IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 February 2024, accepted 18 March 2024, date of publication 25 March 2024, date of current version 1 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3381798

RESEARCH ARTICLE

Exploratory Data Preparation and Model Training Process for Raspberry Pi-Based Object Detection Model Deployments

VIDYA KAMATH[1], RENUKA A.[1], (Member, IEEE), VISHWAS G. KINI[1], AND SHWETHA PRABHU[2]

[1]Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India
[2]Department of Electronics and Communication Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

Corresponding author: Renuka A. (renuka.prabhu@manipal.edu)

This work was supported by Manipal Academy of Higher Education, Manipal under Dr. T.M.A Pai Research Scholarship with Research Registration No: 200900143–2021.

ABSTRACT Most computer vision applications that use deep learning on constrained device come from the Internet of Things (IoT) or robotics fields, where low-quality cameras are used to capture input images in real time. Since most pretrained models typically undergo training on high-quality image datasets, the low-quality, noisy, or blurry images captured with these resource-constrained devices could possibly have a negative impact on the models' performance. To determine if model performance is impacted by training models using low-quality data, a secondary image dataset named MOD-2022 was prepared for object detection and tracking tasks using an exploratory data preparation methodology. This dataset was primarily designed to include a wider range of classes with an adequate number of images per class while being free of errors due to inaccurate labelling or annotations. Additionally, a training approach is also proposed to support the model's training when the dataset is considerably large. A VGG16-SSD model was trained with this approach on the prepared dataset and deployed on a Raspberry Pi and it showed that this approach is very useful in developing models for resource-constrained applications. Furthermore, this data preparation approach can be extended to prepare numerous other datasets required for training models designed to be deployed on constrained devices similar to the Raspberry Pi.

INDEX TERMS Dataset preparation, deep learning, model training, object detection, Raspberry Pi, resource-constrained.

I. INTRODUCTION

Object detection is a well-known problem in computer vision in the present day, and the trend of extending object detection tasks onto constrained devices is a huge development sparked by the advancement of the IoT industry [1]. One of the most popular devices in this line of work is the Raspberry Pi single-board computer. As it is known, deep learning models extract features and learn from the data used to train them. Deploying a deep learning model on constrained devices presents a number of challenges, including limited battery life, high energy consumption, reduced processing capacity, and limited memory while maintaining a certain degree of accuracy [1], [2]. In this research work, an effort is made to consider the issue from a model design perspective. To further optimize the model's performance, we closely monitor the training method by carefully designing the dataset that the model learns on.

When using constrained devices, the image quality acquired differs greatly from those in the datasets being used to train the existing models. This necessitates the creation of a dataset that best captures the context of real-time images on resource-constrained devices, where image quality, camera resolution, and various deformities

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

present in the images, such as noise, blur, capture delays, and jitter, greatly compromise them. Given that the model performance is also influenced by the image quality [3], [4], [5], the dataset as such can play a major role and might help the model perform better.

There are several publicly available image datasets for training models for object detection tasks [6], [7], [8], [9], [10], [11]. However, these datasets are huge and take more time to parse, and may demand more epochs and batches to learn from the data during model training. It takes 14 days to complete 90-epoch ImageNet-1k training with ResNet-50 on an NVIDIA M40 GPU [12]. This time can be reduced by using multiple CPUs and GPUs. Nevertheless, it is debatable whether parsing an excessive number of classes is indeed beneficial. The massive computing requirement is also leading researchers from other disciplines to prefer existing models over inventing new model architectures, for their use on constrained devices [1].

Furthermore, it's possible that the high-quality images used in these datasets do not precisely represent the low-quality images captured in the constrained scenarios. For example, capturing two images, one with a 720p camera and another with a 1080p camera and resizing them to 224 × 224 pixels would not result in identical results.

- Image Quality: A higher resolution does not always imply a higher quality image. Image quality is influenced by a variety of elements, including sensor size, lens quality, and lighting conditions. Image quality may differ between two cameras with the same resolution.
- Post-Processing: The majority of digital cameras use post-processing to enhance the images they capture. This processing can involve noise reduction, sharpening, colour correction, and other changes. The post-processing used by a 720p camera may differ from that applied by a 1080p camera.
- Resolution: When compared to the 720p camera, the 1080p camera captures more pixels in the same frame. This means that the 1080p picture contains greater detail.
- Resizing: is the process of changing the number of pixels in an image. When the size is reduced (down-sampled), pixel information is removed; when the size is increased (up-sampled), pixel information is added. In both cases, an algorithm determines which pixels to remove and how to add pixels. This procedure has the potential to degrade image quality.
- Distortions: The distortion types may include JPEG, JP2K, MP4, and H.264. Most Raspberry Pi cameras, for example, capture real-time video in H.264, which differs from video captured on other cameras that make use of MP4.

So, even if the images are the same size (224 × 224 pixels) after resizing, they may still be different due to the factors stated above [13], [14], [15]. As a result, while operating on a constrained device, the images included in the publicly available dataset might not accurately reflect a real-time scenario, thus the model trained on them fails to provide acceptable accuracy.

Additional preprocessing is necessary to make these images resemble a constrained device's real-world scenario. Most model development requires some preprocessing before training the model on them; nevertheless, the processing mainly involves resizing the images, which may lower image quality but may still not represent the low quality of images captured with constrained devices. Because most existing models perform train-time and test-time augmentations during training and do not save the generated intermediate results, it is challenging to acquire appropriate datasets for training models and much more difficult to figure out what preprocessing to perform on them when constructing models for constrained applications.

Image datasets prove to be a crucial part of the development of deep learning models. Even though deep learning models have recently seen a considerable shift towards deployment in resource-constrained applications, most efforts seek to achieve their objective by deploying pretrained models trained on publicly available datasets. Having a smaller error-free dataset to train the model may always seem advantageous when more training time cannot be invested. This will also motivate more researchers to come up with new model architectures that will enhance research. Consequently, a strategy is adopted in the present work in which publicly available datasets are utilized to generate a new secondary dataset for wide applications of object detection on resource-constrained devices such as the Raspberry Pi. This dataset will be identified as MOD-2022. The purpose of this exploration and preparation of the dataset is to answer the question: Does the image quality affect the model's performance on a constrained device?

To address this, we propose a training approach that can be used to train models on large datasets, and we use it to train and deploy the VGG16-SSD model [16], [17] on a Raspberry Pi. Major contributions from this work to the research community are as listed below:

- The study proposes the notion that having noise in the dataset could turn out to be beneficial when the target application is expected to perform on noisy data.
- The explored dataset MOD-2022 can be useful in boosting the performance of deep learning based models for real-time object detection and tracking applications.
- The MOD-2022 dataset, as prepared and processed, might be useful input when developing and training new model architectures for resource-constrained use, which requires that the images employed match the lower quality of images on which the intended application is deployed.
- This research serves as an outline for researchers who wish to build secondary datasets particular to their applications.
- The incremental training procedure provided for training a model on the dataset may also be beneficial for training

on other large datasets when processing capacity is limited.
- Because the technique used in this research constructs data from scratch while public datasets mainly serve as image sources, the data exploration process can possibly be repeated to generate many other datasets including those that are primary data sources.

Section II discusses related works, whereas Section III provides an overview of the methodology used to build this dataset. Sections IV through VIII elaborates on each stage of the process in depth. This might be valuable to other researchers interested in developing custom applications using a Raspberry Pi or a similar device. For training with limited GPU resources, a unique training approach is also provided. Section IX discusses the outcomes achieved and answers the research question. Section X lists the work's limitations and scope. Section XI provides the conclusion and Section XII provides insights into the future directions.

II. RELATED WORKS

The various standard data preparation techniques, train-time augmentations, and test-time augmentations used on a few well-known models are explored in this section. A variety of object detection datasets that are available in the literature are also reviewed and compared to the MOD-2022 dataset.

One of the popular models that first used the ImageNet [6] dataset was AlexNet [19]. The data preparation in this consisted of resizing the images to a shape of 256×256 pixels, as the dataset consisted of images of differing sizes. Square images were directly resized, whereas rectangular images were first resized to 256 pixels on the shortest side and then cropped at the center. A mean pixel value was subtracted from each pixel from each channel as part of normalization. Other popular models to perform on this dataset are GoogLeNet [20], VGGNet [17] and ResNet [21] also used similar steps of selecting a size, and then resizing all the images to the selected size and centering the pixel values by subtracting the mean pixel value. The only difference was the size chosen to resize, i.e. 224×224 instead of 256×256, as this is the usual input size expected by most of the models.

The train-time augmentations done in all these models were performed in memory and hence are called just-in-time augmentation, which is a widely used approach today. The reason being the huge size of the dataset, which makes storing the results of augmentation a challenging task.

The most commonly used train time augmentations are the horizontal flips of smaller cropped images that are resized to the required shape, random changes to the light level or brightness of the images, random crops with a fixed set of aspect ratios, and photo-metric distortions that involve changing the color and contrasts. The VGGNet used a slightly different approach where the model would first be trained with fixed small-sized images and then the weights would be retained and used as a starting point for training the model again with larger fixed-size images. This proved to be a beneficial way to reuse the dataset.

The test-time augmentations on the other hand required a rigorous approach to make the model best fit the dataset and it involved various techniques. The AlexNet model used five cropped versions of the input image and five cropped versions of the horizontally flipped version of the image and calculated the average of the predictions. GoogLeNet involved multi-scale test-time augmentations, where each image was re-sampled at four different scales, and multiple square crops were taken and resized to the expected shape giving 144 versions of a given input image. Again, the predictions here were averaged. VGGNet and ResNet models used a mixed version of the above approach and used an average of the predictions as well.

In contrast to the ImageNet dataset [6], [18], the MSCOCO dataset [7] tries to increase the number of images per class while limiting the number of classes to a smaller number. In this case, the amount of data is still huge, but the model is able to find more instances of a certain object. It is analogous to making a child memorize 10 answers 100 times as opposed to 100 answers 10 times. It would seem ideal to have additional classes given how broadly applicable the model could be. To put it another way, even if the child could just partially answer 100 questions, it would be highly appreciated, compared to perfectly answering only 10 questions.

There are certain situations in which simply partial solutions may not be sufficient, in which case there is a need to continue training further. When a model is intended to be used only to predict a certain kind of data class after its deployment, it might not be beneficial to continuously over-train it on too many additional classes and attempt to generalize it on them. Further, these datasets typically contain crowd sourced annotations, and annotation errors might have negative effects on the model that was trained using them [22]. All of this influences the choice to build an own dataset, ensure it is error-free, and use it to train the model.

The comparison of the MOD-2022 dataset with other popular datasets is listed in Table 1. It is possible to train a detector with equivalent performance from scratch without using pre-training. Although ImageNet pre-training speeds up convergence, it does not have much effect on the performance of final detection [23]. The same holds true for COCO pretrained models.

In an effort to produce a new dataset, the advantages of both the ImageNet and MSCOCO data preparation techniques have been combined. The purpose of this type of dataset creation is to identify possible classes that may be used in object-tracking applications. Further, during this process, the dataset is processed and prepared to match the image quality on the target application in order to facilitate the model's deployment on constrained devices. It will also be ensured that the dataset is made accessible in both processed and raw formats, with the processed forms of the images having nearly identical features compared to a raw image captured on a resource-constrained device.

