DeepMTL: Deep Learning Based Multiple Transmitter Localization

Caitao Zhan
Stony Brook University
cbzhan@cs.stonybrook.edu

Mohammad Ghaderibaneh
Stony Brook University
mghaderibane@cs.stonybrook.edu

Pranjal Sahu
Stony Brook University
psahu@cs.stonybrook.edu

Himanshu Gupta
Stony Brook University
hgupta@cs.stonybrook.edu

***Abstract*—In this paper, we address the problem of Multiple Transmitters Localization (`MTL`), i.e., to determine the locations of potential multiple transmitters in a field, based on readings from a distributed set of sensors. In contrast to the widely studied single transmitter localization problem, the `MTL` problem has only been studied recently in a few works. `MTL` problem is of great significance in many applications wherein intruders may be present. E.g., in shared spectrum systems, detection of unauthorized transmitters is imperative to efficient utilization of the shared spectrum.**

In this paper, we present `DeepMTL`, a novel deep-learning approach to address the `MTL` problem. In particular, we frame `MTL` as a sequence of two steps, each of which is a computer vision problem: image-to-image translation and object detection. The first step of image-to-image translation essentially maps an input image representing sensor readings to an image representing distribution of transmitter locations, and the second object detection step derives precise locations of transmitters from the image of transmitter distributions. For the first step, we design our learning model `sen2peak`, while for the second step, we customize a state-of-the-art object detection model `YOLOv3-cust`. We demonstrate the effectiveness of our approach via extensive large-scale simulations, and show that our approach outperforms the previous approaches significantly (by 50% or more) in accuracy performance metrics, and incurs an order of magnitude less latency compared to other prior works.

***Index Terms*—Localization, Wireless Sensors, Deep Learning, Image Translation, Object Detection**

I. Introduction

The RF spectrum is a limited natural resource in great demand due to the unabated increase in mobile (and hence, wireless) data consumption [1]. In 2020, the U.S. FCC moves to free up 100 MHz of previously military occupied mid-band spectrum in the 3.45-3.55 GHz band for paving the way of 5G development. Also, the research and industry communities have been addressing this capacity crunch via the development of *shared spectrum*. Spectrum sharing is the simultaneous usage of a specific frequency band in a specific geographical area and time by a number of independent entities where harmful electromagnetic interference is mitigated through agreement (i.e., policy, protocol) [2]. Spectrum sharing techniques are also normally used in 5G networks to enhance spectrum efficiency [3]. However, protection of spectrum from unauthorized users is important in maximizing spectrum utilization.

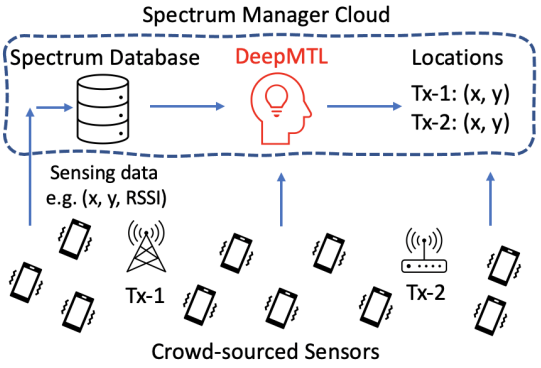

Fig. 1. Spectrum monitoring, multiple transmitter localization by a distributed set of sensors. `DeepMTL` is a deep-learning approach to multiple transmitter localization which helps protect spectrum against unauthorized usage.

The increasing affordability of the software-defined radio (SDR) technologies makes the shared spectrum particularly prone to unauthorized usage or security attacks. With easy access to SDR devices (e.g. HackRF, USRP), it is easy for selfish users to transmit data on shared spectrum without any authorization and potentially causing harmful interference to the incumbent users. Such illegal spectrum usage could also happen as a result of infiltration of computer virus or malware on SDR devices. [3] depicts 3 cases of spectrum attack. As the fundamental objective behind such shared spectrum paradigms is to maximize spectrum utilization, the viability of such systems depends on the ability to effectively guard the shared spectrum against unauthorized usage. The current mechanisms however to locate such unauthorized users (intruders) are human-intensive and time-consuming, involving FCC enforcement bureau which detects violations via complaints and manual investigation [4]. Motivated by the above, we seek an effective technique that is able to accurately localize multiple simultaneous intruders (transmitters). Below, we describe the multiple transmitter localization problem.

Multiple-Transmitter Localization (`MTL`). The transmitter localization problem has been well-studied, but most of the focus has been on localizing a *single* intruder at a time. However, it is important to localize multiple transmitters simultaneously to effectively guard a shared spectrum system. E.g., a malware or virus-based attachment could simultaneously cause many devices to violate spectrum allocation rules; spectrum jamming attacks would typically involve multiple transmitters. More

importantly, a technique limited by localization of a single intruder could then be easily circumvented by an offender by using multiple devices. The key challenge in solving the multiple-transmitter localization (MTL) problem comes from the fact that the deployed sensor would receive only a *sum* of the signals from multiple transmitters, and separating the signals may be impossible.

Prior Works on MTL. The MTL problem has been recently addressed in a few prior works, among which SPLOT [4], MAP* [5], and DeepTxFinder [6] are the most prominent. SPLOT essentially decomposes the MTL problem to multiple single-transmitter localization problems based on the sensors with the highest power readings in a neighborhood. However, their technique implicitly assumes a propagation model, and thus, may not work effectively in areas with complex propagation characteristics, and it is not effective in the case of transmitters being located close-by (a key challenging scenario for MTL problem). Our recent work MAP* solves the MTL problem using a hypothesis-driven Bayesian approach; in particular, it uses prior training in the form of distributions of sensor readings for various transmitter locations, and uses the training data to determine the most likely configuration (i.e., transmitters' locations and powers) for a given vector of sensor readings. However, to circumvent the high computational cost of a pure Bayesian approach, MAP* uses a divide and conquer heuristic which results in somewhat high number of misses and false alarms while still incurring high latency. DeepTxFinder uses a CNN-based learning model approach; however, they use a separate CNN-model for a specific number of transmitters and thus may incur high training cost while also limiting the number of transmitters that can be localized. In our evaluations, we compare our work with each of the above approaches.

Our Approach. As in prior works [4], [7], we assume a crowdsourced sensing architecture wherein relatively low-cost spectrum sensors are available for gathering signal strength in the form of received power. We use a convolutional NNs (CNNs) based approach to solve the MTL problem. In particular, we frame MTL as a sequence of two steps: image-to-image translation and object detection, each of which is solved using a trained CNN model. The first step of image-to-image translation maps an input image representing sensor readings to an image representing distribution of transmitter locations, and the second object detection step derives precise locations of transmitters from the image of transmitter distributions.

