Received 24 June 2022, accepted 18 August 2022, date of publication 26 August 2022, date of current version 8 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3202222

RESEARCH ARTICLE

Clickable Object Detection Network for a Wide Range of Mobile Screen Resolutions

BOSEON KANG[1], MINSEOK JO[2], AND CHANG-SUNG JEONG[3], (Member, IEEE)
[1]Visual Information Processing Department, Korea University, Seoul 02841, South Korea
[2]Front Image Recognition Logic Cell, Hyundai Mobis, Yongin-si, Gyeonggi-do 16891, South Korea
[3]Department of Electrical Engineering, Korea University, Seoul 02841, South Korea

Corresponding author: Chang-Sung Jeong (csjeong@korea.ac.kr)

This work was supported in part by the Artificial Intelligence Industrial Convergence Cluster Development Project Funded by the Ministry of Science and ICT (MSIT), South Korea; and in part by the Gwangju Metropolitan City and the Brain Korea 21 FOUR Project, in 2021.

ABSTRACT Recently, as the development cycle of applications has been shortened, it is important to develop rapid and accurate application testing technology. Since application testing requires a lot of cost, mobile GUI component detection technology using deep learning is essential to prevent the use of expensive human resources. In this paper, we shall propose a Clickable Object Detection Network (CODNet) for mobile component detection in a wide range of mobile screen resolutions. CODNet consists of three modules: feature extraction, deconvolution and prediction modules in order to provide performance improvement and scalability. Feature extraction module uses squeeze and excitation blocks to efficiently extract features by changing the ratio of the input image to 1:2 most close to that of mobile screen. Deconvolution module provides feature map of various sizes by upsampling feature map through top-down pathway and lateral connections. Prediction module selects an anchor size most suitable for the mobile environment using Anchor Transfer block among the set of anchor candidates obtained through the analysis of mobile dataset. Moreover, we shall show that our model achieves competitive performance in mean average precision on our dataset compared to the other models, and object detection performance is improved by building a new mobile screen dataset which consists of data collected from various resolutions and operating systems.

INDEX TERMS Object detection, computer vision, mobile screen, deep neural networks.

I. INTRODUCTION

The development and release of applications have been greatly shortened as the cycle of technology development has accelerated, it becomes essential to find a methodology for accurately and quickly testing applications. Application test is to check if all the functions in the application are working normally by clicking the clickable object. It requires a lot of time, money and human resources to verify all the functions in the application. In order to prevent the use of expensive human resources and time, it is important to develop a deep learning model for detecting mobile GUI component during application testing [1], [2], [3], [4], [5]. Several deep learning methods have been proposed for more accurate component analysis. Redraw [6] proposed accurate prototyping of GUIs through three phases: detection, classification, and assembly. In the classification stage, AlexNet [7] based CNN [8] is used to classify the detected GUI components into an appropriate class. They classified Android GUI components into 15 classes through Redraw. Zhang *et al.* [9] created iOS APP screen dataset which consists of 77,637 screen data from 4,068 iPhone applications to detect GUI components in the iOS environment. They used SSD model [10] with MobileNetV1 [11] as the backbone to run the detection model on-device. Zhu *et al.* [12] used widget recognition technology based on image matching for detecting GUI components. They improved detection accuracy by predicting the intent of the components on deep learning model which combines VGG-16 model [13] and short-term memory (LSTM) [14] for extracting features from images and text processing respectively.

However, the previous researches have several problems which make it difficult to detect mobile GUI components in mobile environment on deep learning models as follows: even

The associate editor coordinating the review of this manuscript and approving it for publication was Shovan Barma.

though the height is longer than the width in the mobile screen while the contrary occurs usually in the clickable objects, the ratio between the width and height for the image is resized to the fixed ratio on deep learning model, resulting in information loss. In addition, they did not efficiently deal with clickable objects with various aspect ratio, since the anchor aspect ratios is fixed. Moreover, the previous researches trained deep learning model using dataset for one operating system, and hence there does not exist sufficient dataset which can be used for the detection model on various operating systems. In this paper, we shall present Clickable Object Detection Network (CODNet) for efficiently detecting clickable objects in a wide range of mobile screen resolutions. Also, we build a new mobile screen dataset which consists of data collected from various resolutions and operating systems, achieves competitive performance in mean average precision on our dataset compared to the other models. Existing deep learning models are difficult to accurately detect objects due to the characteristics of the mobile device, where the height is longer than the width. Our model not only solves the ratio gap problem by setting the aspect ratio of the input image to that most close to the mobile screen, but also extends the input resolution to that of the mobile screen in order to minimize the loss of feature. We generate a set of aspect ratio candidates suitable for the mobile environments through mobile dataset analysis. Among them, the most proper anchor size is selected for clickable objects with various aspect ratios in Prediction module. As a result, the model can flexibly adapt to objects of different sizes, and cover a wide range of resolutions from Video Graphics Array (VGA) to Ultra High Definition (UHD). We build datasets by grouping clickable objects essential for testing application. We collect data from as many mobile devices as possible, such as Android, iOS, and car navigation, in order to detect clickable objects in various mobile environments. Therefore, our model is OS independent regardless of Android, iOS and so on.

