COUNT Forest: CO-voting Uncertain Number of Targets using Random Forest for Crowd Density Estimation

Viet-Quoc Pham, Tatsuo Kozakaya, Osamu Yamaguchi and Ryuzo Okada
Corporate Research and Development Center, Toshiba Corporation
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki, 212-8582, Japan
quocviet.pham@toshiba.co.jp

Abstract

This paper presents a patch-based approach for crowd density estimation in public scenes. We formulate the problem of estimating density in a structured learning framework applied to random decision forests. Our approach learns the mapping between patch features and relative locations of all objects inside each patch, which contribute to generate the patch density map through Gaussian kernel density estimation. We build the forest in a coarse-to-fine manner with two split node layers, and further propose a crowdedness prior and an effective forest reduction method to improve the estimation accuracy and speed. Moreover, we introduce a semi-automatic training method to learn the estimator for a specific scene. We achieved state-of-the-art results on the public Mall dataset and UCSD dataset, and also proposed two potential applications in traffic counts and scene understanding with promising results.

1. Introduction

Counting objects in images is important in many real-world applications, including traffic control, industrial inspection, surveillance and medical image analysis. Counting in crowded scene is non-trivial owing to severe inter-object occlusion, scene perspective distortion and complex backgrounds. Besides counting, estimating crowd density is necessary for understanding crowd behaviour, especially in large public spaces such as train stations and streets. This paper addresses the problem of counting in public scenes based on crowd density estimation, as shown in Fig. 1.

Existing counting methods can be classified into three categories: counting by detection [19, 30], counting by clustering [8, 25] and counting by regression [10, 11, 21, 16, 9, 23]. In the first two categories, counting is based on individual detection and motion segmentation, which are sensitive to heavy object occlusion and cluttered background. Particularly, for such crowded scenes that only a part of object instance can be observed, detection and segmentation of individuals become impracticable. In counting by regression, counting techniques learn a mapping between low-level features and people count, and therefore they can avoid explicit object detection and segmentation in crowded scenes.

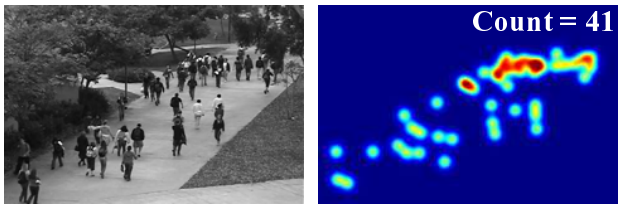

Figure 1. Our objective is to estimate the object density map and the number of objects (right) from an input image (left).

Counting by regression methods differs depending on the target of regression: the object count, or the object density. Chen *et al*. [11, 10] learn the regression mapping between the low-level imagery feature and the number of object. They achieved state-of-the-art results for counting people by introducing the concept of cumulative attribute to the regression problems, which improved the counting accuracy by considering the cumulative dependent nature of regression labels [10]. In another approach, Lempitsky *et al*. [21] estimate an image density whose integral over any image region gives the count of objects within that region. The object density is more informative than the object count, since it can give a *rough estimate on the object locations*, as shown in Fig. 1. Therefore, crowd density estimation is more useful for understanding the crowd behaviour. To estimate the crowd density, Lempitsky *et al*. learn a linear transformation of the feature representation $f(x)$ that approximates the density function at each pixel $D(x) = \omega^T f(x)$ [21]. The problem here is that it is difficult to design a feature satisfying the linear approximation hypothesis.

This paper presents a patch-based approach for crowd

density estimation in public scenes. Different from Lempitsky *et al.*'s work [21] assuming a linear transformation, we aim to learn the nonlinear mapping between patch features and relative locations of all objects inside each image patch using a random forest framework. Object locations estimated in each patch will contribute to generate the patch density map through Gaussian kernels. The patch feature used in our method is much simpler and scene-independent. We name our model *COUNT forest* (CO-voting Uncertain Number of Targets using random forest), as it uses random forest regression from multiple image patches to vote for the densities of multiple target objects. We build the forest in a coarse-to-fine manner with two split node layers, and train a prediction label at each leaf node. One of the advantages of the COUNT forest model is that it requires much less memory to build and reserve the forest, compared with regression forest models with dense structured labels proposed in [20, 16]. This is a prerequisite for an embedded computer vision system.

As there is a large variation in appearance and shape between crowded image patches and non-crowded ones, we propose a *crowdedness prior*, which is a global property of the image, to train two different forests corresponding to this prior. We develop a robust density estimator by adaptively switching between the two forests. Moreover, we also propose a non-trivial approach of effective forest reduction by using permutation of decision trees. This improvement speeds up the estimation so that it can run in real-time. Another contribution is the introduction of a semi-automatic training method to learn the estimator for a specific scene. We synthesize training samples randomly from a large set of segmented human regions and the target scene background. The synthesized training samples not only facilitate labor-saving, but also adapt our estimator to the target scene.

Finally, we verified the performance on various crowded scenes. We achieved state-of-the-art results on the public Mall dataset [11] and UCSD dataset [9], and confirmed high performance for multiple-class counting and scene adaptation with good results for the Train Station dataset [31] and a newly captured dataset. We also proposed two potential applications in traffic counts and scene understanding with promising results.

We discuss related works in Section 2, and explain the proposed COUNT forest model in Section 3. Improvements of the method including the crowdedness prior, forest permutation and semi-automatic training are explained in Section 4. We present experimental results in Section 5.

