ARCN: A Real-time Attention-based Network for Crowd Counting from Drone Images

Subhrajit Nag[1], Yash Khandelwal[2], Sparsh Mittal[3], C. Krishna Mohan[1], A. Kai Qin[4]
[1]IIT Hyderabad, India, [2]Birla Institute of Technology, Mesra, India,
[3]IIT Roorkee, India, [4]Swinburne University of Technology, Melbourne, Australia
cs17resch11006@iith.ac.in,yashkhandelwal.023@gmail.com,sparshfec@iitr.ac.in,ckm@cse.iith.ac.in,kqin@swin.edu.au

***Abstract*—Crowd counting is the process of counting or estimating the number of individuals in a crowd. There has been a rapid surge in the amount of Unmanned Aerial Vehicles (UAV) images over the last few years. However, efficient crowd counting techniques from UAV images have hardly come into the focus of the research community. Crowd counting from UAV images has its unique challenges compared to crowd counting from images in natural scenes. Moreover, solving the problem in real-time makes the task even harder.**

In this paper, we introduce an attention-based encoder-decoder model called Attention-based Real-time CrowdNet (ARCN). ARCN is a computationally efficient density estimation-based crowd counting model. It can perform crowd-counting from UAV images in real-time with high accuracy. Ours is the first work that proposes a real-time density map estimation and crowd counting model from drone-based images. The key idea of our work is to add "Convolution Block Attention Module" (CBAM) blocks in-between the bottleneck layers of the MobileCount architecture. The proposed ARCN model achieves an MAE of 19.9 and MSE of 27.7 on the DroneCrowd dataset. Also, on NVIDIA GTX 2080 Ti GPU, ARCN has a processing speed of 48 FPS, making it a real-time technique. The pre-trained model is available at https://bit.ly/3na7LUy

I. Introduction

Recent years have seen a tremendous interest in the research on estimating the object count from a complex scene across an extensive range of domains such as surveillance, microbiology, crowd management, product counting, and traffic-flow monitoring. Crowd counting is a method of counting or enumerating the number of *people* from an image or video. Automated techniques of object counting reduce the cost and perform the task with higher efficiency than manual object counting methods. Crowd counting can be performed from either ground images or drone images. Of these, performing crowd-counting from drone images is of particular significance. Drone-based crowd counting can be deployed for mission-critical tasks such as video surveillance to help in rescue missions from treacherous terrains and public safety or to estimate the number of people in a pilgrimage or any other gathering.

Although important, automated crowd counting poses several challenges. Firstly, recognizing and counting very small objects in a crowded environment requires dealing with occlusions. The scale of the objects in remote-sensing and UAV images is not fixed and varies drastically [1]. Performing object counting in real-time presents unique challenges. As per our knowledge, DroneCrowd dataset [2] is the only public dataset available for estimating the number of people from UAV-based images. Thus, the availability of inadequate datasets is a crucial challenge.

With the advent of deep learning, CNN-based models have been dominating diverse computer vision tasks, and crowd counting is no exception. Over the past few years, many CNNs have been proposed for performing crowd-counting. Some previous works, e.g., MobileCount [3] perform real-time crowd counting. However, they work on the images captured from the ground and thus, do not work on drone-based or remote-sensing images. To the best of our knowledge, only STANet [2] has paid attention to crowd counting from drone-based images.

In this paper, we seek to address the challenges mentioned above by proposing a CNN-based density estimation model [4], [5]. We propose a network named ARCN, for real-time crowd-counting from drone images. ARCN is inspired from MobileCount. Specifically, similar to MobileCount, in the encoder, ARCN uses a simplified MobileNetV2 which has fewer convolution layers and smaller feature map resolution. Also, similar to MobileCount, ARCN uses 4 bottleneck layers instead of 7 bottlenecks layers and has a 3 x 3 max-pooling layer of stride 2 before the bottlenecks. Further, similar to MobileCount, we use RefineNet, which is a light-weight model, for designing the decoder [6] and a 1x1 convolution layer as the prediction layer for producing the density map of the corresponding input image. The key novelty of our work is to use a CBAM module [7] after each of the bottleneck layers. This reduces the error values for a negligible increase in the number of computations and parameters compared to MobileCount. Still, as we show in Table I, ARCN has the 3rd best performance on the DroneCrowd dataset. Also, it has one of the lowest parameters and computations among the state-of-art models for crowd counting. Further, our work can be extended to various object counting problems such as estimating the green cover in a region, vehicle counting, or counting the number of rare animal species from their colonies.