The contributions of our study are summarized as follows:

1) We propose a new deep learning model which provides performance improvement and scalability by designing a deep learning network which consists of three modules: Feature extraction module, Deconvolution module and Prediction module.
2) We demonstrate that the accuracy can be improved by changing the resolution of the input image to the aspect ratio closest to the mobile screen.
3) We propose a new scheme for finding the most proper anchor aspect ratio in a mobile environment for efficiently dealing with clickable objects with various aspect ratios.
4) We build a new mobile dataset which can be used in a variety of mobile environments by collecting data from various resolutions and operating systems.
5) We demonstrate that our model achieves significant performance improvements over previous deep learning models.

The remainder of this paper comprises four sections. In section 2, we introduce related works. In section 3, we present the overall architecture for our model. In section 4, we describe about the experimental results, and finally in section 5 concludes with our results.

II. RELATED WORK

A. OBJECT DETECTION METHOD

Object detection method is largely divided into multi-stage method and one-stage method. In the multi-stage method, region proposal for finding the location of an object and classification for finding the type of object are sequentially performed. Among the multi-stage method, we deal with Faster R-CNN [15], Cascade R-CNN [16] and DetectoRS [17] which demonstrate excellent performance using CNN. On the other hand, in the case of the one-stage method, region proposal and classification are performed simultaneously. We deal with YOLO series, SSD, FASF [18], FCOS [19], CornerNet [20] and CentripetalNet [21] each of which demonstrates excellent performance among one-stage method.

Faster R-CNN is a model which improves the slow speed of R-CNN [22]. In R-CNN, the step which takes the longest is the selective search, since the selective search is computed in the CPU. To solve this problem, Faster R-CNN introduces Region Proposal Network (RPN) instead of selective search to enable computation on GPU. RPN receives the feature map from the previous CNN as input. A 256-dimension vector is obtained using a sliding window on the received feature map. At this time, the anchor to be used as the window is set in advance. In Faster R-CNN, 9 anchors with various width, height, ratio and size are used. By using 256-dimension vector obtained in this way the class and location are calculated through two layers.

Cascade R-CNN proposes a method for solving two problems which occur with the increase of Intersection over Union (IoU) thresholds. As the IoU thresholds increase, the positive samples disappear exponentially, resulting in overfitting during training. In addition, if the IoU set during training and the IoU set at inference are inconsistent, the accuracy will be lowered. Cascade R-CNN consists of a sequence of detectors with different IoU thresholds set. Detectors are connected sequentially, and use the output from the previous step as input for the next step. As a result, all detectors have the positive set of examples of equivalent size to solve the overfitting problem. It shows that performance is improved through the process of gradual training using the proposals of the learned detectors at low IoU.

DetectoRS proposes a backbone design utilizing a see and think mechanism. At the macro level, they propose a method for building Recursive Feature Pyramid using additional feedback connections to the existing Feature Pyramid Network (FPN) [23]. At the micro level, they propose Switchable Atrous Convolution which collects features extracted at various atrous rates through a switch function.

YOLO [24], a real-time object detector proposed by Redmon Joseph al., is created for high-speed purposes. R-CNN, a two-stage model, detects one image by dividing it into two stages. In the region proposal step, classification is performed 2000 times, since 2000 candidate areas are extracted per image. On the other hand, YOLO finds the location and class through a single network pass. YOLO achieves 45 FPS, which is significantly faster than 5 FPS when using Faster R-CNN. YOLO divides the input image into S $\times$ S grids. The grid cell closest to the center of the object is responsible for detecting the object. Each grid cell predicts B bounding boxes, confidence scores of the corresponding bounding boxes and C conditional class probabilities. The dimension of the final feature map is S $\times$ S $\times$ (B $\times$ 5 + C). YOLO has a limitation in which the performance is slightly lowered due to the small difference in IoU values for small objects. Since then, improvement models such as YOLOf [25], YOLOx [26], YOLOv4 [27], and YOLOv5 [28] are proposed to solve the limitations.

YOLO improves its speed by using the one-stage structure, but its accuracy is relatively low, since it is necessary to find objects of various sizes in one image. SSD proposed by Wei Liu *et al.* improves the accuracy by solving the problem of finding objects of various sizes in one image through a method of performing object detection across multiple layers. YOLO divides the image into S $\times$ S grids, whereas SSD efficiently detects both large and small objects by dividing the image into grids of various sizes across 6 feature maps. The SSD is constructed using 6 additional convolution layers on the 5th convolution layer of VGG-16.

FSAF is a single-shot object detector based on RetinaNet [29]. They improved the performance by solving the problems of the existing anchor-based model by using multi-level anchor-free branches. During training, anchor-free branches attached to each level of the FPN select the most appropriate feature level for training. The selected feature level effectively represents the instance. As a result of training, the model outperforms existing one-stage detectors in detecting small objects.

FCOS is a one-stage object detector which detects objects in a per-pixel prediction fashion. FCOS is anchor box free, and completely eliminates the complex calculations associated with anchor boxes such as calculating overlapping during training. In addition, they use centerness to suppress many bounding boxes generated at locations far from the center of an object. Centerness reduces the influence of predicted values at locations far from the center of the object using the center of the bounding box, left-top and right-bottom corner pairs.

CornerNet predicts object bounding boxes by using a pair of keypoints instead of anchors. Keypoints are the top-left and top-right corners of the target object. Keypoint estimation is based on feature points, so anchor boxes are not used. Centripetalnet is a keypoint-based detector uses centripetal shift to create a pair of corner keypoints from the same object.