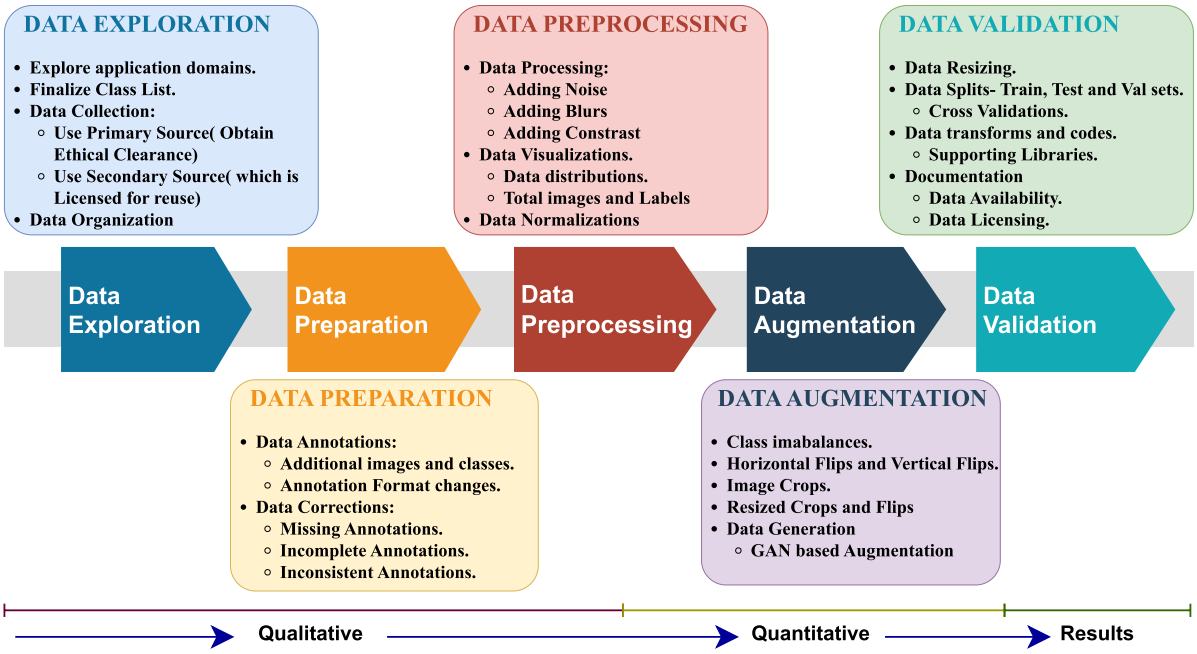

FIGURE 1. Data exploration and preparation pipeline: 1. Data Exploration, 2. Data Preparation, 3. Data Preprocessing, 4. Data Augmentation, 5. Data Validation.

TABLE 1. Summary of related datasets on object detection.

Dataset	Classes	Images/ Size	Description	MOD-2022
use ImageNet [6] [18]	1000	14M/ 121GB	Annotation Format: PascalVOC, one XML file per each image. Several missing annotations are present.	+ Consists of a total of 180 classes. The size of the raw data is 6.4GB, and the size of the augmented dataset is 16.9 GB.
MS-COCO [7]	90	328K/ 42.7GB	Annotation Format: MS-COCO, two JSON object files for train and test sets. Manual annotations plus possible computerized methods.	+ 42K images with manually annotated bounding boxes in PascalVoc format.
Pascal-VOC [8]	20	11K/ 8.4GB	Annotation Format: PascalVOC, one XML file per image. Complete manual annotation.	+ Takes approximately 13 minutes per epoch to parse the entire dataset in a single batch on Nvidia RTX 3050 on VGG16-SSD Model [16], [17].
CalTech-101 [9]	101	9K/ 1.28MB	Annotation Format: Matlab script. One annotation file per image.	- Created utilizing existing public datasets. It was ensured that only image datasets having a free use and redistribution license were utilized in the process.
CalTech-256 [10]	256	30K/ 1.07GB	Annotation Format: Matlab script. One annotation file per image.	
AWA2 [11]	50	37K/ 1.07GB	Annotation Format: Pre-extracted feature representations, 85 numeric attribute values per class. No bounding box annotations are available.	- Targets only applications to be deployed on resource-constrained devices, and may not be suitable for other applications.

III. METHODOLOGY

Exploratory research is a method used to examine questions that haven't been fully researched before. The fact that deep learning models have been trained on high-quality data has led to several studies focusing on improving image quality in order to boost model accuracy [24]. It is unclear, nevertheless, if a model can be trained with images of poor quality [4]. Additionally, using pre-trained models or huge training datasets may not really guarantee increased model performance [23]; this can only be determined by developing a model from scratch on a moderate dataset of low-quality images. This is especially relevant if the intended applications for the models need to operate on constrained devices and generate poor-quality data for the model during deployment. Therefore, this work aims to build one such dataset, to test the hypothesis that developing a dataset of low quality can enhance model performance during deployment on constrained devices. The major objective of this work is to be able to answer the research question:

RQ: Can deep learning models for object detection and tracking, meant to be deployed on constrained devices, be improved by using a dataset that contains a substantial amount of low-quality images?

In light of this, a technique was employed that involves first conducting a qualitative analysis of dataset classes, focusing on potential applications, setting it up to meet the model's requirements, eliminating any potential human errors, and preparing it to balance and match poor quality that prevail in constrained settings. Secondly, the dataset is improved and statistically evaluated to make sure it meets the model standards. A stage of quantitative analysis comes next. Since a qualitative investigation was done before moving on to a quantitative analysis, the methodology followed by this work is identified as exploratory research [25]. Figure 1 illustrates the entire process. The subsequent Sections IV through VIII provide a detailed explanation of each stage in the process.

The first step is data exploration. This entails investigating several application domains to finalize the class list, followed by data collection from primary or secondary sources. This step includes adequate data organization, labelling, and nomenclature. Section IV describes how we executed this step in our work. The next step is to prepare the data to make it suitable to be used for model training. In the object detection task, the data must be prepared by labelling the classes in the images using bounding box annotations. Section V details this procedure. The next step is data preprocessing, which includes visualizing and modifying the data to suit the target distributions, as well as data normalization. Please refer to Section VI for a complete explanation.

The next step, known as data augmentation, involves increasing the dataset size to help with model training by generating new data from the existing dataset. In particular, we use crops to expand our dataset. Finding the right balance between classes and calculating weights for each class are also part of this stage, that will help minimize the impact of dataset imbalances during training. The process is described in Section VII. The combined output of steps three and four makes up the processed MOD-2022 dataset.

An object detection model is trained on the data in the final step, called data validation, and it is implemented on the target application. Determining the dataset splits, computing the mean and standard deviation to normalize the data during training, recording the model deployment outcomes, and documenting the meta data and the results are all part of this process. The process is explained in detail in Section VIII.

IV. DATA EXPLORATION

A total of 180 object classes were finalized by targeting major object tracking applications, and images in JPEG format were randomly sampled across various publicly available datasets, the details of which are shown in Figure 2, and the finalized class list is shown in Figure 3. Applications involving face recognition were not considered due to concerns of privacy and confidentiality, and applications useful in the military were also not addressed for fear of unfair use. In the entire process, we ensured that only images that were licensed for free use and redistribution were utilized.

Various publicly available image datasets were explored for the images required in each class. The data classes were assigned numbers and, around 100 to 600 images per class were chosen and renamed corresponding to the object classes they were a part of. This was done to prevent an excessive skew in the distribution of images per class. Note that this is only the number of images; the number of annotations for each class will vary since there may be several instances of one object present in a single image and also because some images may contain objects other than the ones for which they were chosen. Annotating and re-annotating the images as well as verifying the consistency of the annotations are also some major stages in the data preparation process.

V. DATA PREPARATION

This section summarises the benefits of preparing a relevant dataset along with the major issues addressed through this process. In general, the annotation process can be done in three ways: Manual, Automated and Semi-Automated. Automated annotation generates annotations automatically by using pre-trained models. Annotation quality could vary, but it is faster than manual annotation. The semi-automated approach is where a certain level of automation is used along with manual annotations. The majority of publicly available datasets use manual or semi-automated annotations [26]. These datasets are annotated by a group of researchers, contributors, and crowd-sourced labor utilizing a certain level of automation. This choice majorly depends on the size of the dataset and the availability of time and human resources.

Because the major purpose of this study was to build a balanced dataset that is reasonably large enough to train a deep learning model but not so huge that it burdens the system for training the model, creating the dataset error-free was crucial, and therefore the MOD-2022 was manually annotated. This involved annotating, re-annotating, and reviewing the bounding box annotations of the finalized images from the previous stage. This was found to be the most time-consuming step and labor-intensive since the annotations done required corrections along with a second round of verification to make sure it was as error-free as possible.

Manual annotations, on the other hand, might be time-consuming yet lead to high-quality annotations. The process was focused on avoiding the numerous errors [22] that might emerge from unorganized anonymous annotations and hence, this dataset was not made available for public annotations, unlike other crowd-sourced datasets. The entire process of preparing the MOD-2022 dataset took about eight months and 250 man-hours.

The primary datasets were utilized solely for the images and the entire annotation process was redesigned to achieve the error-free standards outlined for MOD-2022. Model performance decreases with an increase in annotations that

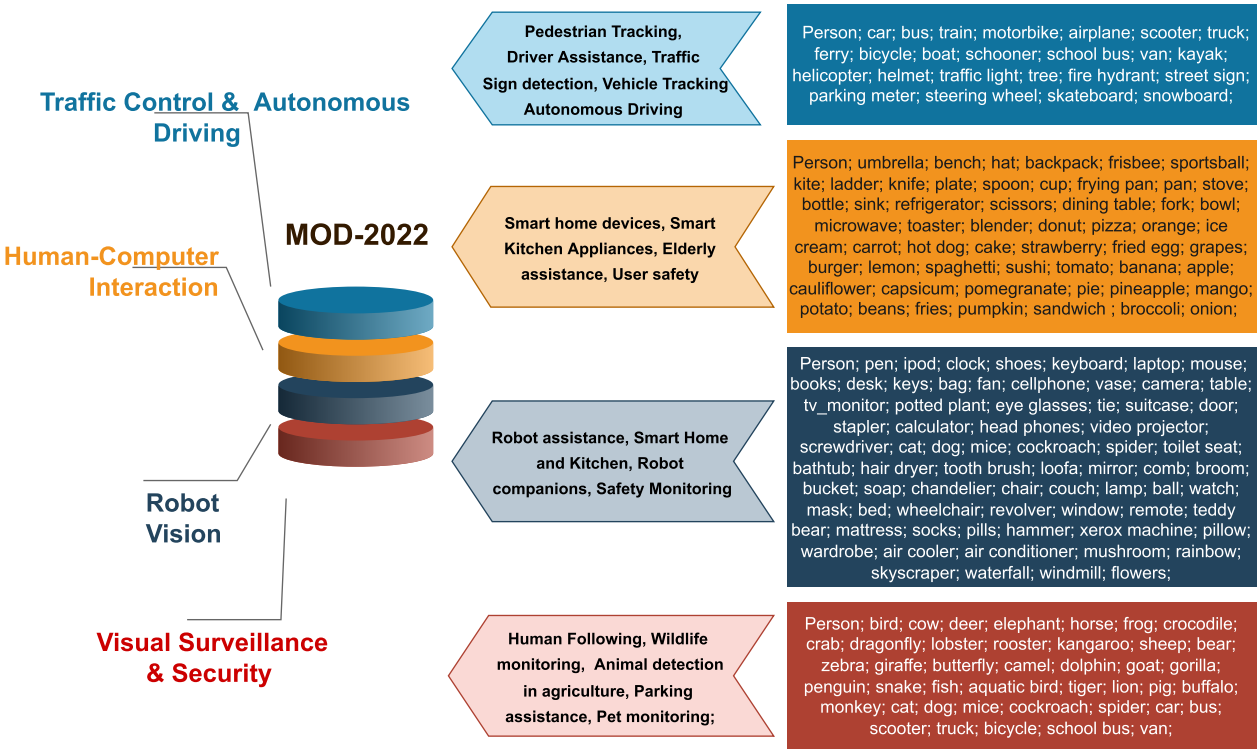

FIGURE 2. Major object tracking applications explored to finalize the classes in the dataset, MOD-2022.