Motivation. Our overall approach and its various aspects are motivated by the following considerations. **First**, we use a learning-based strategy to preclude assuming a propagation model [4] or conducting surveys of sensors reading distributions [5]. Assumption of propagation model suffers from the fact that even sophisticated propagation models yield unsatisfactory accuracy and thus lead to degraded performance. Moreover, even though a learning-based approach incurs a one-time high training cost, it generally incurs minimal latency during evaluation, which is an important consideration for our MTL problem, a intruder detection should incur minimal latency to be effective. **Second**, the geographical nature of the MTL problem suggests that convolutional neural networks (CNNs) are well-suited for efficient learning of the desired function. In particular, the features of the MTL problem can be represented in a 2D array corresponding to their geographic locations, which can be fed as an input to an appropriate CNN model which can leverage the spatial correlation among the input features to facilitate efficient learning. **Lastly,** we use a two-step architecture to facilitate efficient training by essentially providing an additional intermediate target. In particular, we are able to map each step to well-studied standard computer vision problems, allowing us to build upon known techniques.

Overall Contributions. The overall goal of our work is to develop an efficient technique for accurate localization of simultaneously present multiple transmitters/intruders. In this context, we make the following specific contributions.

1) We develop DeepMTL, a novel two-step CNN-based approach, for the MTL problem that frames MTL as a sequence of image-to-image translation and object detection problems.
2) For the first step of image-to-image translation, we develop sen2peak, a CNN model, that translates an image representing sensor readings into a target image that encodes distributions of transmitter locations. For the second step, we customize an available object detection platform, YOLOv3, for our context, and develop YOLOv3-cust that is able to accurately localize transmitters in the image output by the first step.
3) We evaluate our techniques via large-scale simulations and demonstrate their effectiveness and superior performance compared to the prior works.

II. Background, Problem and Methodology

In this section, we describe the background of the shared spectrum systems, formulate the MTL problem, then describe our methodology.

Shared Spectrum System. In a shared spectrum paradigm, the spectrum is shared among licensed users (primary users) and unlicensed users (secondary users, SUs) in such a way that the transmission from secondaries does not interfere with that of the primaries (or secondaries from a higher-tier, in case of a multi-tier shared spectrum system). In some shared spectrum systems, the location and transmit power of the primary users may be unavailable, as is the case with military or navy radars in the CBRS band. Such sharing of spectrum is generally orchestrated by a centralized entity called *spectrum manager*, such as a spectrum database in TV white space [8] or a central spectrum access system in the CBRS 3.5GHz shared band [9]. The spectrum manager allocates spectrum to requesting secondaries (i.e., permission to transmit up to a certain transmit power at their location) based on their location, spectrum demand, configurations of the primaries, other active secondaries, prevailing channel conditions, etc. Users that transmit without explicit permission are referred

to as unauthorized users or *intruders*; the MTL problem is to essentially localize such intruders.

Problem Setting and Formal Definition. Consider a geographic area with a shared spectrum. Without loss of generality, we assume a single channel throughout this paper (multiple channels are handled similarly). For localization of intruders, we assume available crowdsourced sensors that can observe received signal in the channel of interest, and compute (total) received signal strength indicator (RSSI). At any instant, there may be a set of intruders present in the area with each intruder at a certain location transmitting with a certain power which may be different for different intruders. The MTL problem is to determine the set of intruders with their locations at each instant of time, based on the set of sensor observations at that instant. In this paper, for simplicity, as in almost all works [4], [6], [10] except [5], we assume only intruders present, if any, i.e., we ignore the presence of PUs and any authorized users, and thus, assume that the sensor readings represent aggregate received power from the transmitters we wish to localize. Note that presence of PUs and/or authorized users can easily be incorporated by using a technique to localize all (authorized as well as unauthorized) transmitters and then remove the (known) authorized ones to determine the remaining as unauthorized transmitters; this approach, only increases the total number of potential transmitters to be localized.

Methodology. In our context, each sensor communicates its observation to a centralized spectrum manager which then runs localization algorithms to localize any potential (multiple) transmitters. We design and implement a novel two-step localization algorithm named DeepMTL, as illustrated in Fig. 3, based on CNN models. The first step (Section III) is a four-layer image-to-image translation CNN model that is trained to translate an input image representing sensor readings to an image of transmitters' locations distributions. Each distribution of a transmitter can be visualized as a mountain with a peak, so we name this model sen2peak. The second step (Section IV), called YOLOv3-cust, is a customized object-detection platform build upon YOLOv3 [11] which localize the objects/peaks in the translated image. The high-level motivation behind our two-step design is to frame the overall MTL problem in terms of well-studied learning problem(s). The two steps facilitate efficient learning of the models by supplying an intermediate target with the training samples.

III. Step 1: Sensor Readings to TX Location Distributions

In this section, we present the first step of our overall approach to the MTL problem, i.e., the image-to-image translation step which translates/transforms the sensor reading to distributions of TX locations. Here, we first create a grayscale image to represent the input sensor readings; this image encodes both the sensors' RSSI readings and the sensors' physical location. We then train and use a convolutional neural network (CNN) model to transform this input image to an output image which represents the distribution of TX locations. Pixels in the output image that have higher values will have a higher chance of having a TX being present at that location.

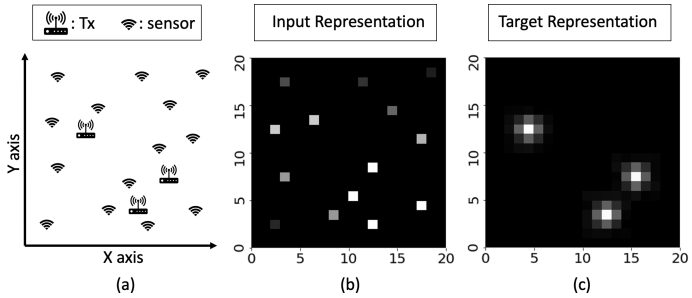

Fig. 2. First Step (Image to Image Translation) Input and Output Images. (a) Area with distributed sensors and transmitters to be localized. (b) Input image representing the sensors' (RSSI) readings and locations. (c) Output/Target Image, where we put a 2D Gaussian distribution with its "peak" at the transmitter's location.

A. Input Image Representing Sensors' Readings

We localize transmitters based on observations from a set of sensors. The input of the localization method is sensor observations. Here, an *observation* at an individual sensor is the received power over a time window of a certain duration, in the frequency channel of interest (we assume only one channel). The received power is computed from the FFT operation of the I/Q samples collected in a time window. An *observation vector* (used in [4], [5]) is a vector of observations from a set of distributed sensors, with each vector dimension corresponding to a unique sensor. However, the observation vector doesn't include the location of the sensors explicitly.

In our setting, to represent complete information about the sensors (locations and readings) in an image like input, we represent the observations of sensors in a form of an *observation matrix* $\mathbf{X}$, which encodes the location as well as the readings of the distributed sensors. In $\mathbf{X}$, the entry $\mathbf{X_{i,j}}$ denotes the observation of the sensor at location (i, j). If there is no sensor at location (i, j), we assign the noise floor $\mathcal{N}$ (i.e. -80 dBm) value to $\mathbf{X_{i,j}}$. Fig. 2(b) shows how a 20×20 observation matrix is used to represent the input information that contains both the RSSI and the spatial location of the distributed sensors from an area that exists 3 transmitters in Fig. 2 (a). Before we pass this observation matrix as input to our CNN model, we do a normalization step; we first subtract the $\mathcal{N}$ from each value and then divide it by $-\mathcal{N}/2$. This ensures that the value $\mathbf{X_{i,j}}$ is zero at locations without sensors, and for locations with senors, the value $\mathbf{X_{i,j}}$ is a positive number between zero and two. The input observation matrix $\mathbf{X}$ can be looked upon as a *gray-scale image*. The color image is a 3D matrix with 3 RGB channels, whereas the gray-scale image is a 2D matrix with only one channel.