978-1-6654-4175-9/21/$31.00

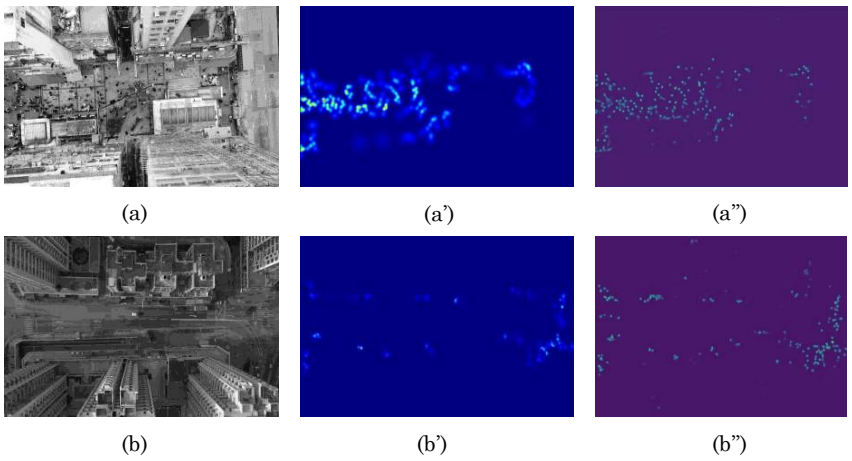

Fig. 1. (a)-(b) Sample Images from Dataset. (a')-(b') Corresponding "Ground Truth Density Maps". (a")-(b") Corresponding "density maps" generated by ARCN.

Figures 1 (a)-(b) show two sample images from the DroneCrowd dataset along with their visual representations. We can use this to compare the output density estimation maps produced by ARCN with the ground truth density estimation maps. Figures 1(a)-(b) show the original image from the dataset. Figures 1(a')-(b') show the "ground truth density maps" of the image, while the third column is the output density map produced by our model ARCN. We can see that the output density map is quite similar to the ground truth density estimation map. In the first image, the actual count of persons is 178, while for the second image the original number of persons is 156. ARCN outputs the count as 186 and 158 for the first and second images respectively. So, even the count of persons is quite similar to the actual count.

II. Background and Related Work

A. Crowd Counting

The existing crowd counting techniques may be broadly categorized into three types:

1. **Detection-based approaches** These approaches use computer vision-based techniques to detect an entire person or part of a person in a scene. This is mostly done using a sliding window detector. Then the total number of detections is aggregated to get the estimate of people. They are successful in sparsely-crowded scenes. However, they perform poorly for highly crowded scenes due to "clutter" and "occlusion".

2. **Regression-based approaches** These approaches learn a mapping from the features extracted from an image patch to the number of people in the image [8], [9]. They work in two stages. The first stage extracts local or global features from the image patch. The second stage uses some regression technique like linear regression or ridge regression. Due to this reason, the quality of feature maps produced is adversely affected, which causes inaccurate results.

3. **Density estimation-based approaches** Initial works based on this approach learn to map the features in the local region to its corresponding "object density maps". The recent works [4], [5] have utilized CNN-based approaches to obtain the density map and then estimate the number of people from that map. State-of-the-art "crowd counting techniques" use CNN-based density estimation approaches. Our proposed ARCN falls under this category.

Crowd counting models can be categorized into two types- multi-column and single-column. In the multi-column crowd counting models, there are several columns of CNNs which are used for the purpose of counting. This is mainly done to deal with the challenge of varying scales of crowds in the images. However, there are a few drawbacks to this type of model. Firstly, their complex network structure increases the training time. Also, there might be a lot of redundant information due to the multiple columns of CNN. By contrast, in the single-column models, there is a single column of CNN which is usually deeper than those used in the multi-column models.