B. MOBILE SCREEN IMAGE DATASET

Rico [30] dataset consists of 27 classes, and contains 9.7k data on Android app design. Rico was created to support five classes of data-driven applications: design search, UI layout generation, UI code generation, user interaction modeling and user perception prediction. It provides visual, textual, structural, and interactive design properties for more than 72k unique UI screens. Chen *et al.* [31] build a mobile screen dataset to solve the label missing problem according to the analysis results of 10,408 Android apps. The dataset consists of 15 classes and a total of 52,524 GUI images. They used App Explorer to collect GUIs from 15,087 applications. The collected dataset consists of 13,145 screenshots containing ImageView and ImageButton from 7,594 apps excluding duplicate screens. ICONINTENT [32] is an app analysis framework which combines program analysis with icon classification to identify UI widgets in Android apps. They download each category icon using Google Image Search, and manually label the icon. As a result, they collect 1576 icons for 8 classes. Redraw [6] dataset consists of 15 classes and 191,300 components, but the number of components in the classes are unbalanced. In case of textview, there are 99,200 components, but in case of spinner, there are only 20 components. To address data imbalances, they create a mobile screen by generating randomly sample components until there are at least 5000 components.

III. MODEL ARCHITECTURE

In this section, we shall describe about the architecture of our model for detecting clickable objects in the mobile environment. Our model consists of three modules: Feature extraction, Deconvolution and Prediction module. Feature extraction module is used as the backbone in the model. It uses Squeeze-and-excitation Networks(Squeeze Net) [33], which adds a Squeeze-and-excitation block to the ResNet [34]. High-quality features are extracted with little computation during feature extraction process compared to ResNet. In each step of feature extraction module, the extracted features are sent to Prediction module and Deconvolution module. Input resolution of most existing deep learning models does not greatly deviate from the aspect ratio of 1:1. However, in the case of mobile screens which our model is trying to detect, the height of the screen image is longer than the width. In order to solve the problem arising from the ratio gap of the mobile screen, the ratio of the input image in feature extraction module is changed to 1:2 most close to that of mobile screen. Deconvolution module connects feature extraction module and prediction module. It produces different feature maps with various size at each layer through decovolution, and sends the feature maps to five prediction modules. Prediction module derives the final result. Five prediction modules attached to Deconvolution module and one prediction module attached to feature extraction module are used for object detection. Since the six prediction modules are based on feature maps of different sizes, it can effectively

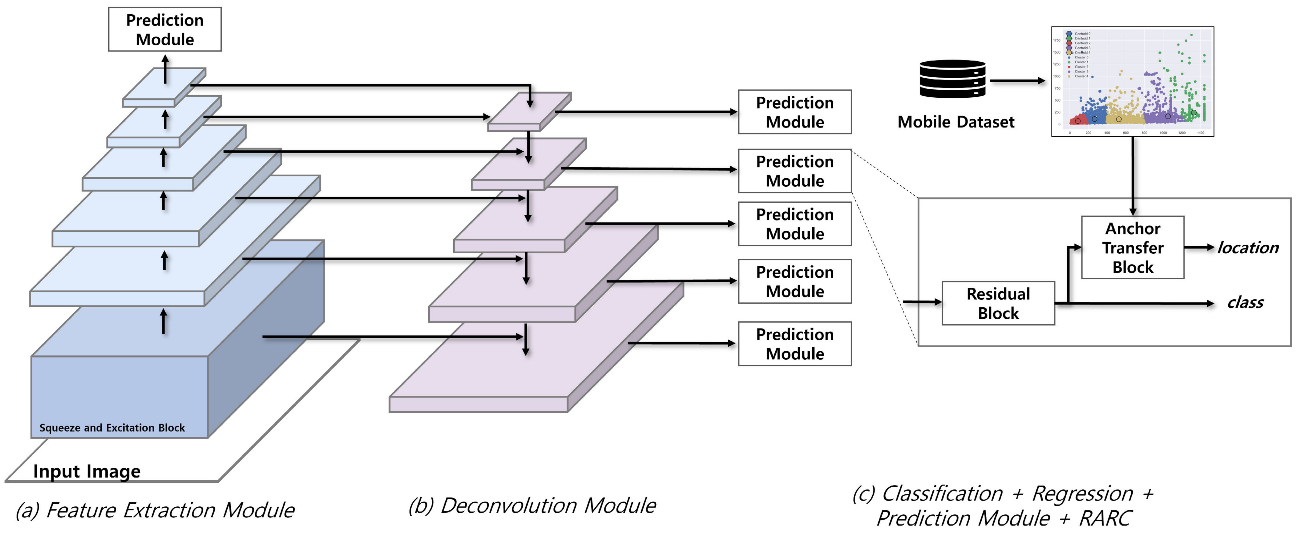


FIGURE 1. **The network architecture of CODNet.**

find objects of various sizes. In most cases, the width of a clickable object on a mobile screen is longer than its height. Especially, such objects as text, navigation bar and status bar have much longer width with an aspect ratio of up to 17:1. The anchor aspect ratios used in the previous papers do not efficiently deal with clickable objects with such aspect ratios of longer width. In this paper, we shall propose a new scheme which finds, for each of clickable objects with various aspect ratios, the most proper anchor size in Prediction module by generating a set of candidates for aspect ratio suitable for the mobile environment through the analysis of mobile dataset and then mapping it onto one of them. It can also adapt to the various size of clickable objects on a mobile screen in a flexible way.

The following subsections detail the modules used in the model.

