DETR	ResNet50	**1.000**	0.985	0.753	41.3
DETR-Focal	ResNet101	**1.000**	0.974	0.769	63.3
DETR-Focal	ResNet50	**1.000**	**1.000**	**0.797**	41.3

and for the convenience of our proposed method, the annotations are converted to COCO [27] format of JSON file. The defect images are trained and evaluated in our proposed method and the images are randomly divided into the training set and testing set by a ratio of 4 : 1. The size of the images is adjusted to 739×558. The "CPLID" data set is synthetic and published several years ago and only contains composite insulator images. To give our proposed method a better verification, we collected 130 real-world glass insulator images taken by UAVs and tested our method on it. We call these images "data set B" in our paper. Some of the images are shown in Figure 7.

Several widely used evaluation metrics are adopted in our experiments to assess the performance and accuracy of the proposed method: Precision (P), Recall (R), and Average precision (AP). The definitions of these metrics are as follows:

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, \tag{11}$$

$$AP = \int_0^1 P(R)dR, \tag{12}$$

where TP denotes the number of defects that are correctly located, FP represents the number of locations reported as defects, while actually not, and FN represents the number of defects that have not been detected correctly. The

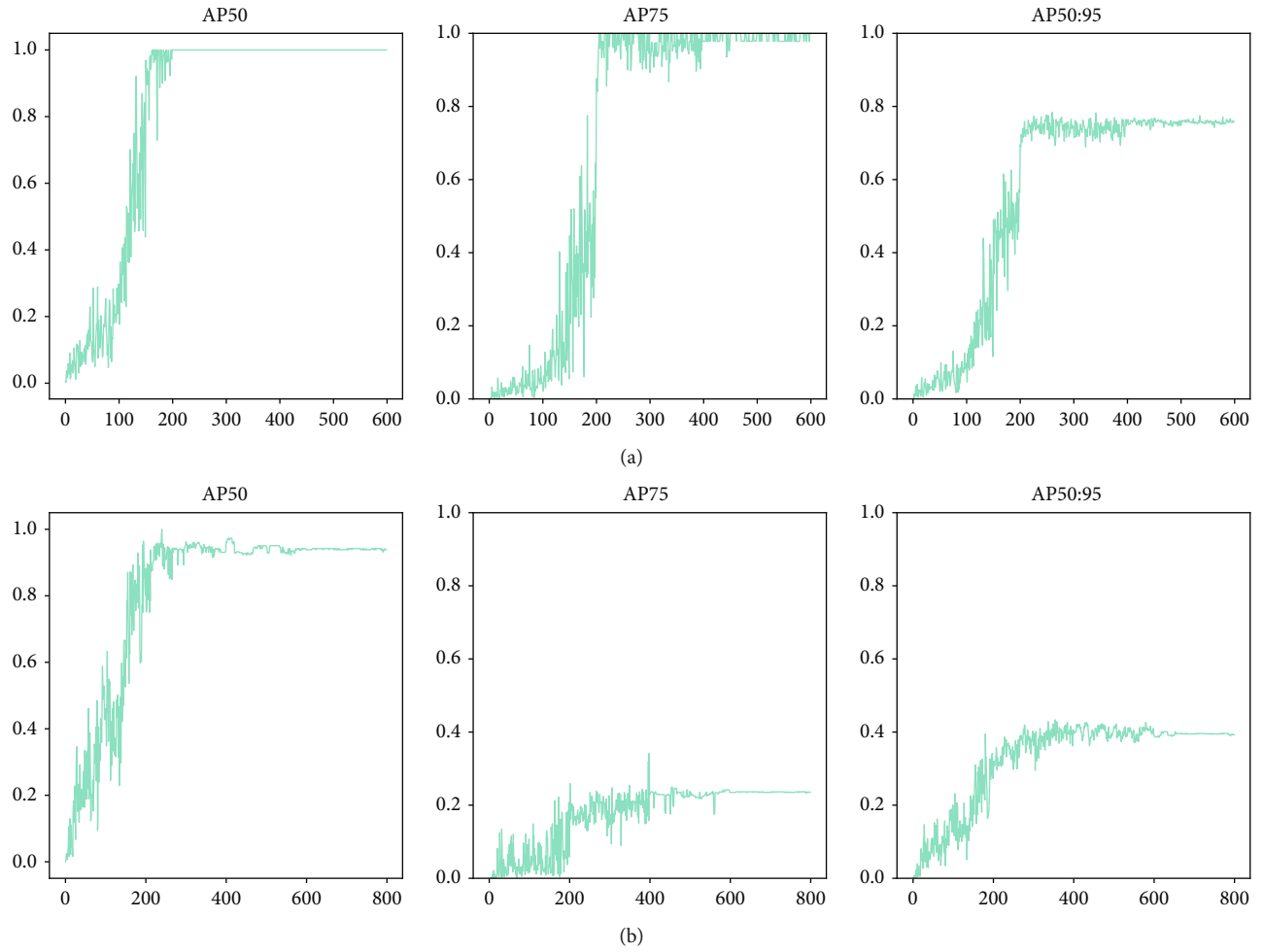


Figure 11: The average precision curves of IoU50, 75, and 50 : 95. (a) is the results of data set "CPLID," a synthetic composite insulator data set, and (b) is from data set "B," which is a data set containing real-world glass insulator images collected by us. As we can see from the curves, the AP75 and AP50:95 of real-world data dropped a lot compared to the synthetic, and the AP50 curve converges to a competitive level.

correctness of the located bounding boxes is determined by the threshold of intersection over union (IoU):

$$\text{IoU} = \frac{\text{Area}(G \cap P)}{\text{Area}(G \cup P)}, \tag{13}$$

where G denotes the ground truth boxes and P stands for the predicted boxes. In the previous methods of insulator defect detection, the most commonly used metric follows the rule of VOC, that is, a predicted box with an IoU threshold greater than 0.5 shall be considered a successful detection. Here, we step one more further to the more detailed and accurate COCO evaluation metrics. The average precision was calculated in three different IoU threshold conditions: 0.5, 0.75, and the average value between 0.5 and 0.95 in steps of 0.05.