B. Output Image with TX locations' Distribution

After devising the input representation, we now focus on designing the output image to represent the distribution of TX locations; the output image is essentially the "label" assigned to each input image that guides the training of the CNN model.

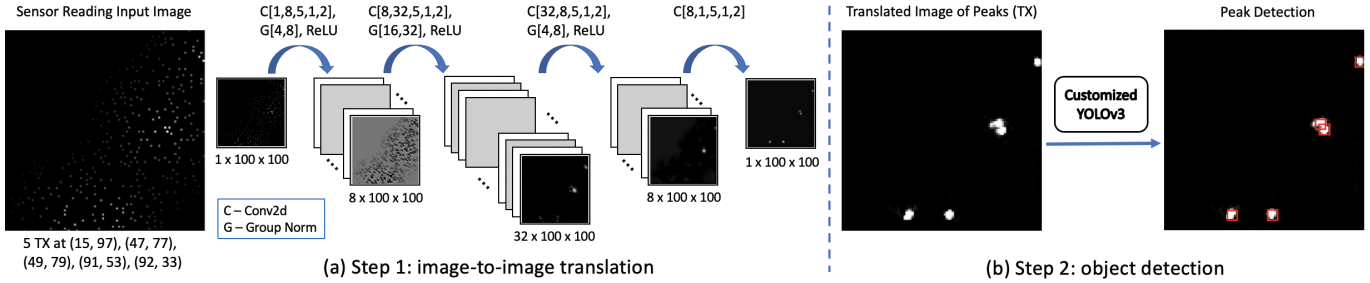

Fig. 3. The overall CNN architecture of the `DeepMTL` model. (a) Architecture of the first step, a four layer image-to-image translation model (`sen2peak`). The figure displays how the data volume flows through the various convolutional layers. C stands for Conv2d, and for each Conv2d layer, the five values shown are [input-channel, output-channel, kernel-size, stride, padding]. G stands for group normalization, and, for each group normalization, the two values shown are [number-of-groups, number-of-channels]. See §III for details. (b) The high-level idea of the second step that uses `YOLOv3-cust`, a customized version of YOLOv3, to perform object/peak detection in the output image of the first step. This step returns the precise location coordinates of TX.

A straight forward representation that represents the TXs with locations is to just use an array of (x, y) elements where each (x, y) element is the location of a transmitter, as in [6]. However, this simple representation is less conducive to efficient model learning, as the representation moves away from spatial representation (by representing locations as positions in the image) to explicit representation of locations by coordinate values. E.g., in [6]'s CNN-based approach to `MTL` problem, the authors assume a maximum number N of transmitters and train as many as $N+2$ different CNN models and thus, limiting the overall solution to the pre-defined maximum number of transmitters. Instead, in our approach, we facilitate the learning of the overall model, by solving the `MTL` problem in two steps, and in this step of translating sensors' reading to transmitter locations' distributions, we represent the output also as an image. This approach allows us to use a spatial learning model (e.g. CNN) for the second step too, and preclude use of regression or fully-connected layers in the first step.

Label/Target as a Grayscale Image. Inspired by a recent work on wireless localization problem [12] which represents the input and targets as images, we also choose to represent our target of the first step as a grayscale image. In the target image, we use 25 (5x5) pixel values to represent presence of a transmitter. It is desirable to use an odd side length square (i.e. 3x3, 5x5, 7x7) for symmetry. For a 100x100 size input we use, while 3x3 gives too less information for a target and 7x7 generates too many overlap for close-by targets, 5x5 is the sweet spot. Other pixels far away from any transmitter are zero valued. Among multiple potential ways to represent a transmitter presence by a number of pixels, we found that using a 2D Gaussian distribution around the pixel of TX location, Fig. 2(c), yields best model performance. Thus, a geographic area with multiple transmitters present is represented by a grayscale image with multiple Gaussian distributions, with each Gaussian distribution's peak inside the pixel corresponding to transmitter's location. Based on preliminary performance tests, we picked the amplitude of the 2D Gaussian peak to 10, the standard deviation to 0.9, and located the center of the distribution at the location of each transmitter. Note that the location of the TX is in continuous domain and usually not at the *center* of the grid cell.

C. `sen2peak`: Image-to-Image Translation Model

At a high-level, we use a deep and spatial neural network, in particular a CNN, to learn the approximation function that maps the input image (of sensor readings) to the target image (of Gaussian distributions for TX locations). We refer to this as the *image-to-image translation* model. Our approach is inspired by the recent work [12] that frames a different wireless localization problem as an image-to-image translation problem. We incorporate the idea into our multiple transmitter localization problem and utilize recent advances in the computer vision area. Encoder-decoder based CNN models like U-Net [13] with down-sampling and up-sampling convolutional layers have been successful in effectively learning image-to-image translation functions. However, in our setting, we observe that the usage of down-sampling layers (such as max-pooling) degrades the performance of the model, especially in the case when transmitters may be close to each other wherein the model is unable to distinguish the near-by transmitters and generate a single large distribution in the target image. To circumvent this, we avoid using any down-sampling layers in our model and redesign the image-to-image translation model as described below.

Our `sen2peak` Model. We refer to our image-to-image translation CNN model as `sen2peak`, as it translates sensors' readings to "peaks" with Guassian distributions corresponding to transmitter locations. It has four [1] convolutional layers, as shown in Fig. 3(a). We use an input size of 100x100. The number of convolutional filters are varying for different layers, with up to 32 in one of the layers. We tried doubling the filter numbers at each layer, but it doesn't lead to significant improvement (it does yield a lower error, but the target image doesn't improve significantly to impact the second step of our architecture). We use a kernel size of 5x5, a stride of 1, and a padding of 2. This ensures that the dimensions don't decrease and all the pixels are treated uniformly, including the ones at the edge of the image. With the above four convolutional layers, the receptive field [14] of each neuron in the output layer is 17x17. Normalization layers can improve the learning process. We chose group normalization [15] and put it after the

[1] We observe that a four layer lightweight and symmetric `sen2peak` model produces good results and adding more layers gives marginal improvement.

first three convolutional layers. We compared group and batch normalization [16] methods in our context, and observed better performance with the group normalization. For the activation layers, we select rectified linear unit (ReLU) and put it after the group normalization layers.

The Loss Function. Our inputs (X) and targets (Y) are images. We use L2 loss function which computes the mean squared error aggregated over individual pixels. More formally, our loss function is defined as:

$$\frac{1}{N}\sum_{i}^{N}||\texttt{sen2peak}(X_i) - Y_i||^2 \quad (1)$$

where N is the number of samples used in computing the loss, $||\cdot||^2$ is L2 loss function, X_i and Y_i are the i_{th} sample's input and target images respectively, and $\texttt{sen2peak}(X_i)$ is the predicted target image corresponding to the input X_i. During training, we use Adam [17] as the optimizer that minimizes the loss function. We set the learning rate to 0.001 and number of epochs to 20 and the model converges well.