A. FEATURE EXTRACTION MODULE

Feature extraction module is a backbone of CODNet. It uses the Squeeze and Excitation block to extract features as shown in Figure 2. Squeeze and Excitation block performs feature recalibration using Squeeze and Excitation operation without significantly increasing model complexity and computational time. The squeeze operation compresses global spatial information through Global Average Pooling (GAP) [35]. The excitation operation is responsible for recalibrating the compressed information. The recalibrating process is simply calculated through two Fully Connected layer(FC) [36] and Rectified Linear Unit (ReLU) [37], and node dependencies are also calculated. In FC, the input layer consists of C nodes, and the nodes in the middle layer are reduced by the reduction ratio r. Output layer consists of C nodes. We set the reduction ratio r = 16. Unlike other models, CODNet focuses on the input resolution most close to that of the mobile screen in order to reflect the characteristics of the mobile environment.

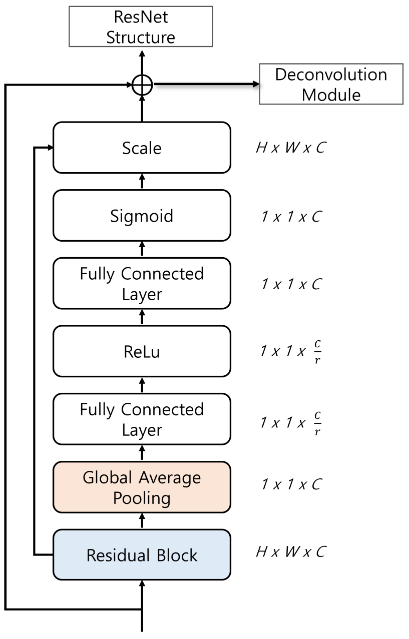


FIGURE 2. **The Squeeze and excitation block of feature extraction module.**

Most mobile screen resolutions have an aspect ratio of 1:2. However, since the aspect ratio of the input resolution of the existing deep learning model is 1:1, information loss occurs due to the ratio gap. Feature extraction module changes the input resolution to 512 $\times$ 1024 or 1080 $\times$ 1920 to solve the problem arising by the ratio gap.

B. DECONVOLUTION MODULE

Deconvolution module is a top-down pathway with lateral connections. The top-down pathway of Deconvolution module generates a high-resolution feature map by upsampling. Each lateral connection merges the feature maps of the

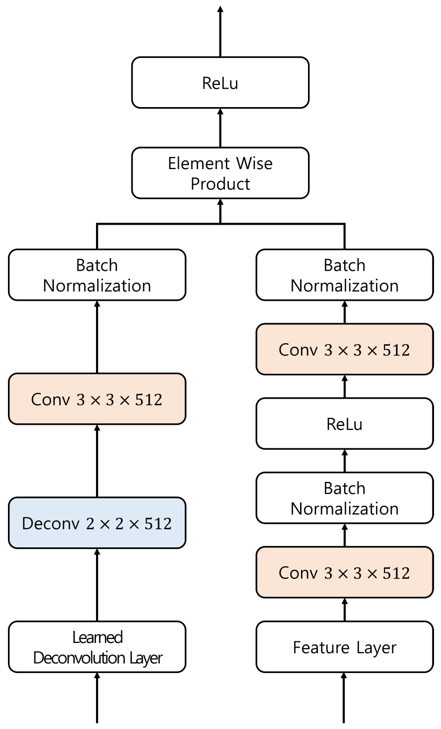


FIGURE 3. Deconvolution module.

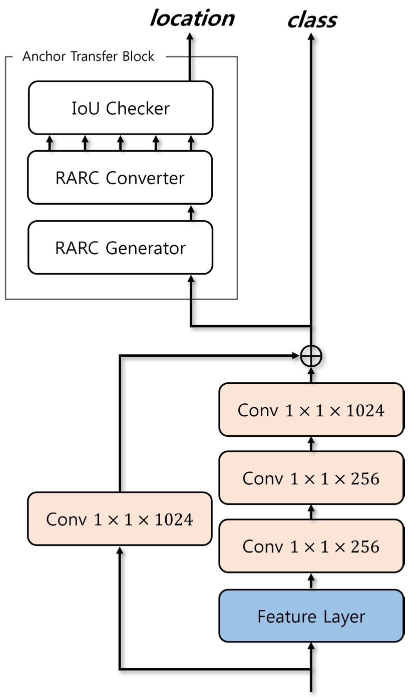


FIGURE 4. Prediction module.

same size from Feature extraction module and Deconvolution module using element-wise product. By using both feature maps of two different layers, it is possible to efficiently extract high-context information, thus improving accuracy. In Deconvolution module, we reduce the inference time by using a learned deconvolution layer rather than a commonly performed bilinear upsampling layer. Batch normalization is performed after each convolution operation. Its structure is shown in Figure 3.

C. PREDICTION MODULE

A total of six prediction modules attached to feature extraction and deconvolution modules are used for object detection in each image, improving detection accuracy by working on the feature maps of various sizes. Prediction module is designed so that it can improve the overall performance by not only exploiting the residual block in ResNet, but also especially creating anchor transfer block as shown in Figure 4.

Anchor Transfer block consists of three units: RARC (Representative Anchor aspect Ratio Candidates) Generator, RARC Converter and IoU Checker. RARC Generator produces a set of candidates for representative aspect ratios suitable for the mobile environment through k-means clustering from mobile dataset. RARC Converter changes the anchor aspect ratio in the predicted region into each of RARC for the mobile environment. IoU Checker finds, for each converted candidate in RARC, its IoU score with respect to ground truth bounding box, and then generates location by using anchor information of the candidate with the highest IoU score. RARC is obtained by RARC generator as follows: First, we extract the bounding boxes for all the objects in the mobile dataset. Let the width and height of the bounding box be x and y respectively. We plot each bounding box (x, y) onto two-dimensional plane. The plotted bounding boxes are clustered through the k-means clustering algorithm, and then RARC is generated by calculating the centroids of each cluster. Table 1 shows RARC generated from our mobile dataset for different k.