2. Related Works

Counting by regression: Chen *et al.* [11, 10] learn the regression mapping between the low-level imagery feature and the object count. In [11], they propose a single multi-output model based on ridge regression, which takes inter-dependent local features from local regions as input and people count from individual regions as multidimensional structured output. They further introduce the concept of cumulative attribute for regression to address the problems of feature inconsistency and sparse data [10]. In this work, they convert the people count into a binary cumulative attribute vector, and use it as the intermediate representation of the image to learn the regression models. Loy *et al.* [23] develop a framework for active and semi-supervised learning of a regression model with transfer learning capability, in order to reduce laborious data annotation for model training. Lempitsky *et al.* [21] model the crowd density at each pixel, casting the problem as that of estimating an image density whose integral over any image region gives the count of objects within that region. This model is effective, but the linear model requires a scene-dependent and relatively complex set of features (BoW-SIFT).

Random forest: Part-based object detection methods learn the appearance of object parts and their spatial layout to model an object. Okada [24] and Gall *et al.* [17] proposed a random forest framework [3, 7] to learn a direct mapping between the appearance of an image patch and the object location. The combination of the tree structure and simple binary tests makes training and matching against the codebook very fast, whereas clustering-based learning of explicit codebooks is considerably more expensive in terms of both memory and time. The idea of probabilistic voting was also exploited in object tracking [18], action recognition [28], facial feature detection [12], and human pose estimation [13]. The main difference between the COUNT forest and other voting-based methods is the output of regression. In our method, each patch votes for locations of nearby objects, whose quantity and structure are not fixed due to the large variation of crowds.

3. COUNT Forest

Our problem is that of estimating a density map of target objects as shown in Fig. 1, given a set of training images with annotations. The annotation is a set of dots located at centers of object regions, as shown in Fig. 2. Dotting is a natural way for humans to count objects, at least when the number of objects is large, and it is less laborious than other annotation methods such as bounding boxes. For a training image I, we define the ground truth density function for each pixel $p \in I$ to be a kernel density estimated based on the dots provided

$$F^0(p) = \sum_{\mu \in \mathbf{A}} \mathcal{N}(p; \mu, \sigma^2 \mathbf{1}_{2\times 2}), \tag{1}$$

where $\mathbf{A}$ is a set of user annotation dots and $\mathcal{N}(p; \mu, \sigma^2 \mathbf{1}_{2\times 2})$ denotes a 2D Gaussian kernel centered on each dot $\mu \in \mathbf{A}$ with a small variance ($\sigma = 2.5$ pixels).

Figure 2. User annotation dots.

We aim to learn a nonlinear mapping $\mathcal{F}$ between patch features v and locations l of all objects inside each image patch relative to the patch center

$$\mathcal{F} : v \in \mathbf{V} \rightarrow l \in \mathbf{L}, \tag{2}$$

where $\mathbf{V}$ is the feature space, and $\mathbf{L}$ is the label space. The patch feature v is a vector concatenating responses of feature channels at every pixel inside the image patch. We augment each image patch with multiple additional filter channels, resulting in a feature vector $v \in \mathbb{R}^{h \times w \times C}$ where C is the number of channels and $h \times w$ is the patch size. A label l denotes a set of displacement vectors from the patch center O to all object locations P_j within its neighbour region of radius R:

$$l = \{\overrightarrow{OP_j} | P_j \in \mathbf{A} \wedge |\overrightarrow{OP_j}| < R\}. \tag{3}$$

Some samples of l are shown in Fig. 3. We propose a novel random forest framework, which we call *COUNT forest*, to learn the mapping $\mathcal{F}$ and use this forest regression from multiple image patches to vote for the densities of multiple target objects. The biggest advantages of our COUNT forest model is that it requires small memory space to build and reserve the forest, as it contains only vector labels at its leaf nodes. Compared with regression forest models with dense structured labels proposed in [20, 16], our model is more suitable for embedded computer vision systems.

3.1. Building COUNT Forest

Our COUNT forest is an ensemble of randomized trees that classify an image patch by recursively branching left or right down the tree in a coarse-to-fine manner until a leaf node is reached, as shown in Fig. 3.

Training of each tree proceeds as follows. Given a training set $\{(v_i, l_i)\}$, we assign all the training samples to the root node and recursively repeat splitting of the node until the number of training samples in a node becomes small or the height of the tree becomes large. For each node, we choose J splits randomly, each of which is a pair of a test function and a threshold. All data $S_j = \{(v_i, l_i)\}$ arriving at node j are split into a left subset $S_{L(j)}$ and a right subset $S_{R(j)}$ based on the thresholding result of the test function

$$\begin{aligned} S_{L(j)} &= \{(v_i, l_i) \in S_j | f_j(v_i) < t_j\} \\ S_{R(j)} &= S_j \setminus S_{L(j)}, \end{aligned} \tag{4}$$

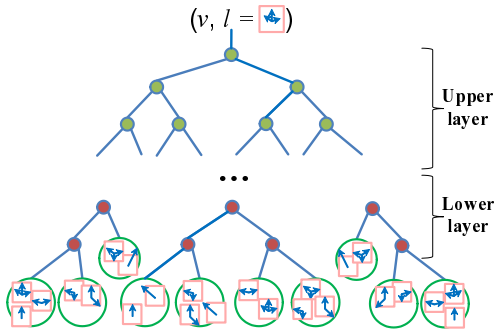

Figure 3. Training COUNT forest in a coarse-to-fine manner using two split layers. For space limit, only one tree is shown.

where $f_j(v)$ and t_j are the test function of the feature vector v and the threshold at a node j, respectively. In our implementation, $f_j(v)$ is a randomly selected element of the feature vector v.