IV. Step 2: TX Locations' Distributions to Precise Locations

In this section, we present the second step of our overall localization approach. We refer to this step as the *peak detection* step, as the goal is to detect the peaks within the Gaussian distributions in the input image (which is also the output image of the first step). The first step outputs an image that has multiple distributions (presumably, Gaussian), whose peaks need to be interpreted as precise locations of the transmitters/intruders. As, our end goal is to determine the precise locations of the present transmitters, we develop techniques to detect peaks within the output image of the first step. We propose two different strategies for the peak-detection task. The first strategy is a straightforward peak detection algorithm based on finding local maximal values, while the second strategy is based on framing the problem as an object detection task; for the second strategy, we utilize a widely used state-of-the-art computer vision model called YOLOv3 [11].

A. Simple Peak Detection Method: `simplePeak`

The simple and straight forward peak detection method is to search for pixels with locally maximal values, based on predetermined threshold values. In particular, we use thresholds for peak value and the radius of the distribution (i.e., the range of pixels with non-zero values around a peak). More specifically, we characterize a pixel as a peak if its value is more than the peak threshold (we choose 2) value and if its value is maximum among pixels within the radius threshold (we choose 3). The above approach only yields a pixel as the peak, which implicitly assigns the *center* of the area corresponding to the pixel to the peak/transmitter location. However, in practice, a peak may represent a large enough area that we may wish to localize the transmitter more precisely *within the grid cell* corresponding to the pixel representing the peak. We achieve this (i.e., localizing transmitters precisely within the grid cell, rather than just assume them to be at the cell's center) by computing a weighted average of the peak pixel's coordinate and the peak's neighbor pixels' coordinates. The weights are the pixel value themselves. With the above improvement, `simplePeak` returns a list of locations in the continuous domain for the detected transmitters. We use `DeepMTL-peak` to denote the architecture consisting of `sen2peak` and `simplePeak`.

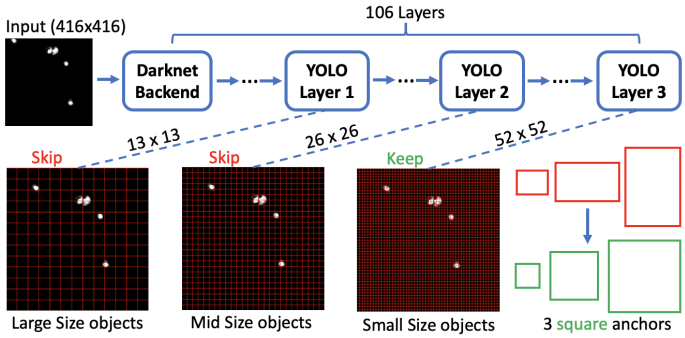

Fig. 4. Our `YOLOv3-cust`. The two major customizations are: (i) Use only the last YOLO layer that detects small size objects, and (ii) change the rectangle anchors to square anchors.

B. CNN Object Detection Method: `YOLOv3-cust`

The simple hand-crafted method described in the previous subsection performs reasonable well in most cases in our simulations. However, it's key drawback is that it needs appropriate threshold values that may vary from case to case; such thresholds can be difficult to determine, especially since the input images (with distributions) are not expected to be perfect as they are themselves output of a learning model. Inaccurate threshold values can lead to false alarms and misses. Also, the previous method is not sufficiently accurate at the sub-pixel level, where each pixel may represent a large area such as 10m by 10m or even 100m x 100m. Thus, we propose a CNN-based learning method that overcomes the above shortcomings.

We frame this problem as an object detection task where the objective is to detect and localize known objects in a given image. We observe that our second-step peak detection problem is essentially an object detection problem where the "object" to detect is a "peak". Thus, we turn the `MTL` problem of localizing multiple transmitters into detecting peaks in the images output by `sen2peak` model. For object/peak detection, we design `YOLOv3-cust`, our customized version of YOLOv3 [11]. Fig. 5 is a zoom-in of localizing two close by transmitters (peaks) in Fig. 3 (b).

Peak Detection Using `YOLOv3-cust`. Object detectors are usually comprised of two parts: (i) a backbone which is usually pre-trained on ImageNet, and (ii) a front part (head), which is used to predict bounding boxes of objects, probability of an object present, and the object class. For the front part, object detectors are usually classified into two categories, i.e., one-stage detectors such as the YOLO [18] series, and two-stage detectors such as the R-CNN [19] series. We choose the one-stage YOLO series because of its computational efficiency,

low latency, high popularity and available ways to customize it for our specific context. We refer to the customized version as `YOLOv3-cust`, see Fig. 4. Implementing a 106-layer deep neural network with a complex design from scratch is out of scope of our work. Thus, we use a publicly available source repository [20] and made customization on top of it. We refer to the architecture that uses `sen2peak` and `YOLOv3-cust` in sequence as `DeepMTL`, our key product. In addition, we also use `sen2peak` in combination with the uncustomized original YOLOv3, and refer to it as `DeepMTL-yolo` (we still change the class number to 1).

Customization of YOLOv3. Overall, we incorporated four customization to YOLOv3, of which two are significant and the other two are relatively minor. See Table I. YOLOv3 is designed to be a general object detector that can detect objects of various sizes, shapes, and classes within input images of various sizes. However, for our context, we can take advantage of the special nature of our context. E.g., the input image (the output of `sen2peak`) dimension is fixed at 100x100 pixels. Also, the objects/peaks to be detected in the image are relatively small objects, and usually can be bounded by a square. Moreover, there is only one class of objects. Based on the above observations, we make changes to the original YOLOv3 that both decrease the model complexity and improve its performance.

Table I. Differences between the original YOLOv3 and our `YOLOv3-cust`.

YOLOv3	`YOLOv3-cust`
Has three YOLO layers at 13x13, 26x26, and 52x52 for detection	Only use the last 52x52 YOLO layer for detection (skip the first two YOLO layers)
Has 3 different rectangle anchors for each YOLO layer	Has 3 square anchors
Every 10 batches, randomly chooses a new input image dimension size	Don't randomly choose new input dimension size
Has 80 different categories of object class	Only has one category for the peak class

Customization Details. The first and second changes presented in Table I are major changes and we elaborate on them in the following paragraphs. Making prediction at three different scales is one of the highlights of YOLOv3 and an improvement comparing to the previous version YOLOv2 which was prone to missing at detecting small objects. As shown in Fig. 4, the coarse-grain 13x13 YOLO layer-1 is designed for detecting large size objects, the 26x26 YOLO layer-2 is designed for detecting middle-sized objects, and the fine-grained 52x52 YOLO layer-3 is designed for detecting small-sized objects. Since the peaks in our translated images are always small objects, we only use the last 52x52 YOLO detection layer (and skip the first two YOLO layers). As shown in Fig. 4, by "skipping" the two YOLO layers means that we do not use them in computing the overall loss function and their outputs are not used in predicting the bounding boxes. In our customized `YOLOv3-cust`, the only YOLO layer predicts 8112 bounding boxes, since it has a dimension of 52x52 and each cell results in prediction of 3 bounding boxes; this is in contrast to the original YOLOv3, which predicts 10647 bounding boxes ($3 \times (13 \times 13 + 26 \times 26 + 52 \times 52) = 10647$).