4.2. Detection of Insulator Defects. The training was performed with the AdamW [35] optimization method. The learning rate was set as 10^{-4} for the transformer, 10^{-5} for the backbone, which is an ImageNet pre-trained ResNet model. A weight decay of 10^{-4} was used. After training on COCO data set, the model was fine-tuned on CPLID and the complete fine-tuning process took 500 epochs. The size of the mini-batch is set to 2. The γ value of focal loss was set to 2.5. As we observed in the fine-tuning process, the model has started to converge after merely 20 iterations. As shown in Figure 8, the bounding boxes predicted with relatively low confidence scores have started to cluster around the defect location; meanwhile, the model still needs more iterations to determine the more exact positions of the defects.

The loss charts are shown in Figure 9. The training results are plotted in solid lines and the validation results are depicted in dashed lines. The class error is the 100's complement of the accuracy for the classification of the defects. The overall loss is the weighted sum of losses for class error, bounding boxes, and generalized IoU (GIoU) [36] scores. The class error loss is the standard cross-entropy loss calculated between the predicted class and the ground truth. The loss of the bounding box is the $L1$ loss for the coordinates. The GIoU loss is an improved IoU loss that can perform well in different cases of overlapping

or even no overlap at all. To make a graphical comparison between the losses of the fine-tune and train-from-scratch models, the outputs are depicted together. We can find that the overall fine-tune loss experiences a sharp drop at the beginning of training, then descends in a more gentle way while the losses of the bounding box and GIoU oscillating around, finally converges to a flatland that hardly changes as the training progresses further. The validation loss correspondingly decreases with the training loss and converges to a level similar to the training loss. Class error also drops to a level slightly above zero, which means that most defects are correctly classified. However, the orange lines appear to proceed differently. The training losses also drop significantly at the beginning but quickly stop at a relatively high level without any further changes. Meanwhile, the validation losses are shocking ineffectively and then stay stable at a level slightly below the starting point. The class error does not drop at all, indicating that the model trained from scratch completely failed at classifying the defects. The training-from-scratch model just overfits the limited training data and behaves incomprehensibly on the testing data. It can be known that the model without pre-training and directly trains on the insulator defect data failed to learn and converge. On the contrary, the fine-tune model successfully went through the learning process and converges to an acceptable level.