Code	Class name	Code	Class name	Code	Class name	Code	Class name	Code	Class name	Code	Class name
MOD001	air conditioner	MOD031	camel	MOD061	fire hydrant	MOD091	lamp	MOD121	pomegranate	MOD151	stove
MOD002	air cooler	MOD032	camera	MOD062	fish	MOD092	laptop	MOD122	potato	MOD152	strawberry
MOD003	airplane	MOD033	capsicum	MOD063	flowers	MOD093	lemon	MOD123	potted plant	MOD153	street sign
MOD004	apple	MOD034	car	MOD064	fork	MOD094	lion	MOD124	pumpkin	MOD154	suitcase
MOD005	aquatic bird	MOD035	carrot	MOD065	fried egg	MOD095	lobster	MOD125	rainbow	MOD155	sushi
MOD006	backpack	MOD036	cat	MOD066	fries	MOD096	loofa	MOD126	refrigerator	MOD156	table
MOD007	bag	MOD037	cauliflower	MOD067	frisbee	MOD097	mango	MOD127	remote	MOD157	teddy bear
MOD008	ball	MOD038	cellphone	MOD068	frog	MOD098	mask	MOD128	revolver	MOD158	tie
MOD009	banana	MOD039	chair	MOD069	frying pan	MOD099	mattress	MOD129	rooster	MOD159	tiger
MOD010	bathtub	MOD040	chandelier	MOD070	giraffe	MOD100	mice	MOD130	sandwich	MOD160	toaster
MOD011	beans	MOD041	clock	MOD071	goat	MOD101	microwave	MOD131	school bus	MOD161	toilet seat
MOD012	bear	MOD042	cockroach	MOD072	gorilla	MOD102	mirror	MOD132	schooner	MOD162	tomato
MOD013	bed	MOD043	comb	MOD073	grapes	MOD103	monkey	MOD133	scissors	MOD163	tooth brush
MOD014	bench	MOD044	couch	MOD074	hair dryer	MOD104	motorbike	MOD134	scooter	MOD164	traffic light
MOD015	bicycle	MOD045	cow	MOD075	hammer	MOD105	mouse	MOD135	screwdriver	MOD165	train
MOD016	bird	MOD046	crab	MOD076	hat	MOD106	mushroom	MOD136	sheep	MOD166	tree
MOD017	blender	MOD047	crocodile	MOD077	head phones	MOD107	onion	MOD137	shoes	MOD167	truck
MOD018	boat	MOD048	cup	MOD078	helicopter	MOD108	orange	MOD138	sink	MOD168	tv_monitor
MOD019	books	MOD049	deer	MOD079	helmet	MOD109	pan	MOD139	skateboard	MOD169	umbrella
MOD020	bottle	MOD050	desk	MOD080	horse	MOD110	parking meter	MOD140	skyscraper	MOD170	van
MOD021	bowl	MOD051	dining table	MOD081	hot dog	MOD111	pen	MOD141	snake	MOD171	vase
MOD022	broccoli	MOD052	dog	MOD082	icecream	MOD112	penguin	MOD142	snowboard	MOD172	video projector
MOD023	broom	MOD053	dolphin	MOD083	ipod	MOD113	Person	MOD143	soap	MOD173	wardrobe
MOD024	bucket	MOD054	donut	MOD084	kangaroo	MOD114	pie	MOD144	socks	MOD174	watch
MOD025	buffalo	MOD055	door	MOD085	kayak	MOD115	pig	MOD145	spaghetti	MOD175	waterfall
MOD026	burger	MOD056	dragonfly	MOD086	keyboard	MOD116	pillow	MOD146	spider	MOD176	wheelchair
MOD027	bus	MOD057	elephant	MOD087	keys	MOD117	pills	MOD147	spoon	MOD177	windmill
MOD028	butterfly	MOD058	eye glasses	MOD088	kite	MOD118	pineapple	MOD148	sportsball	MOD178	window
MOD029	cake	MOD059	fan	MOD089	knife	MOD119	pizza	MOD149	stapler	MOD179	xerox machine
MOD030	calculator	MOD060	ferry	MOD090	ladder	MOD120	plate	MOD150	steering wheel	MOD180	zebra

FIGURE 3. List of finalized classes in the dataset, MOD-2022. Total of 180 classes starting from MOD001 up to MOD180.

cause bounding box shifts [22]. Moreover, adding new classes or re-annotating already-existing annotations to conform to the annotation format is often necessary when preparing a new dataset. In addition, we found additional types

of annotation errors. These include missing, inconsistent, and incomplete annotations which are explained in the subsections that follow. In Figure 4, these annotation errors are illustrated with examples.

A. MISSING ANNOTATIONS

The term ''missing annotation'' refers to an object in an image that is not annotated when it should be. This is troublesome causing the model to be trained on false negatives of the objects. As seen in Figure 4 (top-right), some images have annotations for one class of object while completely ignoring other classes that are present in the same image. Few other partially annotated images were also present were the annotations were incomplete, as seen in Figure 4 (left). To ensure that there were no missing or incomplete annotations, every image in the prepared MOD-2022 dataset was carefully examined and annotated.

B. INCONSISTENT ANNOTATIONS

The object is defined in a given image by its boundary using the bounding box. Because a deep learning based model learns the object features through the defined boundaries, annotating an object differently across the dataset may affect how the model learns the features of the object. For example, as shown in Figure 4 (bottom-right), the extent to which the object 'bus' is annotated in two different images has significant difference. This causes the model to learn with inconsistent boundaries of a particular object, and thus influence the model's accuracy during the test phase. As a result, the bounding boxes were kept consistent during MOD-2022 image annotations facilitating easier learning for the model.

C. ADDITIONAL CLASSES IN THE DATASET

Another major reason for re-annotating images was to annotate objects that weren't previously annotated due to the absence of a particular object class in the dataset they were taken from. For example, MOD-2022 included object classes like 'window' and 'tree' which the widely used datasets did not have. Being able to recognize these would be beneficial in a variety of tracking applications hence these classes were chosen in MOD-2022. The class lists for the annotations in each dataset also varied when the images from several datasets were combined, thus each image had to have its annotations checked. There was a need to make sure that each of the 180 classes of objects contained in any image obtained from any dataset was appropriately labeled and no class was overlooked.

Occasionally the bounding box labels had to be changed to reflect the class labels that have been chosen for the dataset preparation because they varied across different datasets. For instance, the ImageNet labels were a numbered sequence whereas in MOD-2022, relevant and descriptive labels were preferred to represent the object, making the annotations more meaningful and complete.

D. CHANGE IN ANNOTATION FORMAT

Most datasets publicly available along with annotations used various annotation formats, such as '.mat', '.txt', '.xml', and '.json' files, depending on the annotation technique used. A single annotation format, PascalVoc, was chosen in MOD-2022 to achieve consistency and hence, there was a need to convert or re-annotate images with other annotation formats to this chosen format.

The PascalVoc annotation format was chosen for annotations for two reasons. First, it is the format used by ImageNet. Second, the PascalVoc class list was retained in MOD-2022 to compare benchmarks, so having these annotations would save the need to change formats for validation in the future when compared to using the COCO annotation format. In contrast to COCO, which uses a JSON file, Pascal VOC uses an XML file. Additionally, unlike COCO, which uses a single JSON file for all images, separate XML files had to be created for each image that needs an annotation in MOD-2022. The bounding box co-ordinates in the Pascal VOC format are as given below:

$$(xmin - topLeft, ymin - topLeft, xmax - bottomRight, ymax - bottomRight) \quad (1)$$

where, *(xmin, ymin)* are co-ordinates of top left corner of the image and *(xmax, ymax)* are the coordinates of the bottom right corner of the image, and *topLeft* and *bottomRight* refer to co-ordinates of top left corner and bottom right corner of the bounding box that marks the object present in the image.

VI. DATA PREPROCESSING

This included the choice of an input size for the training set which further required careful visualization of the data. Depending on the selected input size, the images would be either up-scaled or down-scaled. These actions could have negative consequences. When a model is down-scaled, the number of pixels containing vital features gets significantly reduced, making it difficult for the model to detect objects. When images are up-scaled, they are typically padded with zeroes, and these pixels have no impact on detection. Hence, the following actions were chosen to be carried out to obtain the best results.

A. DATA VISUALIZATION STEP- FOR PIXEL SCALING AND NORMALIZATION

As it is known that the dataset will determine the ideal size to employ in the model training, it is necessary to first visualize the image sizes. By extracting the images using Python libraries and visualizing their resolutions on a plot, one can easily identify the distribution of the image sizes. The Kaggle images come in a wide range of sizes, with the majority of them averaging around 500 pixels. The images from Caltech-101 are largely centered in the 350×450 pixel range, in contrast to the images from AWA2, which have images with resolutions as high as 1000 pixels. After the completion of the annotation process, a complete plot of all