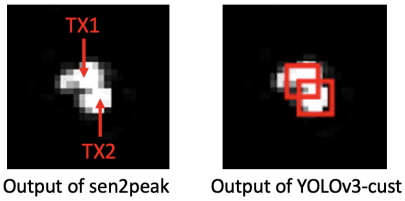

Fig. 5. A zoom-in of two close peaks in Fig. 3(b), to show a clearer understanding of the mechanisms in `sen2peak` and `YOLOv3-cust`.

The anchor box is one of the most important hyperparameters of YOLOv3 that can be tuned to improve its performance on a given dataset. The original YOLO's anchor boxes are 10x13, 16x30, and 33x23 (for the input image of size 416x416 pixels), which are essentially bounding boxes of a rectangular shape. These original YOLOv3 anchors were designed for the Microsoft COCO [21] data set, and were chosen since they best describe the dimensions of the real world objects in the MS COCO data set. In our context, since the peaks are generally squares—we use the anchor boxes to be 15x15, 25x25, and 35x35.

Training Dataset. The label for an object is a 5 values tuple (class, x, y, width, height), where `class` is always the same, x and y are the location coordinates of the transmitter, width and height determine the size and shape of the object—which we consistently set to be 5 each to signify a 5x5 square. Recall that the size of the first step's `sen2peak` output image (also the input to the second step) is 100x100; we resize the image to 416x416 and make three copies of it to yield a 3x416x416 3D matrix, so that we can feed this data directly into the `YOLOv3-cust` model.

V. Evaluation

To evaluate the performance of our proposed techniques, we conduct large-scale simulations over two settings based on two different propagation models. In particular, we consider log-distance based propagation model and the Longley-Rice model obtained from SPLAT! [22]. We evaluate various algorithms, using multiple performance metrics as described below.

Performance Metrics. We use the following metrics to evaluate the localization algorithms.

1) Localization Error ($\mathrm{L_{err}}$)
2) Miss rate ($\mathrm{M_r}$)
3) False Alarm rate ($\mathrm{F_r}$)

Given a multi-transmitter localization solution, we first compute the $\mathrm{L_{err}}$ as the minimum-cost matching in the bi-partite graph over the ground-truth and the solution's locations, where the cost of each edge in the graph is the Euclidean distance between the matched ground-truth node location and solution's node location. We use a simple greedy algorithm to compute the min-cost matching. The unmatched nodes are regarded as false alarms or misses. We put an upper threshold on the cost of an eligible match. E.g., if there are 4 intruders in reality,

but the algorithm predicts 6 intruders then it is said to incur 0 misses and 2 false alarms, so the M_r is 0 and the F_r is 2/6. If it predicts 3 intruders then it incurs 1 miss and 0 false alarms, so the M_r is 1/4 and the F_r is 0. In the plots, we stack the miss rate and false alarm rate to reflect the overall performance.

Algorithms Compared. We implement[2] and compare six algorithms in two stages. In stage one, we compare three versions of our techniques, viz., `DeepMTL`, `DeepMTL-yolo` and `DeepMTL-peak`. Recall that `DeepMTL`, `DeepMTL-yolo`, and `DeepMTL-peak` use `sen2peak` in the first step, and `YOLOv3-cust`, original YOLOv3, and `simplePeak` respectively in the second step. In the first stage of our evaluations, we will show that `DeepMTL` outperforms `DeepMTL-yolo` and `DeepMTL-peak` in almost all performance metrics. Thus, in the second stage, we only compare `DeepMTL` with schemes from three prior works, viz., `SPLOT` [4], `DeepTxFinder` [6], and `MAP*` [5] and show that `DeepMTL` outperforms the prior works.

Training and Testing Dataset. We consider an area[3] of $1km \times 1km$, and use grid cells (pixels) of $10m \times 10m$, so the grid is 100x100. The transmitters may be deployed anywhere within a cell (i.e., their location is in the continuous domain), while the sensors are deployed at the centers of the grid cells. For each instance (training or test sample), the said number of sensors and transmitters arew deployed in the field randomly. For each of the two settings (propagation models described below), we create a 100,000 sample training dataset to train our models, and create another 20,000 sample testing dataset to evaluate the trained model.

We will evaluate the performance of various techniques for varying number of transmitters/intruders and sensor density. When we vary a specific parameter, the other parameter is set to its *default* value; the number of transmitters varies from 1 to 10 and the default value is 5; the sensor density varies from 2% to 10% and the default value is 6% (600 sensors in a 100x100 grid). When not mentioned, the default values are used. The transmitter power varies from 0 to 5 dBm and is randomly picked. To minimize overfitting, the training dataset and testing dataset have sensors placed at completely different locations.

We train the `DeepMTL` model using the 100,000 sample dataset. To train `DeepTxFinder` [6], we partition the 100,000 sample training dataset into 10 datasets based on the number of transmitters in the samples which varies from 1 to 10. These 10 datasets are used to train the 10 "localization" CNN models in `DeepTxFinder`, and the full dataset of 100,000 samples is used to train the `DeepTxFinder` model that determines the number of transmitters. For the `MAP*` scheme [5], we assume availability of all required probability distributions. We note that using a simple cost model (number of samples need to be gathered), the overall training cost for `MAP*` is an order of magnitude higher than `DeepMTL` and `DeepTxFinder`. Lastly, `SPLOT` [4] doesn't require any training.

[2]Source code at: https://github.com/caitaozhan/deeplearning-localization.

[3]To deal with areas of larger than 1 square kilometers, a tiling method similar to the one in [6] can be used. Tiling is not the focus of this work.

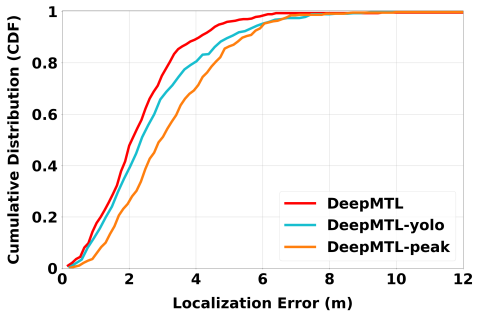

Fig. 6. Cumulative probability of localization error of `DeepMTL`, `DeepMTL-yolo` and `DeepMTL-peak`, for the special case of single transmitter localization with 6% sensor density.