B. Convolution Block Attention Module

Attention mechanism helps CNNs to learn and selectively concentrate on essential parts of the input and ignore irrelevant background information in the rest of the input [10], [11]. In the case of crowd images, useful information is the instances of people, particularly their heads. A major portion of remote sensing images may include large buildings, trees, roads, etc., which need to be ignored by a crowd-counting engine.

CBAM ("Convolutional Block Attention Module") [7] consists of a simple 2D-convolutional layer, multi-layer perceptron (in the case of channel attention), and a "sigmoid function" at last to generate an attention mask over the input feature map. It takes a CxHxW feature map as an input and gives an output attention map of dimension CxHxW. Then the element-wise multiplication of this attention map is performed with the input feature map to get a more processed and highlighted output.

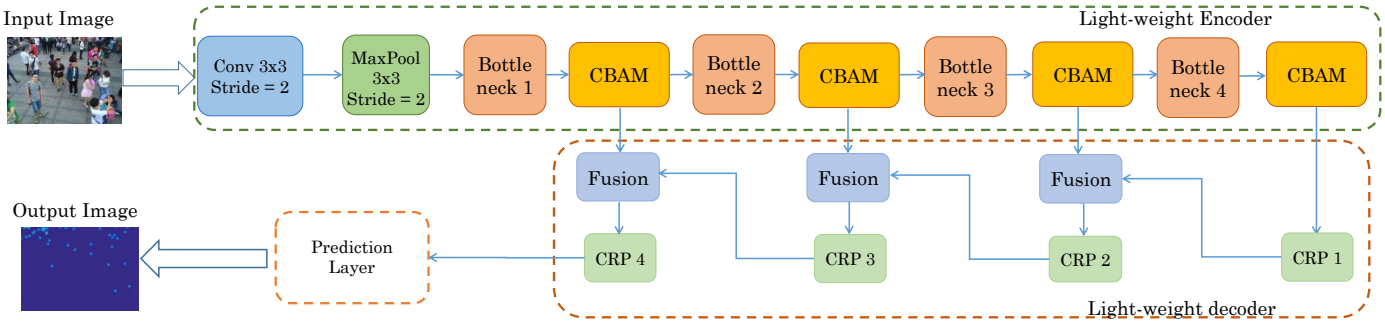

Fig. 2. ARCN network architecture

The attention mechanisms are applied to spatial and channel dimensions sequentially. A "deep neural network" used for crowd-counting must have low memory, and computation overhead [12], so it can run on general-purpose systems and mobile platforms [13]. Hence, our goal is to keep these overheads low.

C. Encoder-Decoder

CNNs have made remarkable advances in semantic segmentation. Recent methods use encoder-decoder architecture to generate pixel-wise segmentation. The encoder produces low-resolution feature maps, and the decoder restores them to high-resolution images for pixel-wise predictions.

III. Proposed Model

We now introduce the architecture of our proposed ARCN. We also discuss the loss function and ground truth generation procedure. Note that the choice of the backbone network for the encoder, decoder, the number of bottleneck layers, the insertion of max-pooling layer before the bottleneck layers, and the choice of prediction layer are adopted from the MobileCount network architecture. The key novelty of our work is to use a CBAM module [7] after each of the bottleneck layers.

Figure 2 shows our model architecture. It is based on encoder-decoder architecture with embedded CBAM modules in it. The encoder aims to capture semantic or contextual information, while the decoder is meant to recover spatial information. To keep the parameters and floating point operations of the model low, we use MobileNetV2 as the backbone network and further customize it to perform crowd counting in drone-based images. The MobileNetV2 in the encoder portion is optimized to make it a mobile-oriented model to deliver high accuracy while keeping the parameters and mathematical computations as low as possible. Also, the reduced number of parameters helps in faster training. MobileNetV2 has an "inverted residual structure", and there are "linear bottlenecks" between layers.

To tackle the different sizes of persons in the image, we need to incorporate the contextual information at multiple scales. In our architecture, we use RefineNet for the decoder network. RefineNet was originally proposed for the purpose of semantic segmentation by combining the high-level spatial features with the low-level (detailed) features. This helps reduce the computation cost and achieve real-time performance. Therefore, we choose RefineNet in our architecture. We now describe each component of ARCN in detail.