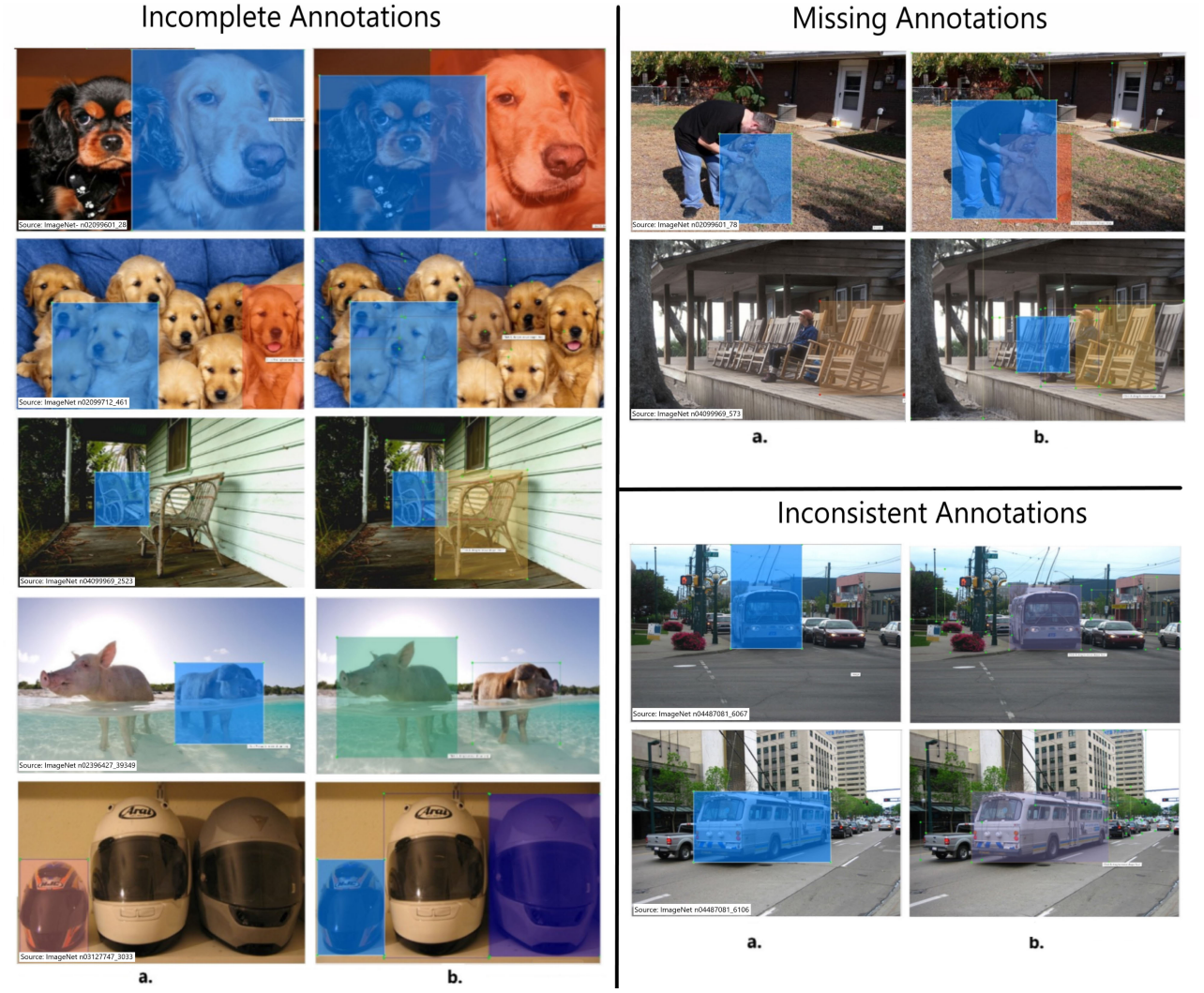

FIGURE 4. Types of annotation errors. 1. Incomplete Annotations(left) 2. Missing Annotations(top-right) 3. Inconsistent Annotations(bottom-right). Note: a. Before re-annotation, b. After re-annotation. (samples shown from the primary dataset: ImageNet [18]. Refer: bottom-left corner of each sample to find the image filename from the source).

the images in the dataset was generated, which would be helpful in choosing the sizes for the model input. However, to avoid any drawbacks from choosing the inappropriate input size, data visualizations were carried out which could lessen the impact. By doing so, the significant effects on the model from resizing images would be eliminated. Figure 5 shows a sample image on which resizing was applied along with horizontal flip.

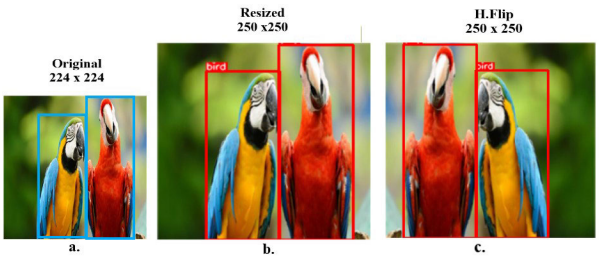

FIGURE 5. Sample- resizing and flipping. (Source: original image from FSS_1000 [27] macaw-2).

B. NOISE, CONTRAST, AND BLUR

The dataset being constructed had a total of 100 to 600 images per class, which could still be considered a small dataset for neural network training. There was a chance on the model memorizing this dataset, which was the first issue. The model could possibly learn the precise input samples and their related outputs rather than learning a general mapping from inputs to outputs. This would lead to the model performing admirably on the training dataset but poorly on new data, which is undesirable. The second issue was that a small dataset would make it harder to describe the structure of the input space and relation to the output. However, adding more data in this case would not make it possible to resolve these problems [28]. Adding random noise is one way to improve generalization error and the structure of the mapping

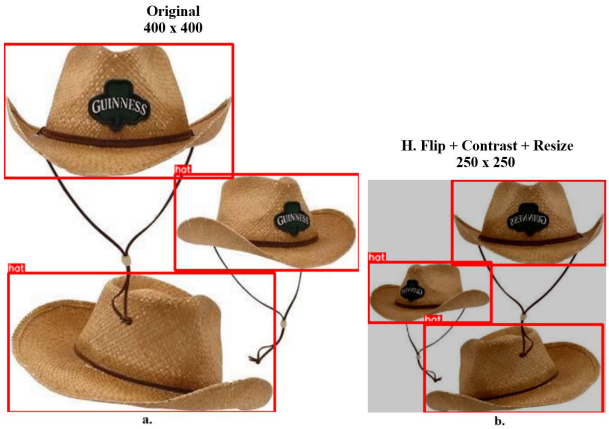

FIGURE 6. Illustration of a sample- flipped, resized, and added contrast. (Source: original image from Caltech-256 [10] cowboy-hat 051_0044).

problem and it can occasionally be seen as a stage in data augmentation [29], [30], [31].

Edges are more clear when contrast preprocessing is applied as the nearby pixel differences gets enhanced. This would make it easier for the deep learning model to learn the features. The model might learn new features from an image by adding contrast to it, as seen in Figure 6. This helps to generalize the model better. Similarly, noise and blur can also be added to an image to create pixel-level irregularities that do not significantly change the image but effects on the model during training. Making the model aware of these anomalies during training can make it robust towards them during deployment as this is more prominent in real-time applications. Figure 7 illustrates the various blurring effects that were employed, and Figure 8 illustrates the various noise effects that were added to an image during the preprocessing of MOD-2022. The libraries that were used to implement the aforementioned image effects provided a wide range of options, such as the ability to establish blur limits, probabilities of applying effects, and the choice of randomly applying these effects.

VII. DATA AUGMENTATIONS

Although the aforementioned actions might be used in association with data augmentations, their effects only prevail when applied over an entire image. A more beneficial and efficient tool to increase the number of images would be to use image cropping. Trimming the objects might improve the learning performance of a model. A minimum crop area for visibility was defined to prevent bounding boxes from being applied to objects below a level. This ensured that the objects in the cropped image retained enough features to identify them as objects. Figure 9 shows a sample image on which the center crop has been used.

A. HANDLING CLASS IMBALANCE

Data augmentations are usually used for handling class imbalances. In a classification problem, the imbalance can be handled by controlling the number of images per class. However, in a detection problem, the amount of labels per class can be handled by using oversampling along with weighted classes.

1) OVERSAMPLING

To balance the overall number of images per class, the classes with the fewest labels are over-sampled. This is an easy solution to the classification problem. Using the MOD-2022 Dataset for training, oversampling of classes with less than 200 images was done to improve the class probabilities. Additional processed images were created for each class during data augmentation process using horizontal flips and crops in order to ensure the model could find adequate samples per each class while training.

However, when considering detection problem this could result in model over-fitting failing to address the imbalance in the class. Therefore, oversampling was used in MOD-2022 to handle classes with lower sample sizes. However, the number of labels per class was balanced during training using weighted classes.

2) WEIGHTED CLASSES

The weighted class technique attempts to give each class a weight that is inversely proportionate to the number of labels observed in it. Class weights affect the loss function directly as they penalize classes with more instances and vice versa. In an unbalanced dataset, intentionally biasing the model to favor more accurate predictions of the higher-weighted or the minority class results in sacrificing some capability to predict the lower-weighted or the majority class.

The weights of each class in MOD-2022 were found using following formula and stored as a tensor.

$$w_j = \frac{n_j}{n\_samples} \tag{2}$$

where, *n_samples* is the number of samples present in the class containing the highest number of labels and n_j is the number of samples present in class j. This tensor was then passed to the loss function during the loss calculation.

VIII. DATA VALIDATION

The dataset was examined for consistency before utilizing it for training, and various tests were performed to ensure the dataset was clean and error-free. First, the entire image dataset was type-checked to ensure that it was in '.jpg' format. The annotations were then type-checked to ensure that they were in '.xml' format and that there were no corrupted image or annotation files.

Following that a consistency check was done by a thorough data verification process using the LabelImg tool, to verify that the annotations in the images were consistent and flawless. Next, code checks were done using the *Dataset* and *DataLoader* libraries from PyTorch to parse and prepare the MOD-2022 dataset for model training and testing. To further assure complete data validation several steps including dataset splits, data normalization, class imbalance correction,

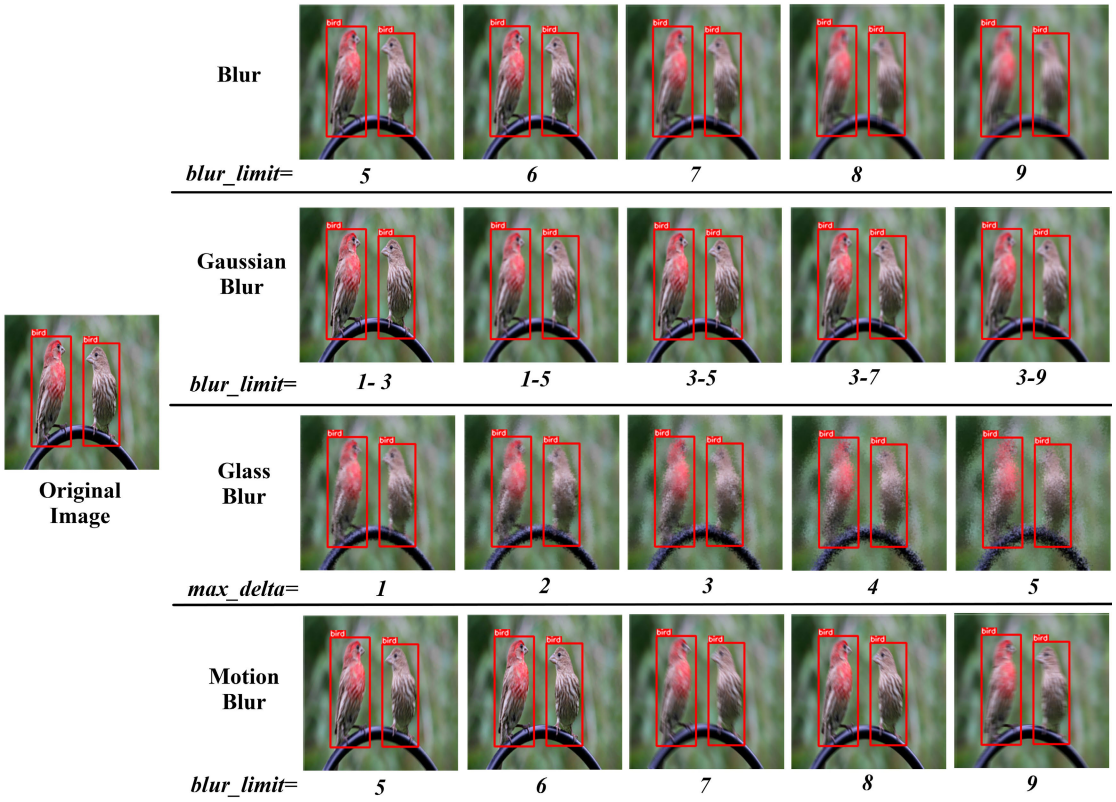

FIGURE 7. Illustration of blur types used during data preprocessing. (Source: original image from FSS_1000 [27] housefinch- 4).