4.3. Case Study of Failure. Not all the clipping choices bring improvement on the target domain task. To find and verify the feasible and suitable method of clipping layers, some explorations are carried out to a limited extent of how the last layers affect the model's accuracy in this task. There are two separate stacks of feed-forward layers on the top of the model: one layer for classification and three layers for bounding box regression. We name these layers c, $b0$, $b1$, and $b2$ for convenience. The experiments are conducted under these conditions: First, only the classification layer is clipped. Second, the last bounding box is clipped and thirdly, the last two bounding box layers are clipped. The losses are shown in Figure 10. For models with bounding box layers clipped, the training losses stick to a high level near the starting position and just drop very slightly throughout the whole process. The testing losses explode to a very high level and remain stable after a period of shock. The orange lines lay just slightly below the red lines, indicating that one more clipped layer caused a slight drop in performance, but these two losses do not make any significant difference. Models with clipped bounding box layers failed to learn and converge on the target task. It can be found that the weights of the transferred bounding box layers are important and useful in the fine-tuning process and should not be abandoned. It can be inferred that a feasible way to get a well-fine-tuned model is to only clip and reconstruct the classification layer with full model weights adjusting in the learning process on the target domain.

4.4. Results in Comparison. To compare the performance of different methods and models, contrastive experiments are conducted on CPLID. Several models are tested following the same way of pre-training and fine-tuning in our proposed method, including RetinaNet [28], YOLOv5 [37], TOOD [38], Spase R-CNN [39], and their variants with different network sizes. The results of the comparison are presented in Table 2.

For the IoU threshold of 0.5, DETR models reached AP performance of 1.000 and for the IoU threshold of 0.75 and $AP_{IoU=.50:.05:.95}$, the DETR-Focal model with a backbone of ResNet50 achieved the best performance of 1.000 and 0.797 which outperforms the others. Notably with a backbone of ResNet50, the DETR-Focal model received 1.5% and 4.4% boosts on $AP_{IoU=.75}$ and $AP_{IoU=.50:.05:.95}$ compared to the corresponding DETR model and with a backbone of ResNet101, the DETR-Focal model received 0.9% and 1.7% boosts. However, the YOLOv5m model has a faster inference speed of 8.2 ms compared to the others. Our proposed method performed well at the common and stricter metric levels, but there is still room for improvement at the inference speed. We noticed that for each kind of model, the best performance was not achieved by the model with the largest backbone or network size. That is partly because for tasks with a few classes and objects to detect, a larger backbone can contribute to the result of overfitting, and choosing a proper network size will help to improve the performance in practical applications.

Note that the data above is from data set CPLID, a synthetic composite insulator defect data set, which only offers a reference to the defect detection performance on composite insulators. To verify the validity of our method, we trained and tested the model on 130 real-world glass insulator images taken by UAVs. The accuracy of the model reaches 96.3%($AP_{\mathrm{IoU=.50}}$) for bunch-drop defects on this part of data. Since in this area $AP_{\mathrm{IoU=.50}}$ is normally enough for applications, this performance is a competitive result. The precision curves which illustrate more detail are in Figure 11. Compared to the results on CPLID, the $AP_{\mathrm{IoU=.75}}$ and $AP_{IoU=.50:.05:.95}$ dropped to level of 23.4% and 39.6% which shows us that the real-world defects are harder to be detected in more granular metrics. Although the results of these two metrics are not very satisfying, we provide these results to offer other researchers a reference and we will continue to improve our work in the future.

5. Conclusion

In this article, an insulator defect detection method based on DETR is presented that combines transfer learning techniques and an improved loss function. Compared to conventional multistage methods, the proposed method shows the convenience and conciseness of end-to-end detection with comparable performance and the possibility of training an insulator defect detector with limited defect data. So far, the method has provided competitive results of detecting bunch-drop defects in composite insulators and glass insulators. It comes with new challenges of detecting smaller faults and more types of defects. The inference speed of our method is still slow compared to the models characterized by fast speed. In future studies, we will gather more insulator defect data with higher diversity and improve the generalization performance and inference speed of the method.