The class output is used to predict the class of the object included in the image through the Cross-Entropy loss function [38]. Let p be the predicted probability of our model for the object region, and t the true probability of the object region respectively. Then, the class loss function is defined as follows:

$$L_{cls} = -\sum_{i=1}^{n} t(i) \log p(i) \tag{1}$$

The location output is used to predict the regression of objects included in the image through the Box regression loss function [39]. Let $p^k = (p_x^k, p_y^k, p_w^k, p_h^k)$ be a tuple which consists of predicted bounding box information. Let $g = (g_x, g_y, g_w, g_h)$ be a tuple which represents a ground truth for class k. Then, the box regression loss function is defined as follows:

$$L_{loc}(p^k, g) = \sum_{i \in \{x,y,w,h\}} \mathrm{Smooth}_{L_1}(p_i^k - g_i) \tag{2}$$

$$\mathrm{Smooth}_{L_1}(x) = \begin{cases} \dfrac{x^2}{2} & \text{if } |x| < 1 \\ |x| - \dfrac{1}{2} & \text{otherwise} \end{cases} \tag{3}$$

TABLE 1. Average aspect ratio of each cluster.

Number of Clusters	1	2	3	4	5	6	7	8
Ratios 1	1.43:1	3.47:1	0.93:1	2.08:1	2:1	1.14:1	0.91:1	1.1:1
Ratios 2	-	15.48:1	7.68:1	4.78:1	2.16:1	2.5:1	1.23:1	1.3:1
Ratios 3	-	-	15:1	10.41:1	6.25:1	2.6:1	2.9:1	1.98:1
Ratios 4	-	-	-	14.81:1	10.39:1	5.9:1	6.4:1	3.78:1
Ratios 5	-	-	-	-	14.52:1	7.19:1	7.11:1	6.7:1
Ratios 6	-	-	-	-	-	14.2:1	12.8:1	7.11:1
Ratios 7	-	-	-	-	-	-	14.42:1	8.6:1
Ratios 8	-	-	-	-	-	-	-	14.4:1

TABLE 2. Annotation data for each category and class.

Category	Text	Image	Region	Button	Edit Text	Status Bar	Navigation Bar
Train	6859	6270	2909	1786	871	993	580
Validation	774	678	308	171	103	113	66
Test	829	757	346	208	119	122	75
Total	8462	7705	3563	2165	1093	1228	721

TABLE 3. Top-1 and Top-5 error rates for CODNet-SE and CODNet-W.

Error rate		ResNet-50	ResNet-101	ResNet-152
Top-1 error	CODNet-SE	23.29	22.38	21.57
	CODNet-W	24.7	23.6	23
Top-5 error	CODNet-SE	6.62	6.07	5.73
	CODNet-W	7.8	7.1	6.7

The final loss value of our model is defined by the average of the above two loss functions.

$$L = \frac{1}{2}(L_{cls} + L_{loc}) \tag{4}$$

IV. EXPERIMENTS

In this section, we show describe about the experimental results for our model CODNet. Our experimental environment consists of one CPU with AMD EPYC Processor with 92 single-cores, GPU with 8 units each NVIDIA A100 40GB and 1.7TB RAM.

This section is organized as follows: Section 4.1 describes in detail the dataset used in the experiments. Section 4.2 shows the performance of each module. Section 4.3 analyzes the loss value and mAP for each input resolution. Section 4.4 analyzes the mAP for each aspect ratio of anchor. Section 4.5 shows the performance of CODNet for each class on our dataset. Finally, section 4.6 compares the performance of our model over the existing deep learning models.

A. DATASET

We build our mobile screen dataset by focusing on clickable objects. It comprises a total of 1,261 images with a wide range of resolutions and 24,937 annotation data. The dataset is classified into three groups: training, validation and testing. The training set is about 80% of the entire dataset, and each of the validation and test set is about 10% of the entire dataset. Clickable objects are classified into 7 classes: Text, Image, Button, Region, Status bar, Navigation bar and Edit text. Our dataset format follows VOC [40].

TABLE 4. mAP of CODNet for the number of clusters k.

-	k	ResNet-50	ResNet-101
CODNet-512	**1**	0.633	0.689
	2	0.417	0.502
	3	0.709	0.761
	4	0.525	0.6
	5	**0.734**	**0.818**
	6	0.68	0.758
	7	0.71	0.764
	8	0.713	0.76
CODNet-1080	**1**	0.617	0.653
	2	0.493	0.557
	3	0.72	0.79
	4	0.65	0.724
	5	**0.787**	**0.865**
	6	0.716	0.812
	7	0.723	0.798
	8	0.727	0.791

Table 2 shows the number of annotation for various classes of each category. Text class has the maximum number of 8,462 data, since it appears most frequently on the screen, Image class has the second largest number of 7,705 data. Region class is a component with clear boundary which contains Text or Image objects. Edit test class is a text area which can be edited. Status bar is a status window at the top of the mobile screen. Navigation bar is a navigation area with system buttons at the bottom. As a data augmentation method for preventing overfitting, we randomly change color, saturation and brightness, and use vertical and horizontal flips instead of expansion augmentation trick.