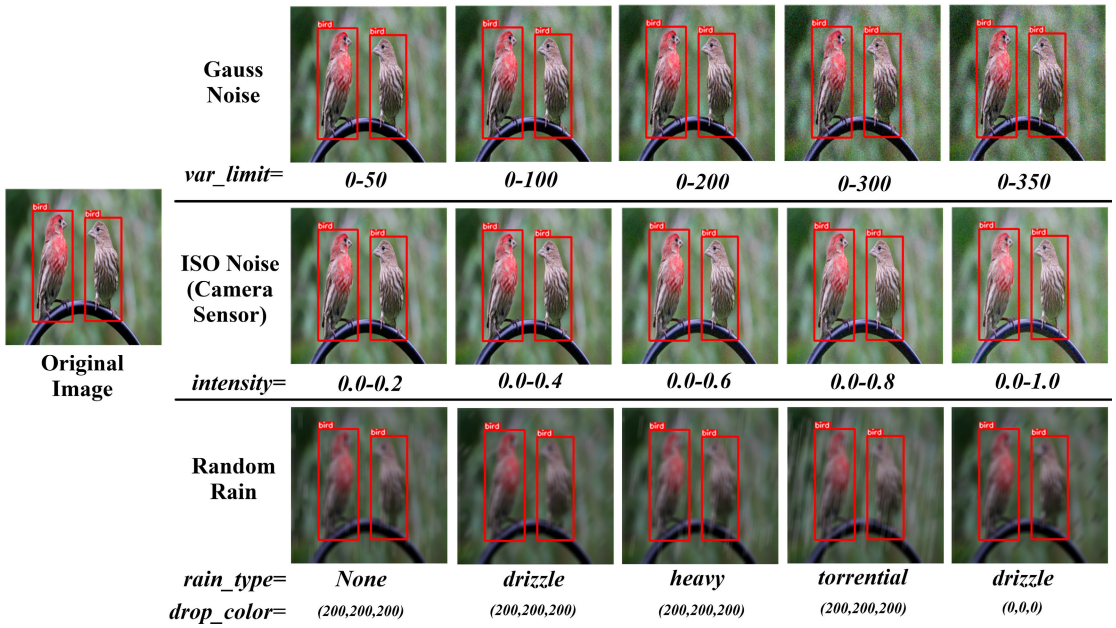

FIGURE 8. Illustrations of noise types used during data preprocessing. (Source: original image from FSS_1000 [27] housefinch- 4).

data documentation, and deployments of model trained using the dataset on a Raspberry Pi were carried out.

A. DATASET SPLITS

During the data exploration, the dataset was partitioned into a train set and test set with an 8:2 ratio in each class. This was done so that proportionate samples of the data explored from the various public datasets were available in the train set and test set. Since the trained model is validated using real-time data from the Raspberry Pi, we did not use a separate validation set. The produced dataset contained *33,435* images with annotations in the train set and *8,483* images with

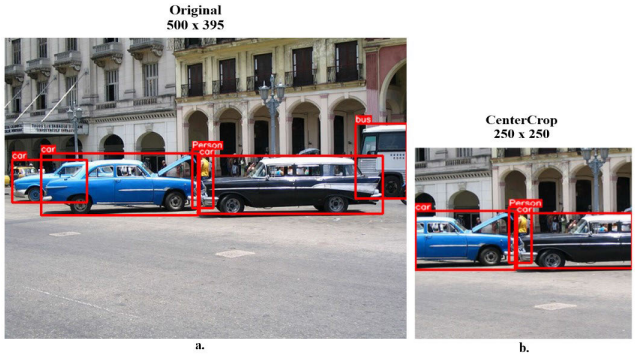

FIGURE 9. Illustration of a sample- center crop with resize. (Source: original image from ImageNet [18] n02814533_57788.).

annotations in the test set, with a total size of around 6GB with *42,911* annotated images. This is used as the set1: *MOD*2022*TR*1 and *MOD*2022*TS*1, while training the model (Refer: Algorithm 1 from Section VIII-D2).

1) TRAIN-TIME AND TEST-TIME AUGMENTATIONS

Data preprocessing had been carried out on all *42,911* images, and each image was augmented thrice, resulting in a processed dataset with *1,00,305* train augments and *25,473* test augments, leading to a total size around 17GB with *1,25,778* annotated images. The per-class train and test weights were calculated and saved using the number of instances per class. This is useful for dealing with class imbalance and data scarcity in a dataset without resulting in data repeats while employing the oversampling or weighted class approach. This is used as the set2: *MOD*2022*TR*2 and *MOD*2022*TS*2, while training the model (Refer: Algorithm 1 from Section VIII-D2).

B. DATA NORMALIZATION

Data normalization is the process of modifying the data to make all features measure on the same scale, which usually is 0 to 1. This is especially useful for datasets with characteristics scaled at significantly different levels, such as secondary datasets like MOD-2022, which incorporate images from multiple datasets. To normalize the dataset during the training and testing phases, the mean and standard deviation of all train and test subsets of the prepared and processed dataset were calculated. The mean and standard deviations are shown in Table 2.

C. DOCUMENTATION

The entire dataset and its creation process require documentation, and the raw, prepared, and processed versions of MOD-2022 were all documented and archived. The information on the entire process is additionally recorded in an Excel sheet to ensure desirable data availability and process transparency. A detailed data note of the entire process is given in Table 3 and Figure 10 shows a sample visualization of the prepared dataset, MOD-2022.

TABLE 2. Mean and standard deviation values per channel of each subset of MOD-2022.

	Mean[RGB]	STD[RGB]
Prepared: Trainset	[0.132, 0.124, 0.112]	[0.060, 0.060, 0.060]
Prepared: Testset	[0.132, 0.125, 0.113]	[0.060, 0.060, 0.060]
Processed: Trainset	[0.128, 0.123, 0.115]	[0.058, 0.058, 0.058]
Processed: Testset	[0.128, 0.123, 0.115]	[0.058, 0.058, 0.058]
Overall: MOD-2022	[0.129, 0.123, 0.114]	[0.058, 0.058, 0.058]

FIGURE 10. Illustration of a few samples from MOD-2022 as viewed through the data visualisations.

D. DEPLOYMENTS

The absence of ground truth values during real-time deployments is one of the primary reasons why model performance issues go unnoticed in the deployed application. The models often perform well on the train and test sets and may exhibit excellent accuracy. However, in real-time performance, the visualization of the model's predictions must be examined and analyzed to determine if the model is indeed performing well. This cannot be quantified using an evaluation metric such as accuracy.

The model's speed in frames per second and the number of correct or incorrect predictions it renders throughout a set of real-time trials can only be used as a basis for assessing the model. This dataset was prepared and processed to enhance this sort of model performance. If the model can detect objects better than before it is trained on this data set, then the objective of preparing the dataset is achieved.

1) EXPERIMENTAL SETUP

Figure 11 shows a Raspberry Pi 4 Model B with 8GB RAM that was utilized in this work, along with a Raspberry Pi-Camera Module that was used to provide real-time input to the model during the deployment.

An object detector, SSD [16] which is a popular model from the PyTorch package, along with pretrained VGG16

TABLE 3. A data-note on the MOD-2022 dataset.

Subject	Computer Science, Artificial Intelligence, Computer Vision and Pattern Recognition.
Type of data	Images with bounding box annotation files in PascalVoc format.
How the data were acquired	Images were explored and collected from primary/secondary datasets freely available for use for research. Annotations were done for the finalized 180 classes on all the images using the LabelImg tool in Pascal Voc annotation format.
Data format	1. Filtered and Annotated, 180 folders. Each folder contains .jpg files of train images and .xml files for annotations that correspond to them. A subdirectory called "test" is also included in each folder, and it includes the test images and annotations for that class. 2. Prepared train and test sets. Two folders, a train set folder, and a test set folder, that are ready to be placed into the preprocessing pipeline for model training. Both folders include images in the .jpg and corresponding annotation files in .xml. Prepared Version- Total images: 42,911, Train set: 33,435, Test set: 8,483. Processed Version- Total images: 1,25,778, Train set: 1,00,305, Test set: 25,473.
Description of data collection	A total of 180 object classes were finalized and images in JPEG format were randomly sampled across various publicly available datasets. A number of publicly available image datasets were explored to find the images needed for each class. The data classes were assigned numbers and, around 100 and 600 images per class were chosen and named corresponding to the object class they were a part of. This was done to prevent an excessive skew in the distribution of images per class.
Data Sources	This dataset was created by exploring other primary/secondary sources: ImageNet [18], MSCOCO [7], AWA2 [11], Caltech-101 [9], Caltech-256 [10], PascalVoc [8], FSS-1000 [27], Kaggle [32]–[42] RoboFlow [43] Care was taken to ensure that only CC BY/CC 0 licensed and reusable data were used during the exploration.
Data and Code accessibility	The MOD-2022 dataset, its metadata, the codes used to handle the data, and the codes used to process and use the data, will be made available upon reasonable request to the Corresponding author's email id.
Value of data	Useful in offline training of models that are developed to be deployed in real-time object detection and tracking applications on devices with constrained resources.

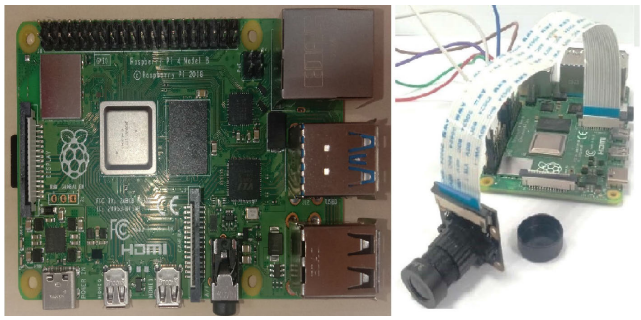

FIGURE 11. Raspberry Pi 4 model B with Pi camera module.

[17] as BaseNet, was chosen to be deployed and evaluated to determine if the dataset preparation purpose was attained. The pretrained VGG16 model on the ImageNet dataset was downloaded and used for weight initialization. The model was re-trained with MOD-2022, and several versions of the model were deployed on Raspberry Pi to check if the model's detection improved.

The model was trained on an Asus TUF Gaming Laptop with 16GB RAM, an AMD Ryzen 7 CPU, and a 4GB NVIDIA RTX 3050 GPU. Spyder development environment from the Anaconda Distribution was used to develop and train the model.

2) MODEL TRAINING ON MOD-2022

To record the train loss and validation loss, the model was trained using PyTorch (torch.nn) libraries on the train sets and validated on the test sets using the multi-box loss function as described in [16]. The main parameters that were modified were the Batch_Size (BS) and Learning_Rate (LR) until the model fit the data. The Stochastic Gradient Descent (SGD) optimizer, a weight decay of 5e-5 with a momentum of 0.9 was utilized during the training. The final BS and LR chosen were 8 and 1e-6 respectively. Depending on the underlying GPU characteristics and dataset used, this choice may vary. If the same model is to be trained using the dataset again, the values for BS and LR must be identified again for the system used to train the model. The decay value used was *0.1*, and significant tweaking was done to discover the optimal epochs at which the decay could be applied on the LR.