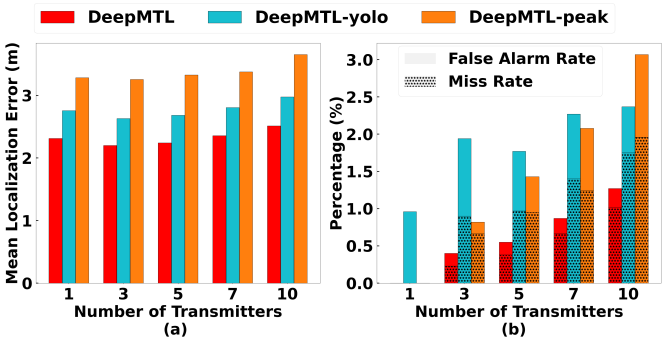

Fig. 7. (a) Localization error and (b) miss and false alarm rates, of `DeepMTL`, `DeepMTL-yolo` and `DeepMTL-peak` variants for varying number of transmitters in log-distance dataset/propagation model.

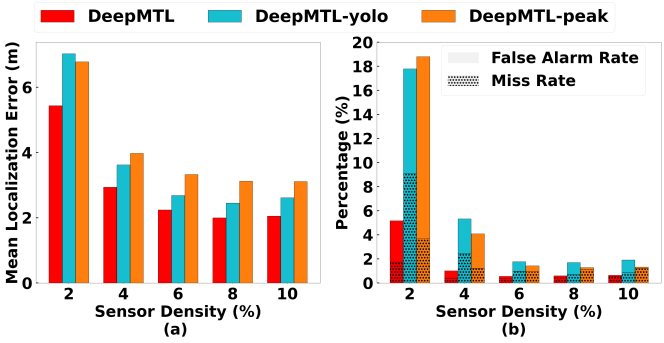

Fig. 8. (a) Localization error and (b) miss and false alarm rates, of `DeepMTL`, `DeepMTL-yolo` and `DeepMTL-peak` variants for varying sensor density in log-distance dataset/propagation model.

Two Propagation Models and Settings. The sensor readings (i.e. the dataset) are simulated based on a propagation model. To demonstrate the generality of our techniques, we consider two propagation models as described below.

Log-Distance Propagation Model and Setting. Log-Distance propagation model is a generic model that extends Friis Free space model which is used to predict the path loss for a wide range of environments. As per this model, the path loss (in

Table II. Compare Running Time (ms).

# Intru.	DeepMTL	DeepMTL-yolo	DeepMTL-peak
1	18.0	18.0	1.3
3	18.6	18.3	1.4
5	18.9	19.2	1.6
7	19.4	19.6	1.8
10	20.6	20.5	2.3

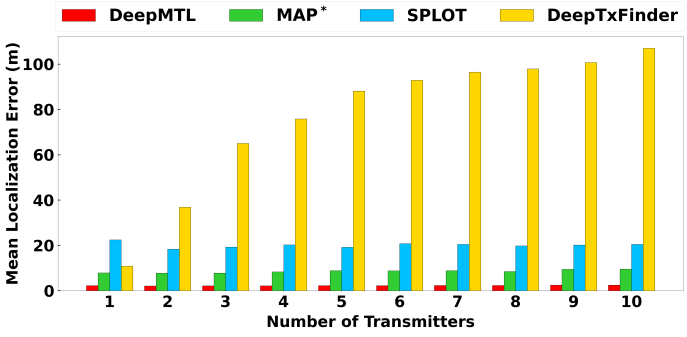

Fig. 9. Localization error of DeepMTL, MAP*, SPLOT, and DeepTxFinder for varying number of transmitters in the log-distance dataset.

dB) between two points x and y at a distance d is given by: $PL_d = 10\alpha \log d + \mathcal{X}$, where α (we use 3.5) is the path-loss exponent and $\mathcal{X}$ represents the shadowing effect of the propagation channel which can be represented by a zero-mean Gaussian distribution with a certain (we use 1) standard deviation. Power received (in dBm) at point y due to a transmitter at point x with a transmit power of P_x is thus: $P_x - PL_d$. Power received at point y due to multiple sources is assumed to be just an aggregate of the powers (in linear) received from each of the sources.

Log-Distance Propagation Model and Setting. This is a complex model of wireless propagation based on many parameters including locations, terrain data, obstructions, soil conditions, etc. We use SPLAT! [22] to generate path-loss values. SPLAT! is an open-source software implementing the Longley-Rice [23] Irregular Terrain With Obstruction Model (ITWOM) model. We consider an area of $1km \times 1km$ in our state and use the 600 MHz band to generate path losses.

A. DeepMTL vs. DeepMTL-yolo vs. DeepMTL-peak

In this subsection, we compare the three variants of our technique, viz., DeepMTL, DeepMTL-yolo, and DeepMTL-peak. For simplicity, we only show plots for the log-distance propagation model setting in this subsection (we observed similar performance trends for the Longley-Rice propagation model too).

Performance Results. In Fig. 6, we plot the cumulative density function (CDF) of the localization error, for the simple case of a single transmitter. We observe that DeepMTL outperforms the other variants, as it yields higher cumulative probability for lower range of errors. In addition, we evaluate the three variants for varying number of transmitters (Fig. 7) and sensor density (Fig. 8), and evaluate the localization error as well as the false and miss rates. We observe that DeepMTL

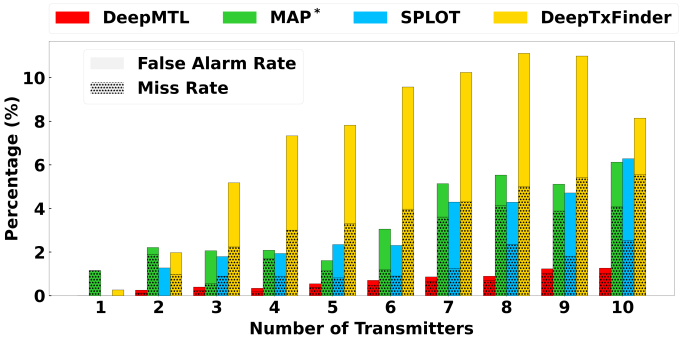

Fig. 10. Miss and false alarm rates of DeepMTL, MAP*, SPLOT, and DeepTxFinder for varying number of transmitters in the log-distance dataset.

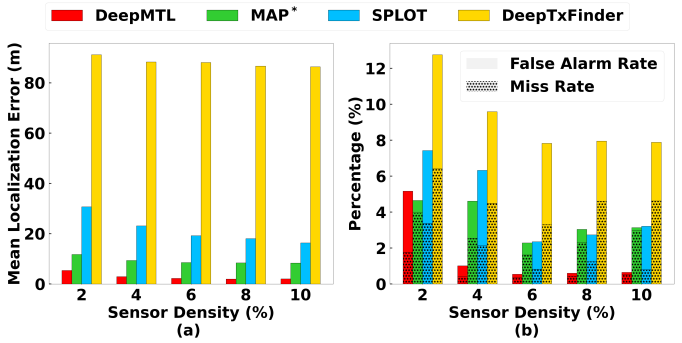

Fig. 11. (a) Localization error, and (b) miss and false alarm rates, of DeepMTL, MAP*, SPLOT, and DeepTxFinder for varying sensor densities in the log-distance dataset.

consistently outperforms the other two variants across all plots and performance metrics. As expected, the performance of all algorithms degrades with increase in number of transmitters (in terms of false alarms and miss rates) or with decrease in sensor density. In general, the localization error of DeepMTL is around 15-30% lower than the other variants. Impressively, the total cardinality error (i.e., false alarms plus miss rates) is only 1% or less for the DeepMTL technique, in most cases.