B. MODULE PERFORMANCE

1) FEATURE EXTRACTION MODULE

We investigate the performance of Squeeze-and-Excitation Block (SEBlock) attached to Feature extraction module

TABLE 5. Comparison between existing deep learning models.

Model	Bockbone	mAP	mAPs	mAPm	mAPl
SSD300	VGG-16	0.407	0.367	0.4	0.399
SSD512	VGG-16	0.438	0.444	0.435	0.433
Faster R-CNN	ResNet-50	0.682	0.494	0.538	0.687
Faster R-CNN	ResNet-101	0.702	0.514	0.551	0.684
Cascade-RCNN	ResNet-50	0.711	0.51	0.557	0.724
Cascade-RCNN	ResNet-101	0.748	0.549	0.581	0.735
DetectoRS	ResNet-50	0.73	0.52	0.532	0.75
DetectoRS	ResNet-101	0.784	0.551	0.607	0.793
FCOS	ResNet-50	0.513	0.45	0.438	0.498
FCOS	ResNet-101	0.554	0.442	0.508	0.549
FSAF	ResNet-50	0.61	0.499	0.487	0.611
FSAF	ResNet-101	0.657	0.495	0.508	0.659
CornerNet	HourGlass-104	0.538	0.546	0.527	0.528
CentripetalNet	HourGlass-104	0.516	0.554	0.602	0.515
YOLOv3	DarkNet-53	0.565	0.432	0.533	0.571
YOLOv4	CSP-Darkent53	0.701	0.494	0.532	0.736
YOLOv5	Modified CSP v5	0.742	0.558	0.576	0.801
YOLOF	ResNet-50	0.51	0.416	0.458	0.518
YOLOX-X	Modified CSP v5	0.721	0.478	0.572	0.727
CODNet-512	ResNet-50	0.734	0.53	0.574	0.751
CODNet-1080	ResNet-50	0.787	0.59	0.67	0.823
CODNet-512	ResNet-101	0.818	0.588	0.698	0.835
CODNet-1080	ResNet-101	**0.865**	**0.631**	**0.729**	**0.889**

by comparing the performance of the model for two cases: CODNet-SE with SEBlock and CODNet-W without SEBlock respectively. Experiments are conducted for top-1 and top-5 error by changing the size of three backbone, ResNet-50, ResNet-101 and ResNet-151. As shown in Table 3, the top-1 and top-5 error rates are become lower when using Squeeze-and-Excitation Block, and ResNet-151 has the smallest top-1 and top-5 errors.

2) DECONVOLUTION MODULE

We check the performance of Deconvolution module by comparing the performance of the model for two cases: CODNet with and without Deconvolution module respectively. Experiments are conducted for mAP. CODNet without Deconvolution module has a mAP 77.1. CODNet with Deconvolution module has a mAP of 86.5, which is 9.4 higher than that of CODNet without Deconvolution module.

C. MODEL PERFORMANCE

We investigate the performance of our model in terms of input resolution and aspect ratio on two backbones ResNet-50 and ResNet-101 as shown in Table 4. First, we compare the performance of CODNet for two different resolutions, that is, CODNet-512 with input resolution 512 × 1024 and CODNet-1080 with input resolution 1080 × 1920. As shown in Table 4, as the resolution increases, the performance of our model also increases. When the input resolution is 1080 × 1920, CODNet achieves the highest performance.

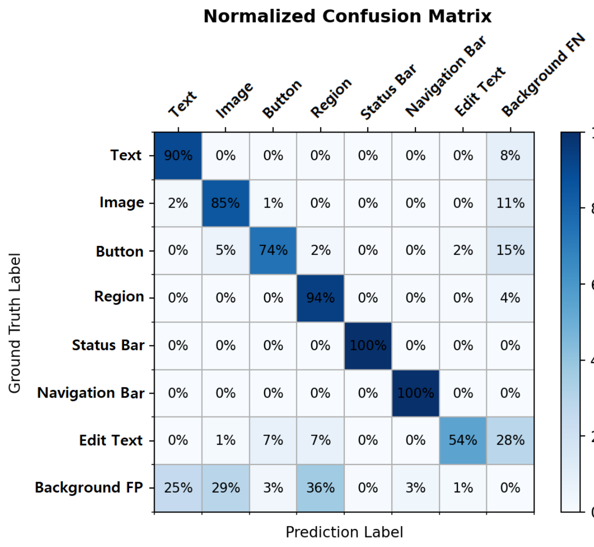


FIGURE 5. Confusion matrix of CODNet-1080.

Next, we investigate the performance of our model for each aspect ratio candidate generated through mobile dataset analysis. Table 4 shows the accuracy for each aspect ratio candidate obtained through k-means clusters on ReNet-50 and ReNet-101. For the aspect ratio candidate when using 5-means clusters, CODNet achieves the best performance for all cases of resolution and backbone. The result shows that the accuracy does not necessarily increase in proportional to k. Therefore, we use RARC with $k = 5$ as the aspect anchor ratio for our model.