Because our regression target is the vector label which differs by both the number of displacement vectors and their spatial distribution, we found that a single split criterion used in previous works is not enough to classify such labels. Our solution is to apply a coarse-to-fine approach where we separate the split nodes into *two split layers* with different split criterion. In the upper split layer, we attempt to split the labels based on their spatial distributions. Here, we find the best split (f_j^*, t_j^*) by minimizing the total variance

$$(f_j^*, t_j^*) = \underset{(f_j^n, t_j^n)}{\operatorname{argmin}} \sum_{o \in S_{L(j)}} ||\mathbf{H}^o - \bar{\mathbf{H}}^L||_F^2 + \sum_{o \in S_{R(j)}} ||\mathbf{H}^o - \bar{\mathbf{H}}^R||_F^2, \tag{5}$$

where $\mathbf{H}^o$ is a 2D histogram representing the spatial distribution of object locations indicated by the label, as shown in Fig. 4. Each histogram bin of $\mathbf{H}$ is computed from contributions of nearby object locations via Gaussian kernels. $\bar{\mathbf{H}}^L$ and $\bar{\mathbf{H}}^R$ are the average histogram of $\mathbf{H}$'s belonging to the left subset $S_{L(j)}$ and the right subset $S_{R(j)}$ respectively after splitting. $||\cdot||_F$ is the Frobenius norm. When the maximum depth or the minimum node size of this split layer is reached, patches are then passed to the lower split layer.

In the lower split layer, we attempt to make finer splits based on the label size, which is defined as the number of displacement vectors, *i.e.* the number of object locations, indicated by each label. We use the class uncertainty measure as the split function

$$(f_j^*, t_j^*) = \underset{(f_j^n, t_j^n)}{\operatorname{argmin}} \frac{|S_{L(j)}|}{|S_j|} \mathcal{H}(S_{L(j)}) + \frac{|S_{R(j)}|}{|S_j|} \mathcal{H}(S_{R(j)}), \tag{6}$$

where $\mathcal{H}(S)$ is defined as the Shannon entropy of distribution of label sizes in S. As a result, the uncertainty of class distribution of all labels associated with a node decreases with increasing tree depth, and each leaf should contain a set of labels with almost the same label size.

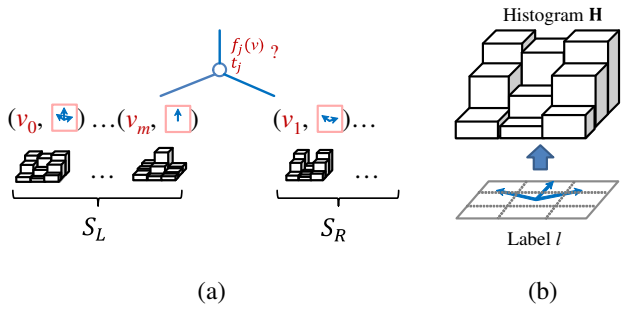

Figure 4. Splitting criterion of the upper split layer.

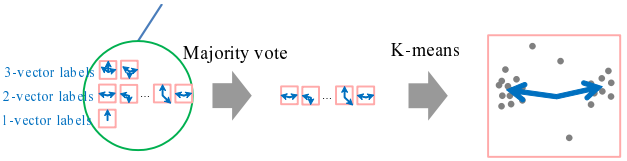

Figure 5. Prediction model by majority voting and K-means. In this sample, $K = 2$.

We must notice that in the test phase, we do not need to distinguish the two split layers because we use the same testing criterion for every node.

3.2. Prediction Model

In this section, we explain how to predict the output label of the above-mentioned randomized tree using the labels stored in the leaves during training time. An illustration of this prediction model is shown in Fig. 5.

After the training process stated in Section 3.1, training samples are stored in leaf nodes of the forest. We divide all the labels stored in each leaf node into clusters with the same label size (see Fig. 5). Among these clusters of labels, we select the cluster G having the largest number of members. As stated in the previous section, labels in each leaf node have almost the same size, therefore G contains most labels in the leaf and the remaining labels can be considered as noises. We assume that the labels in G have the same label size K. Note that K varies with the labels stored in each leaf. In Fig. 5, we show the case when $K = 2$. In the next step, we project the object locations indicated by all the labels in G on a plane whose center corresponds with each patch center of the labels, and apply K-means to obtain K clusters of such locations. Finally, we create a new label consisting of K displacement vectors from the plane center to the K centroids of these clusters. We call it the prediction label, and substitute it for all labels in the same leaf node to make the final trained tree. The built forest consists of only vector labels in its leaf node, and therefore *its memory size is much smaller* than regression forest models with dense structured labels proposed in [20, 16].

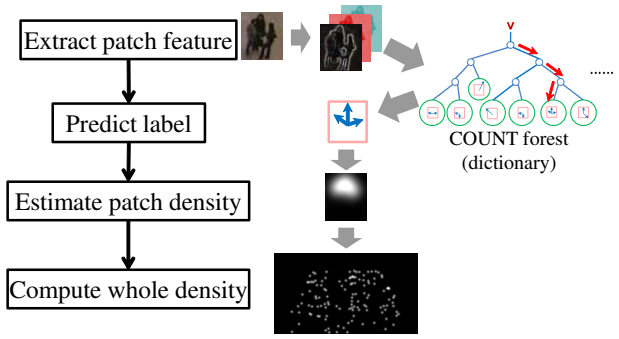

Figure 6. Density estimation procedure.