Running Time Comparison. For the running time comparison of the variants, see Table II. Our hardware is an Intel i7-8700 CPU and a Nvidia RTX 2070 GPU. We observe that, as expected, DeepMTL and DeepMTL-yolo which use a sophisticated object-detection platform do incur higher latency (around 20 milliseconds) than DeepMTL-peak (around 2 milliseconds). As our key performance criteria is accuracy and the run time of DeepMTL is still quite low, we choose DeepMTL for comparison with the prior works in §V-B.

Localizing Transmitters Close By. Localizing two or more transmitters close by is a hard part of the MTL problem. Fig. 5 gives an example when an advance object detection algorithm will work while a simple local maximal peak detection might not. Fig. 5 shows DeepMTL can successfully localize two transmitters as close as 3 pixels apart. When a pixel represents a $10m$ x $10m$ area, then it is 30 meters apart. If a pixel represents a smaller area, such as $1m$ x $1m$, it has the potential to localize two transmitters as close as 3 meters apart.

Table III. Compare Running Time (s)

# Intru.	DeepMTL	MAP*	SPLOT	DeepTxFinder
1	0.0180	8.78	1.53	0.0015
3	0.0186	15.1	1.79	0.0016
5	0.0189	19.3	2.06	0.0017
7	0.0194	24.1	2.32	0.0019
10	0.0206	28.5	2.72	0.0022

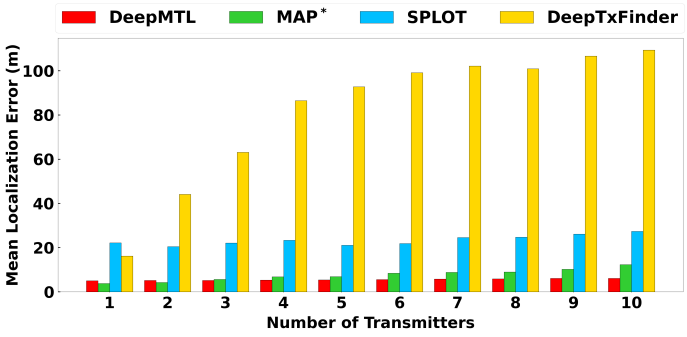

Fig. 12. Localization error of DeepMTL, MAP*, DeepTxFinder and SPLOT for varying number of transmitters in the SPLAT! Dataset.

B. DeepMTL vs. Prior Works

In this subsection, we compare DeepMTL with SPLOT, MAP*, DeepTxFinder in both log-distance (Fig. 9, 10, 11) and SPLAT (Fig. 12, 13, 14) propagation models and thus, datasets. We observe similar performance trends for both datasets, i.e., DeepMTL significantly outperforms the other approaches by a large margin (in many cases, by more than 50% in localization errors, false alarms and miss rates). For all techniques, as expected, the performance is generally worse in the SPLAT dataset compared to the log-distance dataset.

Varying Number of Transmitters. Fig. 9 and Fig. 12 show the localization error with varying number of transmitters, in the two datasets. We see that DeepMTL has a mean localization error of only 2 to 2.5 meters (roughly, 1/4-th of the side length of a pixel/grid cell) in the log-distance dataset and about 5 to 6 meters in the SPLAT dataset. In comparison, the localization errors of MAP*, SPLOT, DeepTxFinder are two to three times, eight to nine times, and few tens of times respectively more than that of DeepMTL. Fig. 10 and Fig. 13 show the miss and false alarm rates in the two datasets. We observe that DeepMTL's summation of miss and false alarm rate is only 1% even at 10 transmitters in the log-distance dataset, and about 4% for the case of SPLAT! dataset. In comparison, the summation of miss and false alarm rates for other schemes is at least 6% and 10% respectively for the two datasets, when there are 10 transmitters.

Varying Sensor Density. Fig. 11 and Fig. 14 plot the performance of various algorithms for varying sensor density in the two datasets. For very low sensor density of 2%, all algorithms perform badly (in comparison with higher sensor densities), but DeepMTL still performs the best. For higher sensor densities, we observe a similar performance trend as above—i.e., DeepMTL easily outperforms the other schemes by a large margin.

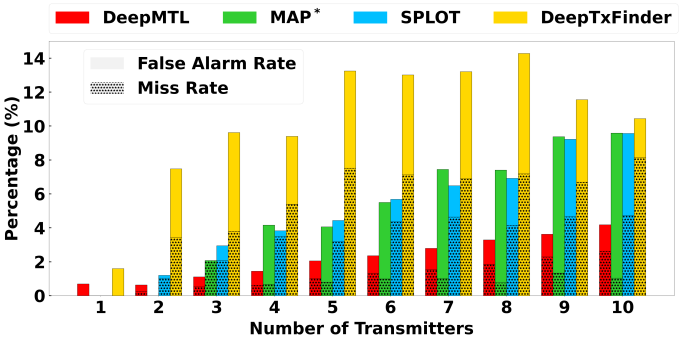

Fig. 13. Miss and false alarm rates of DeepMTL, MAP*, SPLOT, and DeepTxFinder for varying number of transmitters in the SPLAT! Dataset.

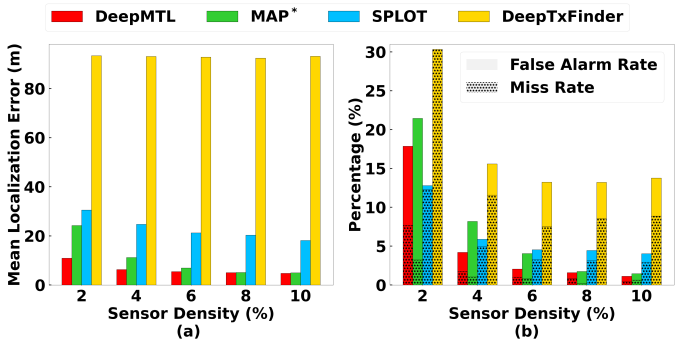

Fig. 14. (a) Localization error, and (b) miss and false alarm rates, of DeepMTL, MAP*, SPLOT, and DeepTxFinder for varying sensor densities in the SPLAT! Dataset.

Running Times. The run time of DeepMTL is orders of magnitude faster than MAP* and SPLOT (Table III), but an order of magnitude slower than DeepTxFinder (due to the deep YOLOv3-cust taking up over 90% of the run time).

Summary and Analysis. In summary, our approach significantly outperforms the other approaches in all the accuracy performance metrics, as well as in terms of latency. In particular, our approach also significantly outperforms the other CNN-based approach DeepTxFinder. The main reason for DeepTxFinder's inferior performance is its inability to accurately predict the number of TXs—which forms a fundamental component of their technique. In contrast, DeepMTL is able to circumvent explicit pre-prediction of number of transmitters by using a well-developed object-detection technique which works well for multiple objects especially in our context of simple objects.