A. Feature extraction with CBAM (Encoder)

The model encoder extracts the features from the image and passes them to the decoder to recover the spatial resolution. We place attention blocks to solve the complex, cluttered background problem from the drone-based images. So, the idea is to make the model ignore the background information in the images and focus on the regions with crowd or instances of people. This can be done once the feature maps have been obtained so that they can be refined using the attention blocks where spatial and channel attention is applied. Hence, in the encoder, we place CBAM after each bottleneck, so that the feature maps produced by the encoder are refined and fed to the decoder and the next bottleneck. Placing CBAM after each bottleneck ensures that the channel attention and spatial attention are preserved across all the encoder stages.

The encoder consists of a 3x3 convolution layer followed by a 3x3 max pooling layer with a stride of 2 before the bottleneck. This makes the model lightweight due to the reduction in input resolution during the initial stage of the encoder layer. The idea of having 4 bottleneck layers instead of 7 bottleneck layers has been taken from MobileCount since they have experimentally shown that removing the last three bottleneck layers saves both computation and improves accuracy. After each bottleneck, we insert an "attention module" to capture more high-level semantic and contextual information, which helps remove congested backgrounds to highlight the regions of the object. The output feature maps of a bottleneck are fed to the attention module, and the refined features hence produced are output to the decoder and the next bottleneck. The detailed architecture of CBAM module is shown in Figure 4.

B. Decoder

In the decoder, the output from the last CBAM block after the backbone in the encoder propagates through "chained residual pooling" (CRP) (Figure 3(a)) block. Then, it is fed

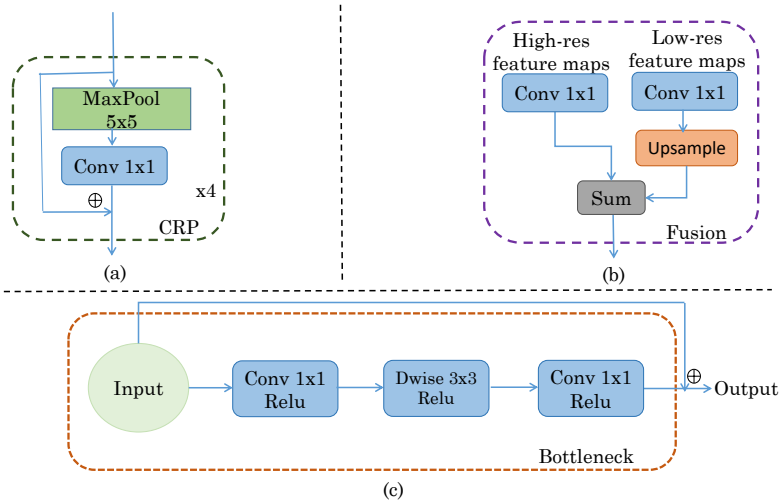

Fig. 3. a) CRP block and b) Fusion block in decoder c) Bottleneck block of our encoder architecture [6]

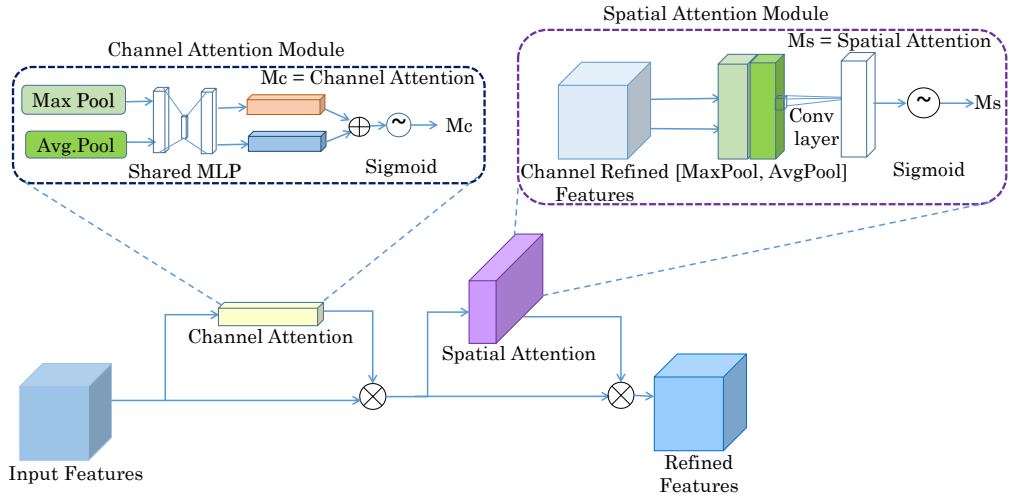

Fig. 4. Architecture of CBAM block [7]

into the "fusion block" (Figure 3(b)). The output feature maps from the previous CBAM blocks are fed into the respective fusion blocks. The output feature maps from the fusion blocks are further propagated across multiple CRP and fusion blocks. This is done to achieve the target resolution.