Appendix

Structure of DETR

```
DETR(
        (transformer): Transformer(
          (encoder): TransformerEncoder(
             (layers): ModuleList(
              (0): TransformerEncoderLayer(
                (self_attn): MultiheadAttention(
                 (out_proj): NonDynamicallyQuantizableLinear(
                        in_features=256, out_features=256, bias=True)
                )
                (linear1): Linear(in_features=256, out_features=2048, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
                (linear2): Linear(in_features=2048, out_features=256, bias=True)
                (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                (dropout1): Dropout(p=0.1, inplace=False)
                (dropout2): Dropout(p=0.1, inplace=False)
                      )
                     (1): TransformerEncoderLayer(
                      (self_attn): MultiheadAttention(
                        (out_proj): NonDynamicallyQuantizableLinear(
                              in_features=256, out_features=256, bias=True)
                      )
                      (linear1): Linear(in_features=256, out_features=2048, bias=True)
                      (dropout): Dropout(p=0.1, inplace=False)
                      (linear2): Linear(in_features=2048, out_features=256, bias=True)
                      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                      (dropout1): Dropout(p=0.1, inplace=False)
                      (dropout2): Dropout(p=0.1, inplace=False)
                    )
                    (2): TransformerEncoderLayer(
                      (self_attn): MultiheadAttention(
                        (out_proj): NonDynamicallyQuantizableLinear(
                              in_features=256, out_features=256, bias=True)
                      )
                      (linear1): Linear(in_features=256, out_features=2048, bias=True)
                      (dropout): Dropout(p=0.1, inplace=False)
                      (linear2): Linear(in_features=2048, out_features=256, bias=True)
                      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                      (dropout1): Dropout(p=0.1, inplace=False)
                      (dropout2): Dropout(p=0.1, inplace=False)
                      )
                     (3): TransformerEncoderLayer(
                      (self_attn): MultiheadAttention(
                        (out_proj): NonDynamicallyQuantizableLinear(
                              in_features=256, out_features=256, bias=True)
                      )
                      (linear1): Linear(in_features=256, out_features=2048, bias=True)
                      (dropout): Dropout(p=0.1, inplace=False)
                      (linear2): Linear(in_features=2048, out_features=256, bias=True)
                      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
                      (dropout1): Dropout(p=0.1, inplace=False)
                      (dropout2): Dropout(p=0.1, inplace=False)
```

```
            )
            (4): TransformerEncoderLayer(
              (self_attn): MultiheadAttention(
                (out_proj): NonDynamicallyQuantizableLinear(
                    in_features=256, out_features=256, bias=True)
              )
              (linear1): Linear(in_features=256, out_features=2048, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
              (linear2): Linear(in_features=2048, out_features=256, bias=True)
              (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
              (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
              (dropout1): Dropout(p=0.1, inplace=False)
              (dropout2): Dropout(p=0.1, inplace=False)
            )
            (5): TransformerEncoderLayer(
              (self_attn): MultiheadAttention(
                (out_proj): NonDynamicallyQuantizableLinear(
                    in_features=256, out_features=256, bias=True)
            )
             (linear1): Linear(in_features=256, out_features=2048, bias=True)
             (dropout): Dropout(p=0.1, inplace=False)
             (linear2): Linear(in_features=2048, out_features=256, bias=True)
             (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
             (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
             (dropout1): Dropout(p=0.1, inplace=False)
             (dropout2): Dropout(p=0.1, inplace=False)
            )
        )
)
            (decoder): TransformerDecoder(
            (layers):   ModuleList(
           (0): TransformerDecoderLayer(
            (self_attn):    MultiheadAttention(
            (out_proj):   NonDynamicallyQuantizableLinear(
       in_features=256, out_features=256, bias=True)
 )
 (multihead_attn):   MultiheadAttention(
   (out_proj): NonDynamicallyQuantizableLinear(
         in_features=256, out_features=256, bias=True)
 )
   (linear1): Linear(in_features=256, out_features=2048, bias=True)
   (dropout): Dropout(p=0.1, inplace=False)
   (linear2): Linear(in_features=2048, out_features=256, bias=True)
   (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
   (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
   (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
   (dropout1): Dropout(p=0.1, inplace=False)
   (dropout2): Dropout(p=0.1, inplace=False)
   (dropout3): Dropout(p=0.1, inplace=False)
)
            (1): TransformerDecoderLayer(
              (self_attn): MultiheadAttention(
                (out_proj): NonDynamicallyQuantizableLinear(
                    in_features=256, out_features=256, bias=True)
              )
              (multihead_attn): MultiheadAttention(
                (out_proj): NonDynamicallyQuantizableLinear(
                     in_features=256, out_features=256, bias=True)
                )
```

```
      (linear1): Linear(in_features=256, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=2048, out_features=256, bias=True)
      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (dropout3): Dropout(p=0.1, inplace=False)
    )
    (2): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True)
      )
      (multihead_attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True)
      )
      (linear1): Linear(in_features=256, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=2048, out_features=256, bias=True)
      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (dropout3): Dropout(p=0.1, inplace=False)
  )
    (3): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True)
      )
      (multihead_attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True)
      )
      (linear1): Linear(in_features=256, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=2048, out_features=256, bias=True)
      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (dropout3): Dropout(p=0.1, inplace=False)