However, due to the limited storage of the GPU available in comparison to the >20GB dataset, the model would get trapped in local minima. Considering most users have limited GPU for training, a new incremental training approach was adopted to train the model. To achieve this, the decay was applied to the alpha value in the loss function when applied on the LR. This was done to balance the pace of convergence without enticing the model in local minima.

In addition, to address the issue of model skew on a certain class of object due to constrained BS, the entire dataset was partitioned into two sets. The first set featured a smaller file size and more images in their higher-resolution form, whereas the second set included image augments in their lower-quality form. The ratio of low-quality to high-quality images was set to 3:1 to ensure that low-quality images predominated in the dataset. The Figure 14 depicts dataset splitting and how data is sent in parts during the model training. The model is trained with gradients and back propagation using the train sets to obtain the train loss; the test sets without gradients are used to obtain the validation loss.

Algorithm 1 Incremental Training Methodology on the MOD-2022 Dataset Using Step Decay

Choose the Model *Model* ← *Model_name*
Load the Dataset *MOD*2022 Prepared and Processed, Train and Test sets for training the Models as *MOD*2022*TR*1, *MOD*2022*TS*1, *MOD*2022*TR*2 and *MOD*2022*TS*2 respectively.
Set number of Epochs to Train:*numEpoch*
Initialize an array *EpochsDecay* consisting of Epochs to Swap the Dataset.
Configure the Hyper-parameters used in the Model Training.
Set a decay value to apply at *EpochsDecay*, say *Decay* = 0.05
repeat
    if *Epoch* in *EpochsDecay* **then**
        TrainSet = *MOD*2022*TR*2
        TestSet = *MOD*2022*TS*2
        Apply *Decay* on the Learning rate.
        else:
        TrainSet = *MOD*2022*TR*1
        TestSet = *MOD*2022*TS*1
    end if
    repeatBegin training with torch.grad:
        Train the *Model* on the *TrainSet*.
        Record the Metrics *MultiBoxLossM*1.
    until End of Dataset
    repeatBegin validation with torch.nograd:
        Validate the *Model* on the *TestSet*.
        Record the Metrics *MultiBoxLossM*2.
    until End of Dataset
    Evaluate the Loss Function
    Save *Model*.state as *Model.pth*
    Plot the Loss Curves.
until $M1 < M2$ OR *numEpoch* < *Epoch*

The training approach that is proposed for developing deep models such as the VGG16-SSD [16], [17] on our Dataset: MOD-2022 using a modest GPU with 4GB RAM is described in Algorithm 1. Figure 12 and Figure 13 depict the difference between the loss during the model development when using decay compared to the one without it. To deploy the model, an interface in the 'Class' defined for the model was required which would accept the input image and return the predicted boxes, labels, and scores. This was also incorporated into the model during its development to make it deployable on a Raspberry Pi.

It is worth noting that the suggested training approach is only effective provided the train and test sets are rigorously constructed, as done in the MOD-2022, and may not behave similar on all datasets. Depending on the problem, it's also important to carefully select the ratio of processed images to the prepared ones. We have maintained the 3:1 ratio respectively since our target applications are for constrained

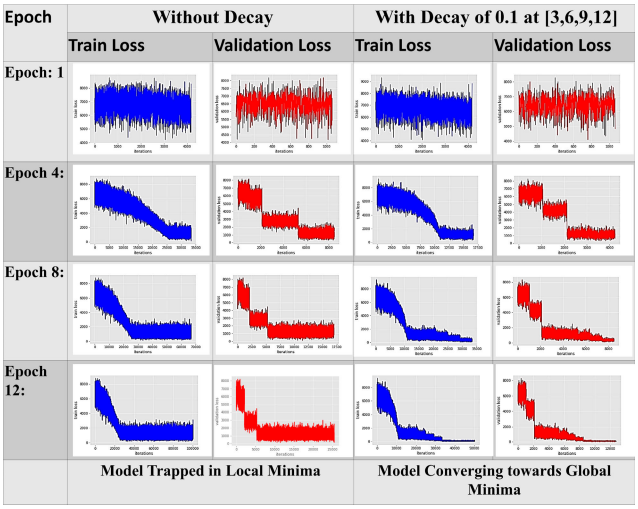

FIGURE 12. Loss comparison during model training using the proposed incremental training with step decay vs one without.

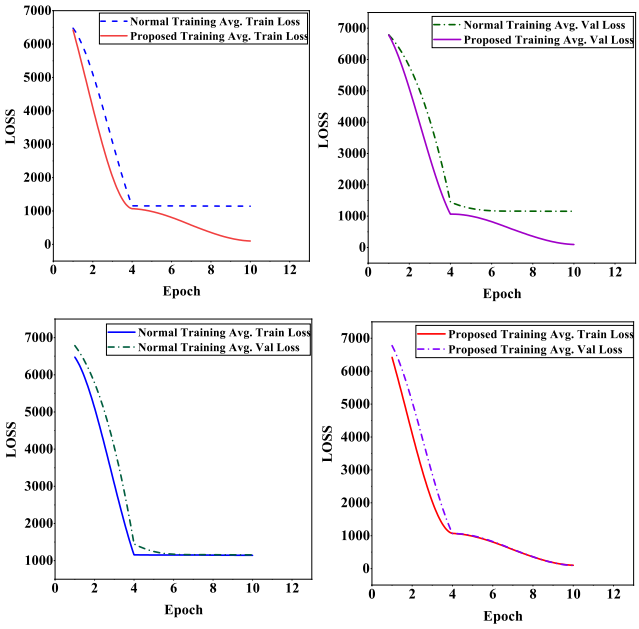

FIGURE 13. Average loss comparisons while using the proposed incremental training with step decay vs one without.

device deployments, and the approach works adequately in this case.

IX. DISCUSSION

The proposed method addresses the research question by using a VGG16-SSD [16], [17] model and training the SSD layers from scratch. The BaseNet utilized, VGG, cannot be trained from scratch as it is an extremely deep network; however, the task can be accomplished while using lightweight feature extractors. The proposed method of employing the MOD-2022 dataset can perform better when utilized with various lightweight BaseNet designs, either pretrained or from scratch, to enhance the obtained results. This is especially useful for developing novel BaseNet architectures when pretrained models are not readily

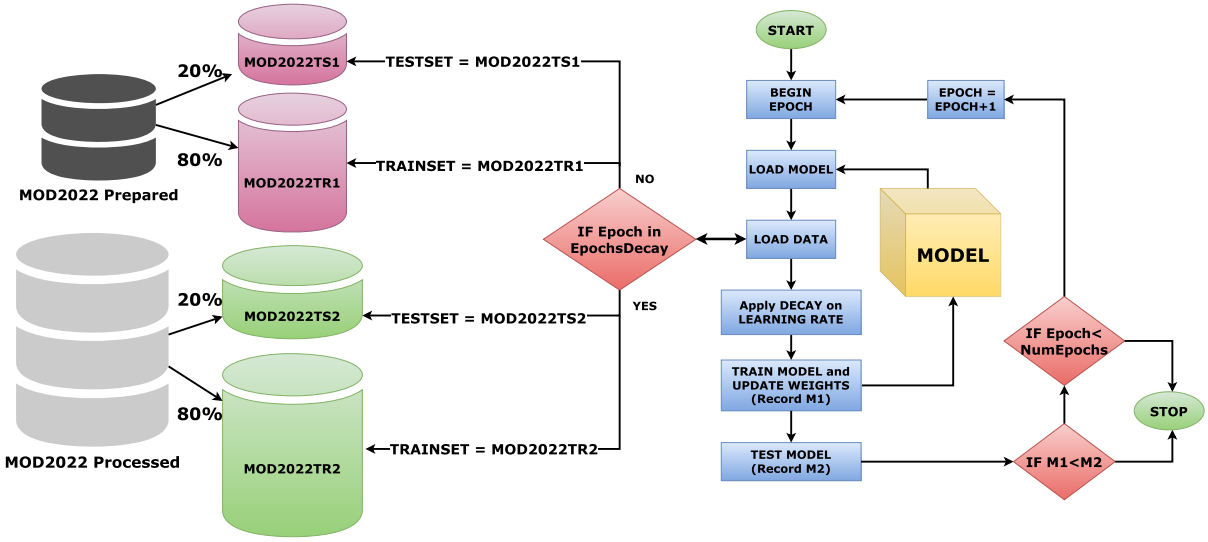

FIGURE 14. Flow diagram depicting the dataset splits and model training strategy using the proposed training approach.

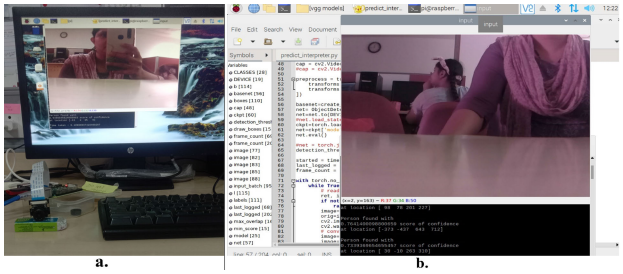

FIGURE 15. a. Snapshot taken during the model testing on Raspberry Pi. b. Sample screenshot of the Raspberry Pi monitor during the model testing.

available. This is where our dataset will be quite helpful, since it can offer a reasonably sized and potentially diversified set of classes to train a model from scratch. This work largely describes the dataset and its preparation process, and includes one small implementation that could address the research question and illustrate the model's usability based on the qualitative outputs. This dataset will eventually be used for developing equivalent models in the future, where pretrained weights are not available.

The model trained on our MOD-2022 dataset is deployed on a constrained device for real-time model validation, which uses live video streaming to validate the model. The model variants were stored in the *'model.pth'* format and exported from the Laptop to the Raspberry Pi device along with the model, utility functions, and configuration files required for deployment. Figure 15 shows a screenshot taken during model deployment and testing on the Raspberry Pi.

The Raspberry Pi 4 was used for deploying the model variants for real-time detection on the Pi Camera input feed. Figure 16 compares the variants of the model trained for varying numbers of iterations. It is evident that the model performs better in terms of its ability to identify objects much more accurately after training for 100000 iterations and more. With regard to this, utilizing the processed images from MOD-2022 can help improve the model's performance accuracy in real time. Regardless of the amount of training iterations, a particular model's real-time inference speed remains constant. This dataset is not intended to increase the speed of the model; rather, it is meant to boost prediction accuracy. A few sample outputs from the model during real-time testing are shown in Figure 17.

To evaluate the model's outcome qualitatively with other pretrained models on high-quality image datasets, we conducted experiments to run the Imagenet pretrained models [44] and the trained VGG16-SSD model on MOD-2022 on Raspberry Pi in real-time on a similar scene. Figures 18 and 19 show how our model performs in a cluttered scene with multiple objects compared to pretrained models. Figure 20 depicts the performance of the models when objects are placed individually. Some models that work well with single objects also under-perform when more objects are included. Our model operates effectively even in crowded scenes.

It is evident that our model detects objects more accurately than the pretrained models. Despite its slow detection speed, our model correctly recognizes correct objects. It is also worth noting that the pretrained model architectures were designed to be lightweight when compared to the VGG16-SSD model employed in this study. This suggests that lightweight model architectures, such as MobileNets [45], EfficientNets [46], and Inception_v1 [20], could possibly benefit from training on datasets generated using the proposed method. These results therefore are effective in answering the research question.