3.3. Density Estimation by COUNT Forest

In Fig. 6, we explain the procedure of estimating object densities using the trained COUNT forest. The procedure starts by extracting all patches from the input image, and a feature vector v_i from each patch i is computed. We then classify this vector v_i by recursively branching left or right down each tree T_j in the learned forest until a leaf node is reached, and obtain a prediction label l_{ji} stored in this leaf node. In the standard voting procedure [17], each label votes a delta peak for each object location and the vote counts accumulated in each pixel are then Gaussian-filtered to obtain a voting result map. Because each false predicted object location will increase the error of object counts by 1, we can not use this this voting map as our desired density map. Applying non-maxima suppression and counting from the object detection results does not work either because objects are so closed to each other and often partly occluded in crowded scenes (see Fig. 1).

Borrowing the idea of neighbour smoothing in [20], we suppress the error in estimating a density for each pixel by collecting predictions from neighbouring pixels. We calculate a patch density map from a predicted label and average it across trained trees and across neighbour patches. For details, each object location indicated by a label l_{ji} contributes to the patch density map by a Gaussian kernel, following the definition of the density function in (1):

$$D_{ji}(x) = \sum_{\mu \in l_{ji}} \mathcal{N}(x; \mu, \sigma^2 \mathbf{1}_{2\times 2}), \tag{7}$$

where x is an arbitrary pixel inside patch i, and μ is an object location indicated by label l_{ji}. Note that this patch density map has the same size as the input image patch. For fast computation, we precompute the Gaussian kernels $\mathcal{N}(x; \mu, \sigma)$ and store them in a lookup table indexed by $(x - \mu)$. The final density map of patch i is computed by averaging D_{ji} over the set of trees $\mathcal{T}$ of the trained forest:

$$D_i(x) = \frac{1}{|\mathcal{T}|} \sum_{T_j \in \mathcal{T}} D_{ji}(x). \tag{8}$$

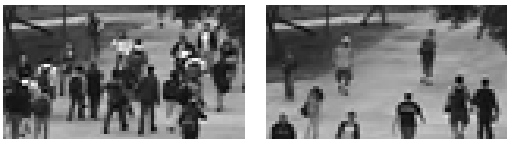

Figure 7. Crowded image patches (left) and non-crowded ones (right) differ largely in appearance and shape.

The density map for the whole image is computed by averaging all the predicted overlapping density maps $D_i(x)$:

$$D(x) = \frac{1}{|\mathcal{D}(x)|} \sum_{D_i \in \mathcal{D}(x)} D_i(x), \tag{9}$$

where $\mathcal{D}(x)$ is the set of density maps that contain pixel x in their scope. The average computations in equations (7)-(9) smooth the density predicted at each pixel by incorporating neighbour pixel information, and therefore the estimation error at a single pixel can be suppressed.

4. Robust Density Estimation

In this section, we use the COUNT forest described in the previous section to develop a robust density estimator with three improvements: increasing accuracy with a crowdedness prior, speeding up estimation by an effective forest reduction method, and decreasing annotation work by semi-automatic training.

4.1. Crowdedness Prior

In Fig. 7, we show two examples of crowded and non-crowded scenes. As seen in the figure, there is a large variation in appearance and shape between crowded image patches and non-crowded ones. Therefore, learning a forest from the entire training set including both crowded and non-crowded scenes should be harder than learning different forests for each scene. We use this crowdedness prior, which is a global property of the image, to develop a robust density estimator.

We aim to estimate a density D_i^n of an image patch i in the current frame, given the density D_i^{n-1} estimated from the previous frame. By introducing the crowdedness prior c, the probability $p(D_i^n|D_i^{n-1})$ can be estimated as

$$P(D_i^n|D_i^{n-1}) = \sum_{j=1,2} P(D_i^n|c_j, D_i^{n-1}) P(c_j|D_i^{n-1}), \tag{10}$$

where c_1 and c_2 denote the crowded and non-crowded properties. The probability $p(D_i^n|c_j, D_i^{n-1})$ is independent on D_i^{n-1} and can be learned by training a COUNT forest on different training subsets. We collect crowded patches and non-crowded patches to train a different forest for c_1 and c_2. A patch i is called crowded if the number $\mathcal{N}_i$ of objects in its neighbourhood is inside the range $[1/3\mathfrak{N}, \mathfrak{N}]$, and non-crowded if $\mathcal{N}_i \in [0, 2/3\mathfrak{N}]$, where $\mathfrak{N} = \max \mathcal{N}_i$. The prior probability $p(c_j|D_i^{n-1})$ is defined as

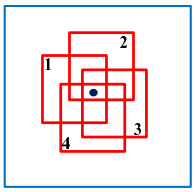

Figure 8. The density map for each red patch is predicted using a different sub-forest, therefore the density at the center dot is computed from all 4 sub-forests.

$$P(c_1|D_i^{n-1}) = [\mathcal{N}_i^{n-1} > \mathfrak{N}/2], \tag{11}$$

$$P(c_2|D_i^{n-1}) = 1 - P(c_1|D_i^{n-1}). \tag{12}$$

The motivation for these simple definitions is that we need to process *only one forest* to estimate a density map for a patch. As a result, we can improve accuracy by using this crowdedness prior without increasing the calculation cost.