VI. Related Work

Spectrum sensing is usually being realized by some distributed crowdsourced low-cost sensors. Electrosense [24] and its follow up work Skysense [25] are typical work of spectrum sensing. In this crowdsourced sensing paradigm [7], sensors collect I/Q samples (in-phase and quadrature components of raw signals) and compute PSD (power spectral density), which is RSSI. Crowdsourced low-cost sensors do not have the capability to collect TOA (time of arrival) or AOA (angle

of arrival) data because they require more expensive hardware. Spectrum sensing platforms serve as the foundation of spectrum applications, and transmitter localization is one of the main applications. Other applications include signal classification [26], spectrum anomaly detection [27], sensor selection [28], etc.

Transmitter localization. Localization of an intruder in a field using sensor observations has been widely studied, but most of the works have focused on localization of a single intruder [29], [30]. In general, to localize multiple intruders, the main challenge comes from the need to "separate" powers at the sensors [31], i.e., to divide the total received power into power received from individual intruders. Blind source separation is a very challenging problem; only very limited settings allow for known techniques using sophisticated receivers [27], [32]. We note that (indoor) localization of a device [33] based on signals received from multiple reference points (e.g, WiFi access points) is a quite different problem (see [34] for a recent survey), as the signals from reference points remain separate, and localization or tracking of multiple devices can be done independently. Recent works on multi-target localization/tracking such as [35] are different in the way that targets are passive, instead of active transmitters in the `MTL` problem. Among other related works, [36] addresses the challenge of handling time-skewed sensors observations in the MTL problem.

Deep learning for localization. Several recent works have harnessed the power of deep learning in the general topic of localization. E.g., DLoc in [12] designs a CNN that localizes a single target in indoor environment using WiFi CSI data, MonoDCell in [37] designs an LSTM that localizes a single target in indoor environment using cellular RSSI data, and `DeepTxFinder` in [6] uses CNN to address the same `MTL` problem using RSSI data in this paper.

VII. Conclusion and Future Work

In this paper, we have designed and developed a novel deep-learning (CNN) based scheme for the multiple transmitter localization problem. Our developed technique outperforms prior approaches by a significant margin in all performance metrics. For future work, we would like to extend our technique to estimate powers of the transmitters, and develop techniques to reduce training cost and optimize sensor deployment. We would also like to develop techniques which will allow the model to be used across geographical areas, perhaps, with some minimal training specific to an area.

VIII. Acknowledgements

This work is supported by NSF grants CNS-1642965 and CNS-1815306. The authors would like to thank the anonymous reviewers as well as Krishna Patel.

References

[1] J. G. Andrews *et al.*, "What will 5g be?" *IEEE Journal on Selected Areas in Communications*, 2014.

[2] "Electromagnetic spectrum superiority strategy," US Department of Defence, Tech. Rep., 2020.

[3] H. Kour, R. K. Jha, and S. Jain, "A comprehensive survey on spectrum sharing: Architecture, energy efficiency and security issues," *Journal of Network and Computer Applications*, 2018.

[4] M. Khaledi *et al.*, "Simultaneous power-based localization of transmitters for crowdsourced spectrum monitoring," in *MobiCom*. ACM, 2017.

[5] C. Zhan, H. Gupta, A. Bhattacharya *et al.*, "Efficient localization of multiple intruders for shared spectrum system," in *IPSN*, 2020.

[6] A. Zubow *et al.*, "Deeptxfinder: Multiple transmitter localization by deep learning in crowdsourced spectrum sensing," in *ICCCN*, 2020.

[7] A. Chakraborty *et al.*, "Specsense: Crowdsensing for efficient querying of spectrum occupancy," in *INFOCOM*, 2017.

[8] C. W. Kim *et al.*, "Design and implementation of an end-to-end architecture for 3.5 ghz shared spectrum," in *IEEE DySPAN*, 2015.

[9] L. Hartung and M. Milind, "Policy driven multi-band spectrum aggregation for ultra-broadband wireless networks," in *DySPAN*, Sep. 2015.

[10] J. K. Nelson, M. R. Gupta *et al.*, "A quasi em method for estimating multiple transmitter locations," *IEEE Signal Processing Letters*, 2009.

[11] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[12] R. Ayyalasomayajula, A. Arun *et al.*, "Deep learning based wireless localization for indoor navigation," in *MobiCom*, 2020.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *MICCAI 2015*, 2015.

[14] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," *CoRR*, 2017.

[15] Y. Wu and K. He, "Group normalization," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1803.08494

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, 2015.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[19] R. Girshick *et al.*, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[20] E. Linder-Norén, *Open source YOLOv3 implementation*, 2019. [Online]. Available: https://github.com/eriklindernoren/PyTorch-YOLOv3

[21] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick *et al.*, "Microsoft COCO: common objects in context," 2014.

[22] J. A. Magliacane, "Splat! a terrestrial rf path analysis application for linux/unix," Downloaded at https://www.qsl.net/kd2bd/splat.html, 2008.

[23] K. Chamberlin *et al.*, "An evaluation of longley-rice and gtd propagation models," *IEEE Transactions on Antennas and Propagation*, 1982.

[24] S. Rajendran, R. Calvo-Palomino, D. Giustiniano *et al.*, "Electrosense: Open and big spectrum data," *IEEE Communications Magazine*, 2018.

[25] B. Reynders, F. Minucci, E. Perenda *et al.*, "Skysense: Terrestrial and aerial spectrum use analysed using lightweight sensing technology with weather balloons," in *MobiSys*, 2020.

[26] S. Rajendran *et al.*, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE TOCCN*, 2018.

[27] Z. Li, Z. Xiao, B. Wang, B. Y. Zhao *et al.*, "Scaling deep learning models for spectrum anomaly detection," in *MobiHoc*. ACM, 2019.

[28] A. Bhattacharya, C. Zhan, H. Gupta, S. R. Das *et al.*, "Selection of sensors for efficient transmitter localization," in *IEEE INFOCOM*, 2020.

[29] A. Chakraborty *et al.*, "Spectrum patrolling with crowdsourced spectrum sensors," in *IEEE Infocom*, 2018.

[30] A. Dutta and M. Chiang, ""see something, say something" crowdsourced enforcement of spectrum policies," *IEEE TOWC*, 2016.

[31] N. Patwari *et al.*, "Locating the nodes: cooperative localization in wireless sensor networks," *IEEE Signal processing magazine*, 2005.

[32] M. Schmidt *et al.*, "Wireless interference identification with convolutional neural networks," in *INDIN*, 2017.

[33] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *IEEE INFOCOM*, 2000.

[34] F. Zafari, A. Gkelias *et al.*, "A survey of indoor localization systems and technologies," *IEEE Communications Surveys Tutorials*, 2019.

[35] C. Karanam *et al.*, "Tracking from one side – multi-person passive tracking with wifi magnitude measurements," in *ACM/IEEE IPSN*, 2019.

[36] M. Ghaderibaneh, M. Dasari, and H. Gupta, "Multiple transmitter localization under time-skewed observations," in *DySPAN*, 2019.

[37] H. Rizk and M. Youssef, "Monodcell: A ubiquitous and low-overhead deep learning-based indoor localization with limited cellular information," in *SIGSPATIAL*, 2019.