Fusion Block: The fusion block is a multi-resolution block. It is used to combine the inputs from the bottleneck of the encoder and the previous CRP block into a high-resolution feature map as shown in Fig. 3(b). In the first stage, the fusion block performs convolutions for adapting to the input dimension. It outputs feature maps of similar dimensions as the smallest input dimension. Then it converts the feature maps of smaller dimensions to the input size having the largest dimension by up-sampling. Inside the fusion block, the feature map in both paths convolves with a filter of size 1x1. Finally the output feature maps from both the paths are fused by summation.

Chained residual pooling: From the backbone, the last output feature map with the smallest resolution of 1/32 times the resolution of the input image propagates through the CRP block, as shown in Figure 2. The CRP is made of 4 consecutive pooling blocks. The output feature maps of all pooling blocks and the input feature map are fused. This is done by adding the residual connections. These features are then fused together with the help of learnable weights.

C. Prediction Layer

The prediction layer generates density maps by applying 1 x 1 convolution. Then, it uses bilinear interpolation to upsample the resulting "density map" to the size of the initial image.

D. Loss Function

Our network transforms the input image to its corresponding "density map" directly. We use the Euclidean distance to measure divergence between the "ground truth" and the

predicted "density map". The loss function adopted in this work is given as follows:

$$L_{den} = \frac{1}{M}\sum_{j=1}^{M}\left\|D^{PM}(j)) - D^{GT}(j)\right\|_2^2 \quad (1)$$

Where M is the count of training samples; $D^{GT}(j)$ denotes the ground truth density maps and $D^{PM}(j)$ represents the predicted density maps; j denotes the j-th training sample; $\|\cdot\|_2$ represents the "Euclidean distance".

E. Ground truth generation

We use the "geometry-adaptive kernels" to tackle the highly congested scenes, similar to previous works [5]. The ground truth is generated by blurring each annotated head by applying a "Gaussian kernel" that is normalized to 1. The geometry-adaptive kernel is given as:

$$D(z) = \sum_{j=1}^{m}\delta(z - z_j) * G_{\sigma_j}(z) \quad (2)$$

where z_j, j = 1, ..., m is the location of the j-th head annotation and $\delta(\cdot)$ represents a delta function; σ_j shows the variance of the "Gaussian kernel" applied at location j. We have used $\sigma_j = 4$ in this work.

IV. Experimental Results

We now discuss the evaluated dataset, the implementation details and adopted evaluation metrics. Then, we compare the performance of different state-of-the-art "density based crowd counting" method with our proposed method, named ARCN.

A. Dataset

The **DroneCrowd** [2] dataset consists of total 33,600 frames from 112 video clips with a resolution of 1920×1080 pixels. The data is taken from different drone-mounted cameras [2]. The images in the dataset cover a wide variety of scenarios. The DroneCrowd dataset contains 24600 training images and 9000 testing images. The dataset has a maximum count of 455 people, a minimum count of 25 people and an average count of 144.8 people in each image.

B. Implementation details

We use PyTorch framework on an NVIDIA GeForce GTX 2080Ti GPU. We use Adam optimizer with a batch size of 16. The "weight decay rate" is 1×10^{-4} and the "initial learning rate" is 1×10^{-4}.

C. Evaluation Metrics

We use "Mean Absolute Error (MAE)" and "Mean Square Error (MSE)" metrics, defined as:

$$MAE = \frac{1}{M}\sum_{i=1}^{M}\left|C_i^{PM} - C_i^{GT}\right| \quad (3)$$

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(C_i^{PM} - C_i^{GT})^2 \quad (4)$$

where M is the count of test images; C_i^{PM} is the projected crowd estimate in an image, which is equal to the summation of all pixels of the density maps. C_i^{GT} is the real count (ground-truth).