  )
    (4): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True)
      )
      (multihead_attn): MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
          in_features=256, out_features=256, bias=True)
      )
      (linear1): Linear(in_features=256, out_features=2048, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
```

```
        (linear2): Linear(in_features=2048, out_features=256, bias=True)
        (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
        )
      (5): TransformerDecoderLayer(
        (self_attn):  MultiheadAttention(
        (out_proj): NonDynamicallyQuantizableLinear(
              in_features=256, out_features=256, bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(
              in_features=256, out_features=256, bias=True)
        )
        (linear1): Linear(in_features=256, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=256, bias=True)
        (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
        )
      )
      (norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    )
  )
  (class_embed):  Linear(in_features=256, out_features=2, bias=True)
  (bbox_embed): MLP(
    (layers):  ModuleList(
      (0): Linear(in_features=256, out_features=256, bias=True)
      (1): Linear(in_features=256, out_features=256, bias=True)
      (2): Linear(in_features=256, out_features=4, bias=True)
    )
  )
```

```
(query_embed): Embedding(100, 256)
(input_proj): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
(backbone): Joiner(
        (0): Backbone(
          (body): IntermediateLayerGetter(
            (conv1): Conv2d(
              3,    64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
            (bn1): FrozenBatchNorm2d()
            (relu): ReLU(inplace=True)
            (maxpool): MaxPool2d(
                kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
            (layer1): Sequential(
              (0): Bottleneck(
                (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn1): FrozenBatchNorm2d()
              (conv2): Conv2d(
                    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
              (bn2): FrozenBatchNorm2d()
              (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn3): FrozenBatchNorm2d()
              (relu): ReLU(inplace=True)
              (downsample): Sequential(
                (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (1): FrozenBatchNorm2d()
              )
            )
            (1): Bottleneck(
              (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn1): FrozenBatchNorm2d()
              (conv2): Conv2d(
                    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
              (bn2): FrozenBatchNorm2d()
              (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn3): FrozenBatchNorm2d()
              (relu): ReLU(inplace=True)
            )
            (2): Bottleneck(
              (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn1): FrozenBatchNorm2d()
              (conv2): Conv2d(
                    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
              (bn2): FrozenBatchNorm2d()
              (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn3): FrozenBatchNorm2d()
              (relu): ReLU(inplace=True)
            )
            (1): Bottleneck(
              (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn1): FrozenBatchNorm2d()
              (conv2): Conv2d(
                    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
              (bn2): FrozenBatchNorm2d()
              (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (bn3): FrozenBatchNorm2d()
              (relu): ReLU(inplace=True)
            )
          )
          (layer2): Sequential(
            (0): Bottleneck(
```

```
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
         (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
         (1): FrozenBatchNorm2d()
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
         (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
         (1): FrozenBatchNorm2d()
      )
    )
    (1): Bottleneck(
         (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
         (bn1): FrozenBatchNorm2d()
```

```
    (conv2): Conv2d(
          256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): FrozenBatchNorm2d()
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): FrozenBatchNorm2d()
    (relu): ReLU(inplace=True)
)
(2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): FrozenBatchNorm2d()
    (conv2): Conv2d(
          256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): FrozenBatchNorm2d()
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): FrozenBatchNorm2d()
    (relu): ReLU(inplace=True)
)
    (3): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): FrozenBatchNorm2d()
    (conv2): Conv2d(
          256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): FrozenBatchNorm2d()
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): FrozenBatchNorm2d()
    (relu): ReLU(inplace=True)
)
(4): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): FrozenBatchNorm2d()
    (conv2): Conv2d(
          256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): FrozenBatchNorm2d()
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
    )
)
(layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): FrozenBatchNorm2d()
      (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): FrozenBatchNorm2d()
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): FrozenBatchNorm2d()
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
```

```
                    (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
                    (1): FrozenBatchNorm2d()
                  )
                )
                (1): Bottleneck(
                  (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (bn1): FrozenBatchNorm2d()
                  (conv2): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
                  (bn2): FrozenBatchNorm2d()
                  (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (bn3): FrozenBatchNorm2d()
                  (relu): ReLU(inplace=True)
                      )
                (2): Bottleneck(
                  (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (bn1): FrozenBatchNorm2d()
                  (conv2): Conv2d(
                      512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
                  (bn2): FrozenBatchNorm2d()
                  (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (bn3): FrozenBatchNorm2d()
                  (relu): ReLU(inplace=True)
                )
              )
            )
          )
          (1): PositionEmbeddingSine()
        )
      )
```