4.2. Forest Permutation

As other regression forest based methods, the speed of our density estimation method depends on the the number of trees to be loaded and the sampling stride, *i.e.* the distance between sampled image patches. A trivial approach of fixedly reducing the forest could affect the accuracy heavily due to the weak ensemble of few decision trees.

We proposed a non-trivial approach of reducing the forest using permutation of decision trees. We divide the original forest into several sub-forests (typically 4), and circularly shift these sub-forests whenever moving to the next patch (*e.g.* $1234 \rightarrow 2341 \rightarrow ...$). We then modify the formulation (8) as follows:

$$D_i(x) = \frac{1}{|\mathcal{T}_{i,1}|} \sum_{T_j \in \mathcal{T}_{i,1}} D_{ji}(x), \tag{13}$$

where $\mathcal{T}_{i,1}$ is the first sub-forest after the forest permutation at patch i. Our idea is based on the fact that a density at a pixel is calculated by averaging predicted patches across trained trees and across neighbour locations. As shown in Fig. 8, although each density patch is predicted using only one sub-forest, a density at each pixel is eventually computed from the whole forest which appears partly in the surrounding patches.

4.3. Semi-Automatic Training

In this section, we address another common problem of counting-by-regression methods: manual annotation. As in other methods [10, 11, 21, 16, 9], we also require a different set of annotated images to train the estimator for a specific scene. Annotating dozens of images is laborious work, even in the case of a simple task such as marking the head of each person as shown in Fig. 2.

Figure 9. Segmentation masks obtained by GrabCut [26].

To deal with this problem, we introduce a semi-automatic training method to learn the estimator for a specific scene, making our technique more practical. Instead of selecting and annotating real images, we synthesize training images from a large set of segmented human regions and the target scene background. The synthesized training samples not only facilitate labor-saving, but also adapt our estimator to the new scene. Moreover, since we deal with image patches in crowded scenes with small object sizes, we do not need high qualities of synthetic data. As shown in our experiments in Section 5, our COUNT forest model is sufficiently robust to take a simple synthesized data as the training samples to obtain a high performance.

In our implementation, we perform image segmentation on the PETS2009-S2 dataset [15] including 795 images with bounding-box annotations, by using the GrabCut method [26]. We obtain more than 2000 segmentation masks, as shown in Fig. 9. The synthesizing process starts by pasting the random segmentation masks at random positions in the target scene background. We then fix the mask size according to the perspective scale of the scene, and synthesize shadows to make the final augmented images.

5. Experiments

The parameters of the our COUNT forest are set as follows: the number of trees $|\mathcal{T}| = 32$, the tree depth $h_{max} = 11$, the maximum depth for the upper split layer $h^1_{max} = 8$, and the minimum split size $n_{min} = 20$. The patch size is fixed to 13×13 pixels. In our training process, we extracted 1000 patches from each training image to build the set of training samples. We used the following feature channels to compute the feature vector v: the raw image, the background subtraction result, the temporal derivative, the Gaussian gradient magnitude, the Laplacian of Gaussian, and the eigenvalues of the structure tensor at different scales 0.8, 1.6. 3.2 (here we used the covariation matrix of derivatives over the pixel neighbourhood). To account for the perspective distortion, we multiplied all feature values with the square of the provided camera perspective map.

We use the same metrics as conventional works [11, 10, 23] for evaluating counting performance: mean absolute error $mae = E(|\kappa_j - \hat{\kappa}_j|)$, mean squared error $mse = E((\kappa_j - \hat{\kappa}_j)^2)$, and mean deviation error $mde = E(|\kappa_j - \hat{\kappa}_j|/\kappa_j)$, where κ_j and $\hat{\kappa}_j$ are the true number and the estimated number of objects in frame j, respectively. $\hat{\kappa}_j$ is computed as the sum of estimated densities over the whole image.

Data	Frame	Resolution	FPS	Count	Total
UCSD	2000	238×158	10	11-46	49885
Mall	2000	640×480	2	13-53	62325

Table 1. Dataset details. From left to right: dataset name, number of frame, resolution, frame per second, minimum and maximum number of people in the ROI, total number of people instances.

(a) (b)

Figure 10. Datasets used in our experiments: (a) Mall [11], (b) UCSD [9]. Upper row: input image, lower row: density map. The red border line indicates the ROI for density estimation.

5.1. Counting Performance

We evaluate the performance of counting people on the two public dataset: UCSD [9] and Mall [11]. Both datasets are provided with dotted ground truth [2, 1]. The details of both dataset are shown in Tab. 1. Sample images from the two datasets and the corresponding density maps estimated by our method are shown in Fig. 10.

In the first experiment, we used the same experiment setting as [11, 10, 23]. For the UCSD dataset, we employed 800 frames (600-1400) for training and the rest (1200 frames) for testing. For the Mall dataset, the first 800 frames were used for training and the remaining 1200 frames for testing.