D. Results and comparison

We compare our model with several recent models on the DroneCrowd *test* dataset. Table I shows the results.

Parameter count: In terms of parameter count, our model has one of the lowest parameter counts of 3.5M parameters, which is only more than MCNN and C-MTL and comparable to the parameter count of MobileCount model. DA-Net, on the other hand, has the highest number of parameters - 16.47M.

Number of computations: The number of GFLOPs in ARCN is among the lowest of all the models. The use of CBAM in ARCN leads to a slight increase in the GFLOPs compared to that of MobileCount. However, the use of CBAM also reduces the error rates, which makes it a profitable trade-off. Overall, ARCN is suitable for deployment on edge devices.

Accuracy results: ARCN achieves an overall MAE and MSE of 19.9 and 27.7, respectively. Comparing ARCN with other state-of-the-art models, the estimation error rates achieved by ARCN are in the top-3. Only STANet and CSRNet achieve lower error rates than ARCN. STANet has the lowest error rates with an MAE and MSE of 16.7 and 19.8, respectively. Also, the error rates of ARCN are almost comparable to that of CSRNet. However, the number of computations and the number of trainable parameters for CSRNet and STANet are several times higher than that of ARCN. Also, ARCN achieves much higher FPS values than CSRNet and STANet. Thus, considering a balance of multiple metrics, our model ARCN may be regarded as the best. ARCN has a single-column architecture. So it has a simple architecture that is not only easy to train but also efficient. This way, ARCN avoids an unnecessary increase in the number of parameters. The use of CBAM modules in ARCN enhances the representation power of the already-reduced number of feature maps coming out of the bottleneck layers. CBAM allows ARCN to focus on the relevant portions of the feature maps. This helps ARCN achieve lower error rates as compared to MobileCount. The high FPS count of ARCN, along with good error rates and low number of computations is due to the careful and balanced selection of lightweight CNN models.

While MCNN achieves a good FPS rate and has the least number of parameters, its error rates are far higher than ARCN. MCNN has very less parameters. Also, the number of large kernels used in MCNN is small. This leads to lower computational complexity. Therefore, it has a high FPS count. However, the feature maps produced from the

TABLE I
RESULTS OF VARIOUS STATE-OF-ART MODELS ON THE *testing* DATASET (EXCEPT FOR FPS, FOR OTHER METRICS, LOWER IS BETTER). (*=VALUE IS NOT KNOWN) († ON NVIDIA GTX 2080 TI GPU, ALL OTHER FPS RESULTS ARE ON NVIDIA GTX 1080TI GPU)

Method	Type of Architecture	#Parameters	#GFLOPs	FPS (Batch Size = 1)	Errors	
					MAE	MSE
STANet [2]	Single-column	10.01M	765	2.74	16.7	19.8
CSRNet [4]	Single-column	16.26M	857.84	3.92	19.8	25.6
ARCN (ours)	Single-column	**3.5M**	**16.51**	**48.07** †	**19.9**	**27.7**
MobileCount [3]	Single-column	3.4M	16.49	70.42 †	22.4	28.8
MCNN [5]	Multi-column	0.13M	56.21	28.98	34.7	42.5
DA-Net [14]	Single-column	16.47M	648	2.52	36.5	47.3
ACSCP [15]	Multi-column	5.1M	*	1.58	48.1	60.2
C-MTL [16]	Multi-column	2.45M	238.4	2.31	56.7	65.9
MSCNN [17]	Single-column	2.9M	*	1.76	58	75.2
SwitchCNN [18]	Multi-column	15.11M	*	0.014	66.5	77.8
LCFCN [19]	Single-column	*	*	3.08	136.9	150.6
AMDCN [20]	Multi-column	*	*	0.16	165.6	167.7

shallow columns of MCNN are fused late. The weighted average method used to perform late fusion cannot distinguish between the density of crowd in different portions of a single image. Hence, it gives higher error rates.

Table II shows the effect of batch size on the processing speed of ARCN model. We can see that as we increase the batch size to 16, the inference speed increases.