Data Availability

Previously reported insulator defect data were used to support this study and are available at https://github.com/InsulatorData/InsulatorDataSet. These prior studies (and data sets) are cited at relevant places within the text as references [12].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] W. Li, G. Ye, F. Huang, S. Wang, and W. Chang, "Recognition of insulator based on developed MPEG-7 texture feature," in *2010 3rd International Congress on Image and Signal Processing*, pp. 265–268, Yantai, China, 2010.

[2] X. N. Huang and Z. L. Zhang, "A method to extract insulator image from aerial image of helicopter patrol," *Power System Technology*, vol. 34, no. 1, pp. 194–197, 2010.

[3] Q. Wu, J. An, and B. Lin, "A texture segmentation algorithm based on PCA and global minimization active contour model for aerial insulator images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 5, pp. 1509–1518, 2012.

[4] L. Khalayli, H. Al Sagban, H. Shoman, K. Assaleh, and A. El-Hag, "Automatic inspection of outdoor insulators using image processing and intelligent techniques," in *2013 IEEE Electrical Insulation Conference (EIC)*, pp. 206–209, Ottawa, ON, Canada, 2013.

[5] U. Iruansi, J. R. Tapamo, and I. E. Davidson, "An active contour approach to insulator segmentation," in *AFRICON 2015*, pp. 1–5, Addis Ababa, Ethiopia, 2015.

[6] S. Liao and J. An, "A robust insulator detection algorithm based on local features and spatial orders for aerial images," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 5, pp. 963–967, 2014.

[7] Y. Zhai, D. Wang, M. Zhang, J. Wang, and F. Guo, "Fault detection of insulator based on saliency and adaptive morphology," *Multimedia Tools and Applications*, vol. 76, no. 9, pp. 12051–12064, 2017.

[8] Y. Zhai, R. Chen, Q. Yang, X. Li, and Z. Zhao, "Insulator fault detection based on spatial morphological features of aerial images," *IEEE Access*, vol. 6, pp. 35316–35326, 2018.

[9] J. Chen, X. Xin, and H. Dang, "Fault detection of insulators using second-order fully convolutional network model," *Mathematical Problems in Engineering*, vol. 2019, Article ID 6397905, 10 pages, 2019.

[10] Z. Gao, G. Yang, E. Li et al., "Insulator segmentation for power line inspection based on modified conditional generative adversarial network," *Journal of Sensors*, vol. 2019, Article ID 4245329, 8 pages, 2019.

[11] D. Sadykova, D. Pernebayeva, M. Bagheri, and A. James, "IN-YOLO: real-time detection of outdoor high voltage insulators

using UAV imaging," *IEEE Transactions on Power Delivery*, vol. 35, no. 3, pp. 1599–1601, 2019.