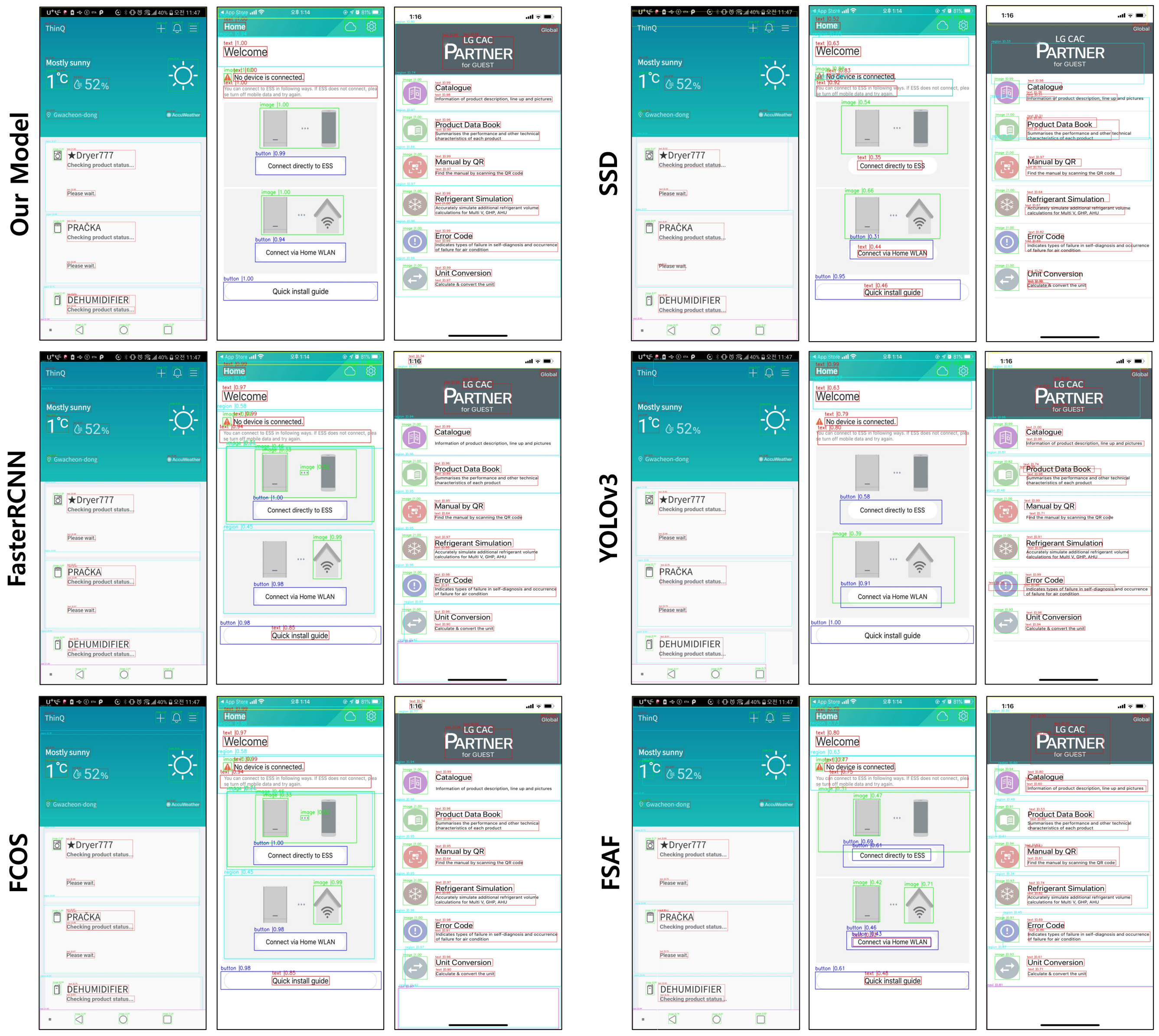


FIGURE 6. Inference result of CODNet and existing deep learning models.

D. CLASS PERFORMANCE

We compare the detection accuracy of each class by using the confusion matrix after training our model with dataset. Figure 4 shows the accuracy of each class on the CODNet-1080 model when using ResNet-101 as a backbone. According to the confusion matrix, CODNet-1080 has significantly higher accuracy for Status bar and Navigation bar. Those two components are not difficult to detect, since their features are clear. Components which change the mobile screen by clicking like text, image and buttons are generally well detected. However, in the case of regions, false positives occur frequently, since they are similar to the background. In particular, the edit text class is one of the most difficult to detect among the clickable object classes, since it is usually made up of only one word or one line, and hence their characteristics are not clear. The accuracy for edit text class on YOLOv3 is 40%, while the accuracy for edit text class on CODNet is 54%, resulting in clear performance improvement.

E. COMPARISON TO OTHER DEEP LEARNING MODEL

We compare the mAP of CODNet and with the other models. As shown in Table 5, we compare mAP according to the size of the object: small, medium and large. For all the cases, CODNet-1080 achieves the highest performance. Among the models excluding CODNet, DetectoRS with ResNet-101 as the backbone has the highest performance. CODNet-1080 has higher mAP of 0.08, 0.122 and 0.096 for small, medium and large sized objects respectively than those of DetectoRS with ResNet-101 as backbone. Among models excluding

CODNet, SSD300 has the lowest mAP of 0.407, while DetectoRS with ResNet-101 as a backbone has the highest mAP of 0.784. As a one-stage model, YOLOv5 has the highest mAP of 0.742. CODNet-1080 achieves the highest mAP of 0.865. It also shows the best performance among the one-stage models.