A. INFERENCE DRAWN

A concern with the model's performance is that the distributions of the training dataset and the input given

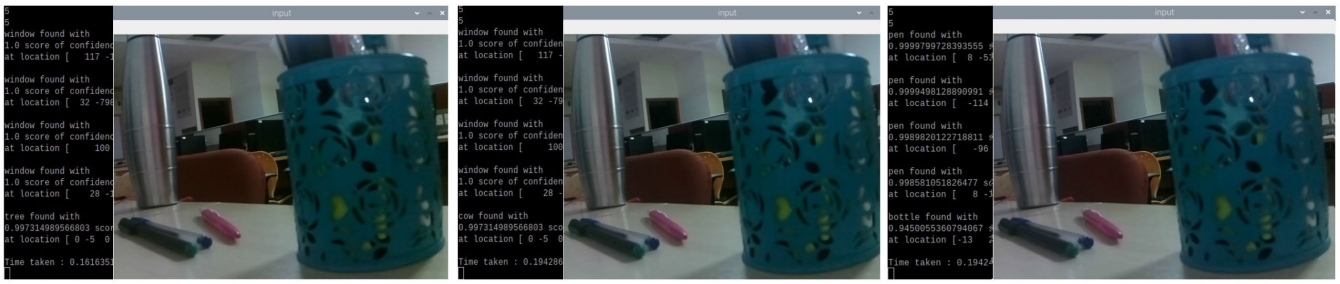

FIGURE 16. Comparison of output from model variants trained for varying numbers of iterations using the proposed training methodology, on similar real-time feed from a Raspberry Pi. *(Detections, Left-to-right: 1. Incorrect: window, tree. 2. Incorrect: window, cow. 3. Correct: pen, bottle.)*

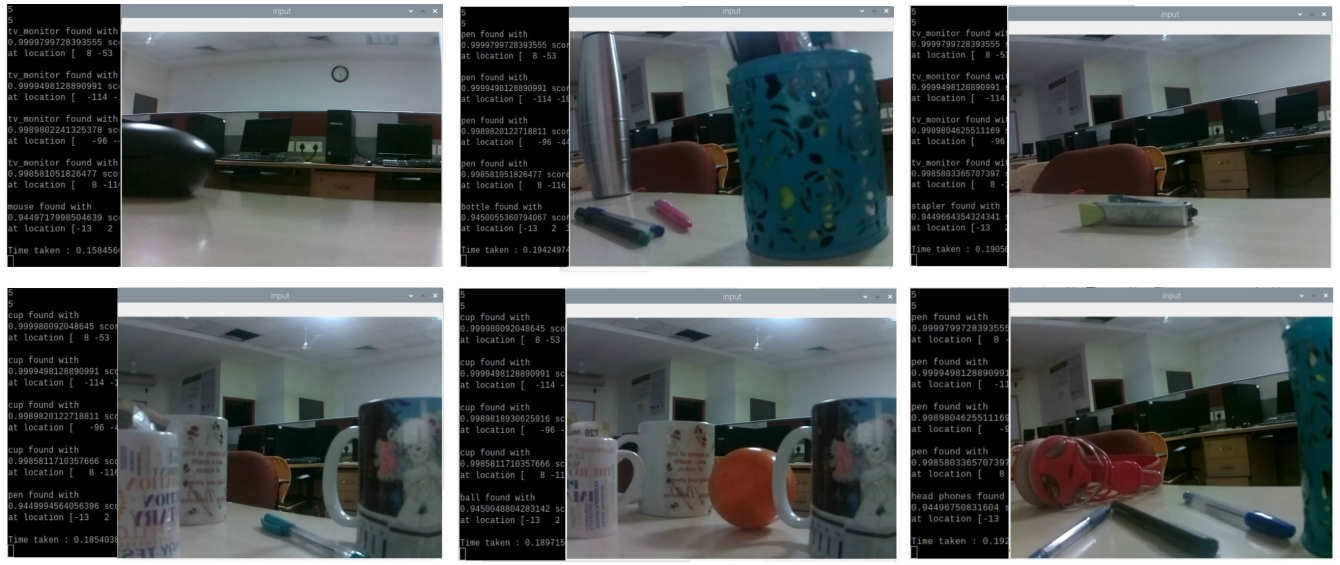

FIGURE 17. Sample outputs for real-time predictions from the trained VGG16-SSD model on Raspberry Pi 4. *(Detections, Left-to-right, Top-to-bottom: 1. tv_monitor, mouse. 2. pen, bottle. 3. tv_monitor, stapler. 4. cup, pen. 5. cup, ball. 6. pen, head phones.)*

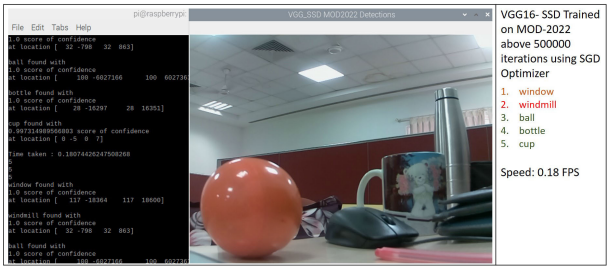

FIGURE 18. Output from the VGG16-SSD pretrained on MOD_2022 on same scene. (Detections, window, windmill, ball, bottle, cup.)

to it during deployments could not always match [4]. A model trained on low-quality data may learn better during deployment than a model built on high-quality images because it can distinguish across image quality and concentrate on objects regardless of irregularities that come up during real-time deployment on constrained devices.

As a result, the MOD-2022 dataset was explored, prepared, and processed with the aim of using it to train an object detection model for utilization on a Raspberry Pi. The trial results show that, while the model's performance is not optimal, it qualitatively performs better compared to pretrained models trained on large data sets containing high-quality images. Increased detection accuracy comes at the cost of speed. However, this study illustrates that training the model on low-quality images can potentially improve the model's accuracy and correctness, hence answering the research question.

X. LIMITATIONS

The work tries to explore images from many publicly available datasets and does not address the challenges experienced during the data-gathering process. On the other hand, the data preparation approach employed in this research could potentially be used on any primary dataset that is acquired.

This study does not include quantitative comparisons of all the performance attributes that must be addressed when

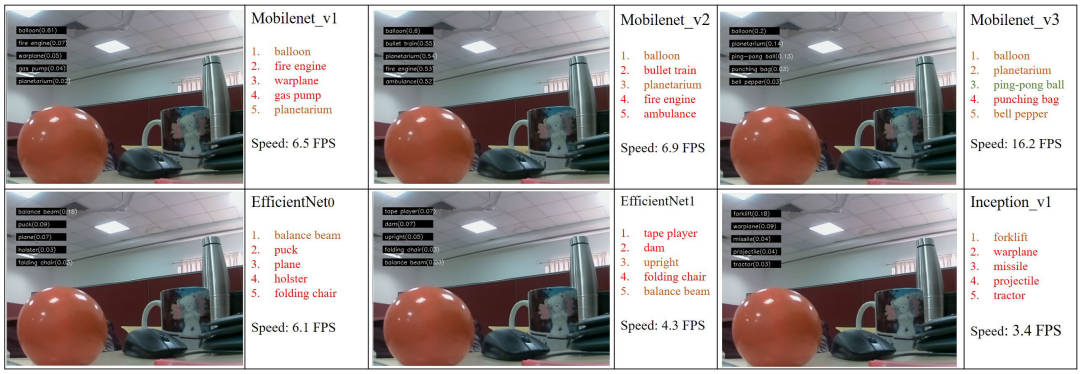

FIGURE 19. Outputs from pretrained models that were trained on high-quality image datasets. (Top-5 detections on same scene.)

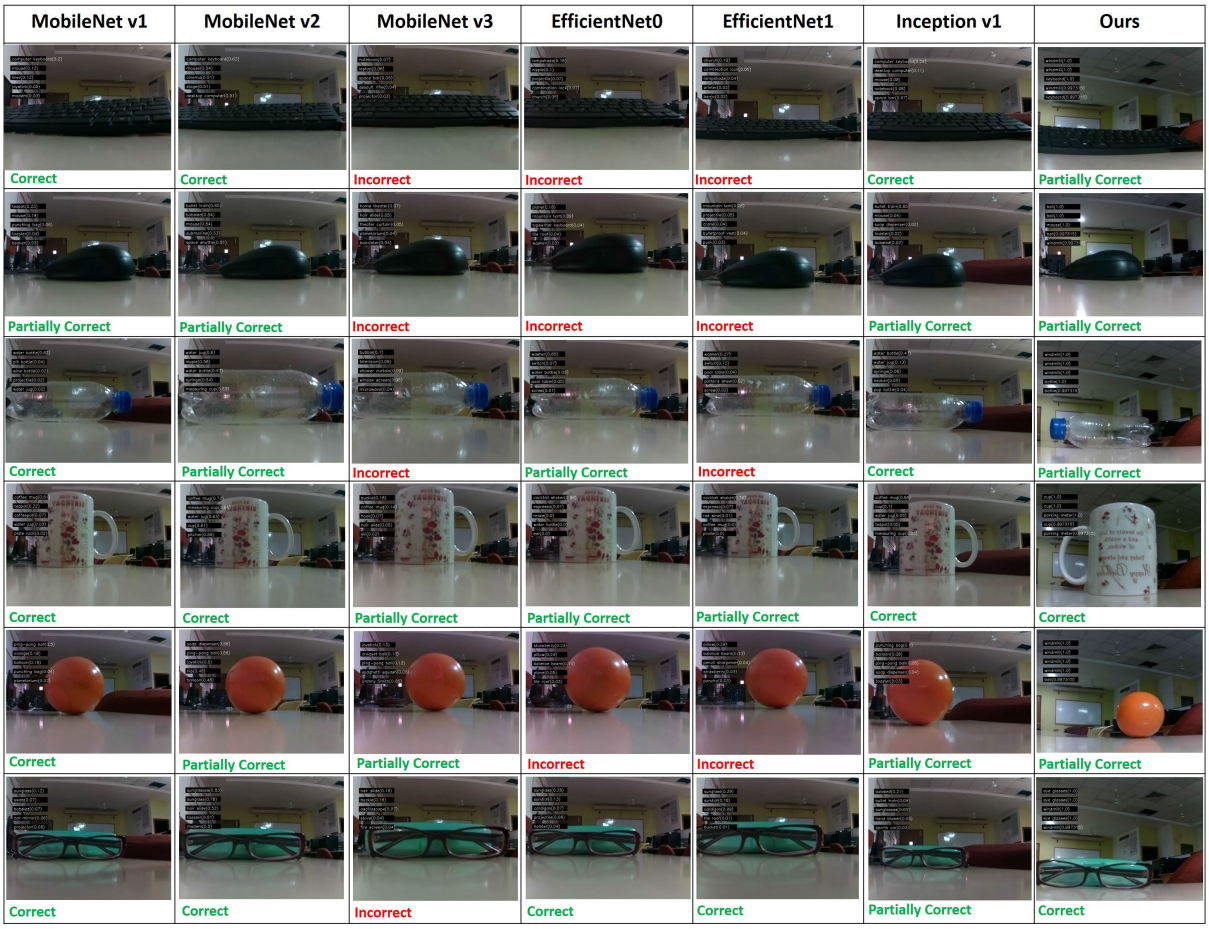

FIGURE 20. Outputs from the pretrained models compared to ours, on single objects scenes.

deploying models on constrained devices. Improving the model's performance by reducing its size, and investigating the model's architecture are not within the scope of this work. This work attempts to answer the research problem qualitatively, and could potentially be valuable for future research in this area, particularly in developing new model architectures for constrained use.