We perform comparison against conventional methods: recent pedestrian detetor (Detector [6])[1], Least Square Support Vector Regression (LSSVR [27]), Kernel Ridge Regression (KRR [4]), Random Forest Regression (RFR [22]), Gaussian Process Regression (GPR [9]), Ridge Regression (RR [11]), Cumulative Attribute Ridge Regression (CA-RR [10]), Semi-Supervised Regression (SSR [23]), Maximum Excess over SubArrays (MESA [21]). The accuracy comparison results of counting people for the two dataset are shown in Tab. 2. We achieved the state-of-the-art performance on all three metrics, and improved the mean absolute error (mae) by 27% relative to the best previous result on the Mall dataset [10].

We also perform comparison on the UCSD dataset with other four different training/testing splits of the data ('max', 'down', 'up', 'min') as used in [21], and show the result in

[1]The pedestrian detector did not work for the UCSD dataset due to small sizes of people.

	Mall[11]			UCSD[9]		
Method	*mae*	*mse*	*mde*	*mae*	*mse*	*mde*
Detector([6])	20.55	439.1	0.641	-	-	-
LSSVR([27])	3.51	18.2	0.108	2.20	7.3	0.107
KRR([4])	3.51	18.1	0.108	2.16	7.5	0.107
RFR([22])	3.91	21.5	0.121	2.42	8.5	0.116
GPR([9])	3.72	20.1	0.115	2.24	8.0	0.112
RR([11])	3.59	19.0	0.110	2.25	7.8	0.110
CA-RR([10])	3.43	17.7	0.105	2.07	6.9	0.102
SSR([23])	-	17.8	-	-	7.1	-
Ours	**2.50**	**10.0**	**0.080**	**1.61**	**4.4**	**0.075**

Table 2. Comparison on the Mall dataset [11] and the UCSD dataset [9]. We achieved the best results on all three metrics.

Method	'max'	'down'	'up'	'min'
MESA [21]	1.70	1.28	1.59	2.02
RF [16]	1.70	2.16	1.61	2.20
Arteta *et al.* [5]	1.24	1.31	1.69	1.49
Ours	1.43	1.30	1.59	1.62

Table 3. Mean absolute errors in the UCSD dataset with four different training/testing splits of the data as used in [21].

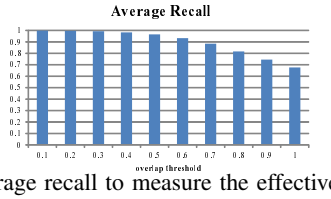

Figure 11. Average recall to measure the effectiveness of person displacement estimation in the UCSD dataset.

Tab. 3. We outperform [16] for all split settings and are comparable with Arteta *et al.* [5].

To evaluate the estimated displacements of pedestrians, we propose a recall function for each image which is the fraction of ground-truth person instances that the sum of estimated density inside its bounding box is larger than an overlap threshold. In Fig. 11, we show the average recalls (AR) over all frames with corresponding overlap thresholds. AR at threshold 50% is 96%, meaning that 96% of persons are detected closely to their correct locations.

5.2. Robustness

In Tab. 4, we present detailed evaluations of the COUNT forest and the proposed improvements in Section 4. The methods are compared in their accuracy, speed and memory cost. All experiments were performed using C++ on a PC with 2 Intel Xeon CPUs (2.80 GHz). We implemented the regression forest with dense structured labels proposed in [20] as our strong baseline[2]. Our COUNT forest gives a better accuracy and the memory cost for loading the forest is 30 times smaller. When applying the crowdedness prior, we further increase the accuracy at the cost of doubling the dictionary size, as we use two forests for estimation. In Fig. 12, we show the ground-truth and estimated pedestrian counts

[2]We used only the upper split layer with a larger maximum height 10 and stored a dense structured label in each leaf node expressing the averaged density patch calculated from the original vector labels.

Method	Error (*mae*)	Runtime	Dictionary Size
Baseline [20]	2.10	82ms	44.9MB
CF with 1 split layer	1.89	92ms	1.7MB
CF	1.59	96ms	**1.5MB**
CF + Prior	**1.43**	98ms	2.9MB
CF + Prior + Speedup	1.54	**36ms**	2.9MB

Table 4. Detailed evaluation on the UCSD dataset with the 'max' split setting (160 training, 1200 testing frames). CF: COUNT forest, Prior: crowdedness prior, Speedup: forest permutation.

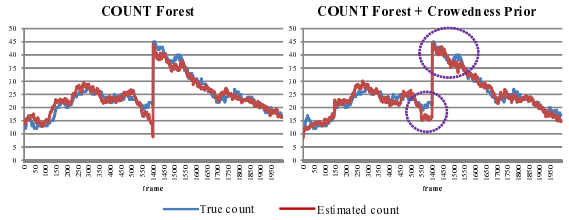

Figure 12. Counting results on the UCSD dataset using COUNT forest without and with the crowdedness prior. The right graph shows better estimation in extremely crowded and non-crowded scenes (highlighted regions).

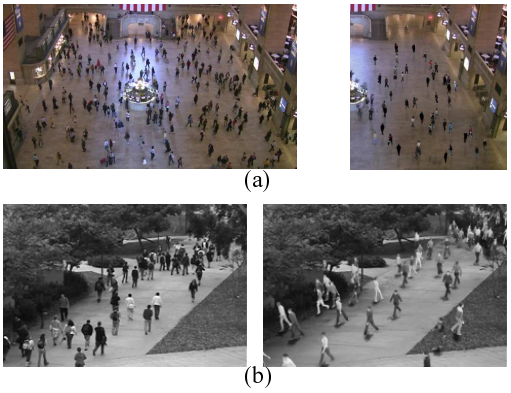

Figure 13. Real images (left) and synthesized images (right) from the Train Station dataset [31] (a) and UCSD dataset [9] (b).

by the two approaches. By adaptively switching between the two forests, we obtained better estimation in extremely crowded and non-crowded scenes.