V. CONCLUSION

In this paper, we have proposed CODNet, a new deep learning neural network for detecting clickable objects in a wide range of mobile screen resolutions. CODNet provides performance improvement and scalability by designing a deep learning network which consists of three modules: Feature extraction module, Deconvolution module and Prediction module. Feature extraction module shows performance improvement through minimizing feature loss by extending the resolution of input image to 1080 $\times$ 1920. Deconvolution module provides feature map of various sizes by upsampling feature map through top-down paths and lateral connections. Prediction module improves the performance by using the Anchor Transfer block which selects the most suitable for the mobile environment among the set of anchor candidates obtained through mobile data set analysis. Moreover, we improve the object detection performance by building a new mobile screen dataset which consists of data collected from various resolutions and operating systems. The performance of CODNet is confirmed through various experiments. Feature extraction module achieves the lowest error rate compared to other models. CODNet with Deconvolution module achieves a mAP of 0.865, which is 9.4 higher than CODNet without Deconvolution module. CODNet achieves higher performance as the resolution increases. For the aspect ratio candidate when using 5-means clusters, CODNet achieves the best performance for all cases of resolution and backbone. CODNet-1080 achieves the highest mAP of 0.865.

For future works, we shall work on the details of CODNet such as the number of layers for each module, activation function and hyperparameters, and continue to work on the test for a wide range of smart devices in the real environment to further improve our model.

REFERENCES

[1] G. Bae, G. Rothermel, and D.-H. Bae, "Comparing model-based and dynamic event-extraction based GUI testing techniques: An empirical study," *J. Syst. Softw.*, vol. 97, pp. 15–46, Nov. 2014.

[2] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for Android: Are we there yet? (E)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 429–440.

[3] W. Muangsiri and S. Takada, "Random GUI testing of Android application using behavioral model," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 27, nos. 9–10, pp. 1603–1612, Nov. 2017.

[4] A. C. R. Paiva, J. M. E. P. Gouveia, J.-D. Elizabeth, and M. E. Delamaro, "Testing when mobile apps go to background and come back to foreground," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Apr. 2019, pp. 102–111.

[5] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with REMAUI (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 248–259.

[6] K. Moran, C. Bernal-Cardenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Trans. Softw. Eng.*, vol. 46, no. 2, pp. 196–221, Feb. 2020.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25, Dec. 2012, pp. 1097–1105.

[8] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights Imag.*, vol. 9, pp. 611–629, Jun. 2018.

[9] X. Zhang, L. de Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach, A. Everitt, and J. P. Bigham, "Screen recognition: Creating accessibility metadata for mobile applications from pixels," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, May 2021, pp. 1–15.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision—ECCV 2016*. Cham, Switzerland: Springer, 2016, pp. 21–37.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[12] P. Zhu, Y. Li, T. Li, W. Yang, and Y. Xu, "Gui widget detection and intent generation via image understanding," *IEEE Access*, vol. 9, pp. 160697–160707, 2021.

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

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 91–99.

[16] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into high quality object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6154–6162.

[17] S. Qiao, L.-C. Chen, and A. Yuille, "DetectoRS: Detecting objects with recursive feature pyramid and switchable atrous convolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 10213–10224.

[18] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 840–849.

[19] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully convolutional one-stage object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9627–9636.

[20] H. Law and J. Deng, "CornerNet: Detecting objects as paired keypoints," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 734–750.

[21] Z. Dong, G. Li, Y. Liao, F. Wang, P. Ren, and C. Qian, "CentripetalNet: Pursuing high-quality keypoint pairs for object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10519–10528.

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2117–2125.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[25] Q. Chen, Y. Wang, T. Yang, X. Zhang, J. Cheng, and J. Sun, "You only look one-level feature," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13039–13048.

[26] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021, *arXiv:2107.08430*.

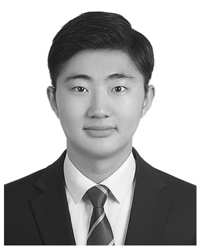
[27] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[28] G. Jocher et al., "ultralytics/yolov5: v6.2—YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations, v6.2," Zenodo, Aug. 2022, doi: 10.5281/zenodo.7002879.

[29] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.

[30] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "RICO: A mobile app dataset for building data-driven design applications," in *Proc. 30th Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 2017, pp. 845–854.
[31] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, L. Zhu, and G. Li, "Object detection for graphical user interface: Old fashioned or deep learning or a combination?" in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1202–1214.
[32] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "ICONINTENT: Automatic identification of sensitive UI widgets based on icon classification for Android apps," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 257–268.
[33] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*.
[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[35] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.
[36] S. H. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing*, vol. 378, pp. 112–119, Feb. 2020.
[37] A. Fred Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.
[38] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.
[39] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, "Bounding box regression with uncertainty for accurate object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2888–2897.
[40] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and W. Zisserman, "The PASCAL visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Sep. 2010.

BOSEON KANG is currently pursuing the integrated M.S./Ph.D. degree with the Visual Information Processing Department, Korea University, Seoul, South Korea. His research interests include deep learning and image processing.



MINSEOK JO received the M.S. degree from the Department of Electrical Engineering, Korea University, in 2022. He is currently a Research Engineer at Hyundai Mobis. His research interests include deep learning and distributed parallel processing.

CHANG-SUNG JEONG (Member, IEEE) received the B.S. degree from Seoul National University, in 1981, and the M.S. and Ph.D. degrees from Northwestern University, in 1985 and 1987, respectively. He was a Professor with POSTECH, from 1982 to 1992. He was an Associate Researcher with UCSC, from 1998 to 1999. He is currently a Professor with the Department of Electrical and Computer Engineering, Korea University. His research interests include distributed parallel computing, grid computing, ubiquitous computing, networked virtual computing, and development of highly intensive applications, such as stereo image processing and 3-D visualization on collaborative grid and ubiquitous computing environment.