[12] X. Tao, D. Zhang, Z. Wang, X. Liu, H. Zhang, and D. Xu, "Detection of power line insulator defects using aerial images analyzed with convolutional neural networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 4, pp. 1486–1498, 2018.

[13] W. Zhao, M. Xu, X. Cheng, and Z. Zhao, "An insulator in transmission lines recognition and fault detection model based on improved faster RCNN," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–8, 2021.

[14] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*, A. Vedaldi, H. Bischof, T. Brox, and J. M. Frahm, Eds., vol. 12346 of Lecture Notes in Computer Science, , pp. 213–229, Springer, Cham, 2020.

[15] A. Vaswani, N. Shazeer, N. Parmar et al., "Attention is all you need," *Advances in neural information processing systems*, vol. 30, article 4245329, 2017.

[16] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," in *Proceedings of the Third Conference on Machine Translation*, pp. 1–9, Brussels, Belgium, 2018.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov et al., "An image is worth 16x16 words: transformers for image recognition at scale," https://arxiv.org/abs/2010.11929.

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," https://arxiv.org/abs/1607.06450.

[19] R. A. Horn, "The hadamard product," *Proceedings of Symposia in Applied Mathematics*, vol. 40, pp. 87–169, 1990.

[20] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[21] L. Liu, X. Liu, J. Gao, W. Chen, and J. Han, "Understanding the difficulty of training transformers," https://arxiv.org/abs/2004.08249.

[22] M. Popel and O. Bojar, "Training tips for the transformer model," https://arxiv.org/abs/1804.00247.

[23] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[24] J. Deng, Z. Zhang, E. Marchi, and B. Schuller, "Sparse autoencoder-based feature transfer learning for speech emotion recognition," in *2013 humaine association conference on affective computing and intelligent interaction*, pp. 511–516, Geneva, Switzerland, 2013.

[25] W.-L. Zheng and L. Bao-Liang, "Personalizing EEG-based affective models with transfer learning," in *Proceedings of the twenty-fifth international joint conference on artificial intelligence*, pp. 2732–2738, New York, New York, USA, 2016.

[26] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[27] T. Y. Lin, M. Maire, S. Belongie et al., "Microsoft coco: common objects in context," in *European conference on computer vision*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693 of Lecture Notes in Computer Science, , pp. 740–755, Springer, Cham, 2014.

[28] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, Venice, Italy, 2017.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, Las Vegas, NV, USA, 2016.

[30] S. Tuli, I. Dasgupta, E. Grant, and T. L. Griffiths, "Are Convolutional Neural Networks or Transformers more like human vision?," https://arxiv.org/abs/2105.07197.

[31] M. M. Naseer, K. Ranasinghe, S. H. Khan, M. Hayat, F. Shahbaz Khan, and M. H. Yang, "Intriguing properties of vision transformers," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23296–23308, 2021.

[32] C. Liu, Y. Wu, J. Liu, Z. Sun, and H. Xu, "Insulator faults detection in aerial images from high-voltage transmission lines based on deep learning model," *Applied Sciences*, vol. 11, no. 10, p. 4647, 2021.

[33] J. Liu, C. Liu, Y. Wu, H. Xu, and Z. Sun, "An improved method based on deep learning for insulator fault detection in diverse aerial images," *Energies*, vol. 14, no. 14, p. 4365, 2021.

[34] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[35] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," https://arxiv.org/abs/1711.05101.

[36] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: a metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 658–666, Long Beach, CA, USA, 2019.

[37] G. Jocher, A. Chaurasia, A. Stoken et al., *Ultralytics/yolov5: v6.1- TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*, Zenodo, 2022.

[38] C. Feng, Y. Zhong, Y. Gao, M. R. Scott, and W. Huang, "TOOD: task-aligned one-stage object detection," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*., pp. 3510–3519, Montreal, QC, Canada, 2021.

[39] P. Sun, R. Zhang, Y. Jiang et al., "Sparse r-cnn: end-to-end object detection with learnable proposals," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14454–14463, Nashville, TN, USA, 2021.