We then applied the forest permutation approach in Section 4.2, where the sampling stride was set to 3 and number of sub-forest was 4. There was a small loss in accuracy, but we could achieve the real-time speed (30 fps). A trivial approach of fixedly reducing the forest also produced the real-time speed but with a lower accuracy ($mae = 1.60$).

5.3. Semi-automatic training

We used the Train Station dataset [31] to evaluate the performance of the semi-automatic training technique in Section 4.3. This dataset recorded over 100 persons traversing inside a train station. Sample synthesized training images are shown in Fig. 13.

The mean deviation errors of counting results with different numbers of synthesized images are shown in Fig. 14. The best result by manual training using the same number of training real images is also shown in the same graph. From

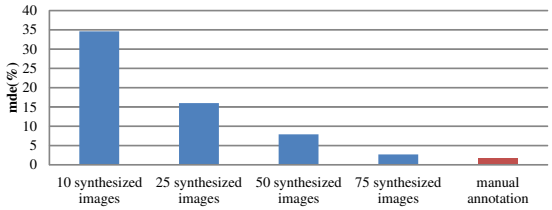

Figure 14. Counting results of training by manual annotation, and by the synthesizing method with different numbers of synthesized images. In this experiment, training with 75 synthesized images converged to the minimum mde (same result for 100 images).

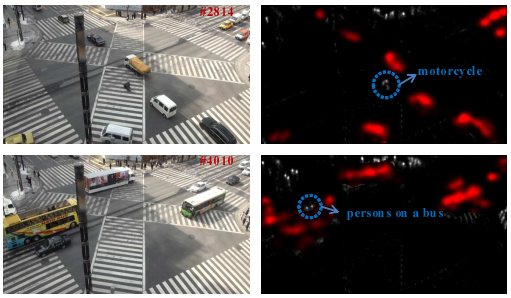

Figure 15. Pedestrian and car densities estimated for the Intersection dataset (person densities are in white, car densities are in red).

Methods	Pedestrian	Car
DPM [14]	17.7	3.4
Separated class counting	15.4	1.6
Simultaneous counting	6.6	1.4

Table 5. Mean absolute error (mae) of counting cars and pedestrians for the Intersection dataset.

the graph, we can observe that the counting performance is improved with an increasing number M of synthesized training images, and even get closed to the performance of the manual training if M is large enough.

5.4. Application 1: Traffic Count

We apply our method to solve the problem of simultaneous density estimation of pedestrians and cars in traffic scenes. It has many potential applications such as accident detection and traffic control. The main difficulty is that there are noises in person density estimation caused by misclassification of cars, because patch features of persons are similar to patch features of some car parts.

Our solution starts by making a copy of the input image for each person and car class, which we call the class layer. We then scale the car layer down by 4×4 times to match the car size to the person size. The advantage of this preprocessing is that it can filter out person appearances from the car layer, because person instances become so small after the resizing. As a result, noises of car density estimation caused by person misclassification can be reduced. We train a different COUNT forest for each class layer, and use these forests to estimate a density map for each object class. We then scale the car density map up to the original image size, and score each class at each pixel with a density smoothed

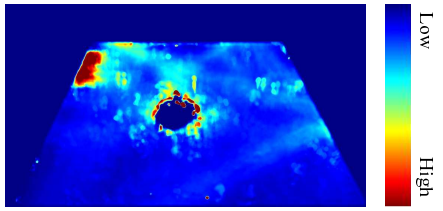

Figure 16. Averaged stationary time distribution over 4 hours.

over the neighbourhood region. For each location we take the maximally scoring class along with the corresponding density as output.

We introduce a new Intersection dataset to evaluate the performance of simultaneously counting pedestrians and cars. This video has a length of 11 minutes, with 30 fps frame rate, and 1920×1080 resolution. In this scene, cars and pedestrians traverse an intersection, making the scene become dense and sparse periodically. The number of pedestrians varies from 10 to 120, while the number of car varies from 0 to 20. We sampled 100 frames from the video with an interval of 100 frames, and used the first 50 frames for training and the remaining 50 frames for testing.

The density estimation results are shown in Fig. 15. We can observe that car and pedestrian regions are correctly classified even when they locate close together. Some interesting results are shown in the first two rows, in which a motorcycle and two persons on a bus are correctly detected. They prove the robustness of our method. The performance comparison results of counting cars and pedestrians for this dataset are shown Tab. 5. Compared with separated class counting, simultaneous counting improved the counting performance for both car and pedestrian classes. We also obtained better results than the Deformable Part Models (DPM) detector [14], which is the most used method for detecting multiple-class objects.

5.5. Application 2: Stationary Time

Our second application is the estimation of the stationary time, which is defined as a period that a foreground pixel exists in a local region allowing local movements [29]. The stationary time estimation can help scene understanding and provide valuable statistics computed over time. Besides, it was confirmed in [29] that simply detecting foreground at individual frames and computing how long a pixel has been in the foreground gave poor results. Our solution is to estimate a density map at each frame and compute how long a density at a pixel has been larger than a threshold. Although our method is much simpler than [29] without complex calculations, we obtained a similar result as [29]. In Fig. 16, we showed an averaged stationary time distribution in four hours of the Train Station dataset [31] (see Fig. 13(a)). As [29], it can be observed that stationary groups tend to emerge and stay long around the information booth and in front of the ticketing windows.