TABLE II
FPS COUNTS WITH VARYING BATCH SIZES

Batch Size	FPS
2	57.8
4	64.1
8	66.6
16	68.02

V. CONCLUSION

We have proposed a network called ARCN for real-time "crowd counting" from drone images. Our model has a light-weight encoder-decoder attention-based architecture. On DroneCrowd dataset, we obtain 3rd best performance on the dataset. With low parameters and a high FPS, our model achieves state-of-the-art in real-time crowd counting from drone images. We hope that ARCN model can be used with remote sensing images also.

REFERENCES

[1] S. Srivastava *et al.*, "A Survey of Deep Learning Techniques for Vehicle Detection from UAV Images," *Journal of Systems Architecture*, vol. 117, p. 102152, 2021.

[2] L. Wen, D. Du, P. Zhu, Q. Hu, Q. Wang, L. Bo, and S. Lyu, "Drone-based joint density map estimation, localization and tracking with space-time multi-scale attention network," 2019.

[3] P. Wang, C. Gao, Y. Wang, H. Li, and Y. Gao, "Mobilecount: An efficient encoder-decoder framework for real-time crowd counting," *Neurocomputing*, vol. 407, pp. 292–299, 2020.

[4] Y. Li, X. Zhang, and D. Chen, "CSRNet: Dilated convolutional neural networks for understanding the highly congested scenes," in *CVPR*, 2018, pp. 1091–1100.

[5] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, "Single-image crowd counting via multi-column convolutional neural network," in *CVPR*, 2016, pp. 589–597.

[6] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *CVPR*, 2017, pp. 1925–1934.

[7] S. Woo, J. Park, J. Y. Lee, and I. S. Kweon, "CBAM: Convolutional block attention module," in *ECCV*, 2018, pp. 3–19.

[8] H. Idrees, I. Saleemi, C. Seibert, and M. Shah, "Multi-source multi-scale counting in extremely dense crowd images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2547–2554.

[9] A. B. Chan and N. Vasconcelos, "Bayesian poisson regression for crowd counting," in *2009 IEEE 12th international conference on computer vision*. IEEE, 2009, pp. 545–551.

[10] P. Rajput *et al.*, "Improving Accuracy and Efficiency of Object Detection Algorithms using Multiscale Feature Aggregation Plugins," in *ANNPR*, 2020, pp. 65–76.

[11] R. K. Saini *et al.*, "ULSAM: Ultra-Lightweight Subspace Attention Module for Compact Convolutional Neural Networks," in *IEEE WACV*, 2020, pp. 1616–1625.

[12] N. K. Jha *et al.*, "E2GC: Energy-efficient Group Convolution in Deep Neural Networks," in *IEEE VLSID*, 2020.

[13] S. Mittal, P. Rajput, and S. Subramoney, "A Survey of Deep Learning on CPUs: Opportunities and Co-optimizations," *IEEE TNNLS*, 2021.

[14] Z. Zou, X. Su, X. Qu, and P. Zhou, "Da-net: Learning the fine-grained density distribution with deformation aggregation network," *IEEE Access*, vol. 6, pp. 60 745–60 756, 2018.

[15] Z. Shen, Y. Xu, B. Ni, M. Wang, J. Hu, and X. Yang, "Crowd counting via adversarial cross-scale consistency pursuit," in *CVPR*, 2018, pp. 5245–5254.

[16] V. A. Sindagi and V. M. Patel, "CNN-based cascaded multi-task learning of high-level prior and density estimation for crowd counting," in *International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2017, pp. 1–6.

[17] L. Zeng, X. Xu, B. Cai, S. Qiu, and T. Zhang, "Multi-scale convolutional neural networks for crowd counting," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 465–469.

[18] D. Babu Sam, S. Surya, and R. Venkatesh Babu, "Switching convolutional neural network for crowd counting," in *CVPR*, 2017, pp. 5744–5752.

[19] I. H. Laradji, N. Rostamzadeh, P. O. Pinheiro, D. Vazquez, and M. Schmidt, "Where are the blobs: Counting by localization with point supervision," in *ECCV*, 2018, pp. 547–562.

[20] D. Deb and J. Ventura, "An aggregated multicolumn dilated convolution network for perspective-free counting," in *CVPR Workshops*, 2018, pp. 195–204.