References

[1] Mall dataset. www.eecs.qmul.ac.uk/~ccloy/downloads_mall_dataset.html. 6

[2] Ucsd dataset. www.svcl.ucsd.edu/projects/peoplecnt/. 6

[3] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, 1997. 2

[4] S. An, W. Liu, and S. Venkatesh. Face recognition using kernel ridge regression. In *Computer Vision and Pattern Recognition*, pages 1–7, 2007. 6, 7

[5] C. Arteta, V. Lempitsky, J. A. Noble, and A. Zisserman. Interactive object counting. In *Proc. Eur. Conf. Comp. Vision*, 2014. 7

[6] R. Benenson, M. Omran, J. Hosang, and B. Schiele. Ten years of pedestrian detection, what have we learned? In *Proc. Eur. Conf. Comp. Vision, CVRSUAD Workshop*, 2014. 6, 7

[7] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 2

[8] G. J. Brostow and R. Cipolla. Unsupervised bayesian detection of independent motion. In *Proc. Comp. Vision Pattern Rec.*, pages 594–601, 2006. 1

[9] A. B. Chan, Z. S. J. Liang, and N. Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *Proc. Comp. Vision and Pattern Rec.*, pages 1–7, 2008. 1, 2, 5, 6, 7

[10] K. Chen, S. Gong, T. Xiang, and C. C. Loy. Cumulative attribute space for age and crowd density estimation. In *Proc. Comp. Vision Pattern Rec.*, pages 2467–2474, 2013. 1, 2, 5, 6, 7

[11] K. Chen, C. C. Loy, S. Gong, and T. Xiang. Feature mining for localised crowd counting. In *Proc. British Machine Vision Conf.*, pages 21.1–21.11, 2012. 1, 2, 5, 6, 7

[12] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool. Real-time facial feature detection using conditional regression forests. In *Proc. IEEE Comp. Vision and Pattern Rec.*, 2012. 2

[13] M. Dantone, J. Gall, C. Leistner, and L. Van Gool. Human pose estimation using body parts dependent joint regressors. In *Proc. IEEE Comp. Vision and Pattern Rec.*, 2013. 2

[14] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010. 8

[15] J. Ferryman. Pets 2009 dataset, 2009. http://www.cvg.rdg.ac.uk/PETS2009/. 6

[16] L. Fiaschi, R. Nair, U. Koethe, and F. Hamprecht. Learning to count with a regression forest and structured labels. In *Proc. Int. Conf. Pattern Rec.*, pages 2685–2688, 2012. 1, 2, 3, 4, 5, 7

[17] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *Proc. Comp. Vision Pattern Rec.*, pages 1022–1029, 2009. 2, 4

[18] J. Gall, N. Razavi, and L. Van Gool. On-line adaption of class-specific codebooks for instance tracking. In *British Machine Vision Conf.*, 2010. 2

[19] W. Ge and R. Collins. Marked point processes for crowd counting. In *Proc. Comp. Vision Pattern Rec.*, pages 2913–2920, 2009. 1

[20] P. Kontschieder, S. Rota Bulo, M. Pelillo, and H. Bischof. Structured labels in random forests for semantic labelling and object detection. *Trans. Pattern Anal. Mach. Intell.*, 36(10):2104–2116, 2014. 2, 3, 4, 7

[21] V. Lempitsky and A. Zisserman. Learning to count objects in images. In *Proc. Advances in Neural Information Processing Systems*, pages 1324–1332, 2010. 1, 2, 5, 6, 7

[22] A. Liaw and M. Wiener. Classification and regression by random forest. *R news*, 2(3):18–22, 2002. 6, 7

[23] C. C. Loy, S. Gong, and T. Xiang. From semi-supervised to transfer counting of crowds. In *Proc. Int. Conf. Comp. Vision*, pages 2256–2263, 2013. 1, 2, 6, 7

[24] R. Okada. Discriminative generalized hough transform for object dectection. In *Proc. Int. Conf. Comp. Vision*, pages 2000–2005, 2009. 2

[25] V. Rabaud and S. Belongie. Counting crowded moving objects. In *Proc. Comp. Vision Pattern Rec.*, pages 705–711, 2006. 1

[26] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 2004. 6

[27] T. Van Gestel, J. A. K. Suykens, B. De Moor, and J. Vandewalle. Automatic relevance determination for least squares support vector machine regression. In *Int. Joint Conf. Neural Networks*, pages 2416–2421, 2001. 6, 7

[28] A. Yao, J. Gall, and L. Van Gool. A hough transform-based voting framework for action recognition. In *Proc. Comp. Vision Pattern Rec.*, pages 2061–2068, 2010. 2

[29] S. Yi, X. Wang, C. Lu, and J. Jia. L0 regularized stationary time estimation for crowd group analysis. In *Proc. IEEE Comp. Vision and Pattern Rec.*, 2014. 8

[30] T. Zhao, R. Nevatia, and B. Wu. Segmentation and tracking of multiple humans in. *Trans. Pattern Anal. Mach. Intell.*, 30(7):1198–1211, 2008. 1

[31] B. Zhou, X. Wang, and X. Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents. In *Proc. Comp. Vision Pattern Rec.*, pages 2871–2878, 2012. 2, 7, 8