Despite the fact that this study aims to create a pipeline with an emphasis on image quality and error-free annotations, this pipeline may appear to be exhaustive if utilized on extremely large datasets due to the time-consuming nature of the process. However, when the dataset is of considerable size, and time and human resources are available, it may be an easy option to choose.

Exploring public datasets necessitates extreme caution to ensure that no licensed image gets utilized, and maintaining metadata becomes a critical responsibility since any demands for removing any image must be responded to without requiring a complete redo of the data processing pipeline, thus handling data integrity.

The work may be expanded to test the deployments on additional devices such as Jetson Nano, while using few other available models, which is considered as a future endeavor. In this attempt, just one of the existing models was used for deployment on a Raspberry Pi. However, the findings provide a stronger foundation to continue the investigations along this line of thought and thus serve as a response to the research problem.

XI. CONCLUSION

Deep learning based object detection models are increasingly being used in computer vision applications, and industries such as IoT and Robotics benefit greatly from them. There have been recent efforts that aim to bring deep learning models to resource-constrained edge devices [1]. Because the majority of models are pretrained on publicly available datasets such as ImageNet [6] and MS-COCO [7], there has been a tendency to leverage these pretrained models and fine-tune them to the desired applications via knowledge distillation/transfer learning.

First, images in these public datasets are often taken using high-resolution cameras, which differ significantly from the low-resolution noisy images adopted in real-time constrained device applications, and the processed and improved images that were used to train these models are usually unavailable. Second, due to the enormous volume of large-scale datasets, crowd-sourcing data annotations often prove to be the only possible option. This can cause several types of issues with annotation consistency and accuracy, which could be a primary reason why these models struggle to achieve acceptable accuracy on constrained deployments.

As a consequence, our research attempts to solve the problem by examining existing public datasets to generate a secondary image dataset that can be used to develop models for constrained usage. The research demonstrates that utilizing low-quality image datasets improves the accuracies of deep learning models used for constrained applications.

By leveraging the MOD-2022 dataset, the work can be expanded to train and test a variety of models with lower complexity, including MobileNet [45], EfficientDet1 [46], Yolov7 [47], and similar lightweight models for deployments on constrained devices. However, a significant endeavour that we will be looking into next is leveraging the dataset MOD-2022 to search for a novel model architecture designed specifically for resource-constrained devices.

XII. FUTURE DIRECTIONS

The future of data preparation is mostly focused on automating the process of image annotation. Several communities provide alternatives and tools for collaborating and annotating image collection for bounding box annotations and image segmentation [43], [48], [49], [50], [51]. However, the oddities caused by crowd-sourced annotations persist in these cases.

There are several confidential datasets that may not be made publicly available to annotate, particularly in the medical field, where making deep learning models with limited data leads to researchers opting for pretrained models re-trained on a small over-sampled dataset, resulting in the model over-fitting.

When image availability is not a problem, the human resource and time constraints of the annotation process may discourage researchers from investing in data preparation, leading to the usage of pretrained models. There are several approaches for annotating objects; however, using a model to annotate may be reliant based on how effectively the model is trained to annotate.

Furthermore, models designed for resource-constrained deployments must focus on decreasing model complexity in terms of memory and computations, while using processed low-quality image datasets for training to compensate for the accuracy drop caused by reduced model complexity.

Finally, image availability for training models is exceptionally difficult in various fields, and recently, GAN-based image generation [52], [53], [54] has been employed to provide new data for augmentations when training models in these fields. The extent to which this influences the model's effectiveness in terms of reliability and precision is debatable, and the choice is left to the researcher to decide which approach to employ for data preparation in their research.

ACKNOWLEDGMENT

The authors are thankful to Peehu from Madhava Kripa School, Manipal, for helping with the renaming of class labels during the data annotation process. They are also grateful to J. Vanajakshi from the Department of CSE, MIT, MAHE, Manipal, for providing them with the Raspberry Pi 4 development board that was utilized in this research work. This work was supported by Manipal Academy of Higher Education, Manipal under Dr. T.M.A Pai Research Scholarship with Research Registration No: 200900143–2021.

REFERENCES

[1] V. Kamath and A. Renuka, "Deep learning based object detection for resource constrained devices: Systematic review, future trends and challenges ahead," *Neurocomputing*, vol. 531, pp. 34–60, Apr. 2023.

[2] H. Hussain, P. S. Tamizharasan, and C. S. Rahul, "Design possibilities and challenges of DNN models: A review on the perspective of end devices," *Artif. Intell. Rev.*, vol. 55, no. 7, pp. 5109–5167, Oct. 2022.

[3] T. He, S. Yu, Z. Wang, J. Li, and Z. Chen, "From data quality to model quality: An exploratory study on deep learning," in *Proc. 11th Asia–Pacific Symp. Internetware*, Oct. 2019, pp. 1–6.

[4] V. Kamath and A. Renuka, "A proposed exploratory study of object detectors to learn the influence of datasets on model performance," *J. Phys. Conf. Ser.*, vol. 2161, no. 1, Jan. 2022, Art. no. 012076.

[5] M. T. Rasheed, D. Shi, and H. Khan, "A comprehensive experiment-based review of low-light image enhancement methods and benchmarking low-light image quality assessment," *Signal Process.*, vol. 204, Mar. 2023, Art. no. 108821.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
[7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Computer Vision–ECCV 2014*. Cham, Switzerland: Springer, 2014, pp. 740–755.
[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.
[9] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories," in *Proc. Conf. Comput. Vis. Pattern Recognit. Workshop*, Jul. 2004, p. 178.
[10] G. Griffin, A. Holub, and P. Perona, "The caltech 256," in *CalTech Report*. CaltechDATA, 2022.
[11] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning—A comprehensive evaluation of the good, the bad and the ugly," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 9, pp. 2251–2265, Sep. 2019.
[12] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "ImageNet training in minutes," 2017, *arXiv:1709.05011*.
[13] Y. Niu, Y. Zhong, W. Guo, Y. Shi, and P. Chen, "2D and 3D image quality assessment: A survey of metrics and challenges," *IEEE Access*, vol. 7, pp. 782–801, 2019.
[14] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Trans. Image Process.*, vol. 15, no. 11, pp. 3440–3451, Nov. 2006.
[15] Y. Deng, C. C. Loy, and X. Tang, "Image aesthetic assessment: An experimental survey," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 80–106, Jul. 2017.
[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD single shot multibox detector," in *Computer Vision—ECCV 2016*. Cham, Switzerland: Springer, 2016.
[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[22] K. Alhazmi, W. Alsumari, I. Seppo, L. Podkuiko, and M. Simon, "Effects of annotation quality on model performance," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Apr. 2021, pp. 063–067.
[23] K. He, R. Girshick, and P. Dollar, "Rethinking ImageNet pre-training," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 4917–4926.
[24] F. Rahimi-Ajdadi and K. Mollazade, "Image deblurring to improve the grain monitoring in a Rice combine harvester," *Smart Agricult. Technol.*, vol. 4, Aug. 2023, Art. no. 100219.
[25] S. Gogo and I. Musonda, "The use of the exploratory sequential approach in mixed-method research: A case of contextual top leadership interventions in construction H&S," *Int. J. Environ. Res. Public Health*, vol. 19, no. 12, p. 7276, Jun. 2022.
[26] Y.-H. Liao, A. Kar, and S. Fidler, "Towards good practices for efficiently annotating large-scale image classification datasets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 4348–4357.
[27] X. Li, T. Wei, Y. P. Chen, Y.-W. Tai, and C.-K. Tang, "FSS-1000: A 1000-class dataset for few-shot segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2866–2875.
[28] J. Brownlee, "Train neural networks with noise to reduce overfitting," in *Deep Learning Performance*. Machine Learning Mastery, Aug. 2019. Accessed: Mar. 21, 2022.
[29] R. Reed, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1999.
[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[31] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural Comput.*, vol. 8, no. 3, pp. 643–674, Apr. 1996.
[32] A. Singh, R. R. Nahak, and R. Kumar, *General Object Detection*. San Francisco, CA, USA: Kaggle, 2021.
[33] B. Muhammed, *Fruit Images for Object Detection*. San Francisco, CA, USA: Kaggle, 2018.
[34] A. Demirbas, *Vegetables Dataset for Object Detection*. San Francisco, CA, USA: Kaggle, 2022.
[35] A. Elkamel, *Corel Images*. San Francisco, CA, USA: Kaggle, 2019.
[36] C. Craine, *Wearing Mask Covid19 Yolo*. San Francisco, CA, USA: Kaggle, 2020.
[37] T. Bisht, *Monkey, Cat and Dog Detection*. San Francisco, CA, USA: Kaggle, 2020.
[38] A. Shaikh, *Common Vegetable Dataset for Object Detection*. San Francisco, CA, USA: Kaggle, 2020.
[39] D. Raj, *Pills-detection-dataset*. San Francisco, CA, USA: Kaggle, 2020.
[40] M. I. Ahmed and S. M. Mamun, *Vegetable Image Dataset*. San Francisco, CA, USA: Kaggle, 2021.
[41] K. Seth, *Fruits and Vegetables Image Recognition Dataset*. San Francisco, CA, USA: Kaggle, 2021.
[42] M. Kammakomati, *COB: Common Objects in Bathroom* (Restroom). San Francisco, CA, USA: Kaggle, 2022.
[43] B. Dwyer, J. Nelson, J. Solawetz, *Roboflow*, 2022.
[44] *Website: TensorFLow Model Hub*, TensorFlow Developers, TensorFlow V2.15.1, Zenodo, 2024. Accessed: Feb. 2, 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10798587
[45] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[46] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10778–10787.
[47] C.-Y. Wang, A. Bochkovskiy, and H.-Y. Mark Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022, *arXiv:2207.02696*.
[48] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2015, *arXiv:1603.04467*.
[49] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Assoc., 2019, pp. 8024–8035.
[50] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "DeepSense: A unified deep learning framework for time-series mobile sensing data processing," in *Proc. 26th Int. Conf. World Wide Web*, Apr. 2017, pp. 351–360.
[51] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "TinyML: Current progress, research challenges, and future roadmap," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1303–1306.
[52] M. Elasri, O. Elharrouss, S. Al-Maadeed, and H. Tairi, "Image generation: A review," *Neural Process. Lett.*, vol. 54, no. 5, pp. 4609–4646, 2022.
[53] B.-G. Han, J. T. Lee, K.-T. Lim, and D.-H. Choi, "License plate image generation using generative adversarial networks for end-to-end license plate character recognition from a small set of real images," *Appl. Sci.*, vol. 10, no. 8, p. 2780, Apr. 2020.
[54] W. Chi, Y. H. Choo, and O. S. Goh, "Review of generative adversarial networks in image generation," *J. Adv. Comput. Intell. Intell. Inform.*, vol. 26, no. 1, p. 3, 2022.

VIDYA KAMATH received the B.E. degree from Visvesvaraya Technological University, Belagavi, Karnataka, India, in 2011, and the M.Tech. degree from Kurukshetra University, Haryana, India, in 2014. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India.

She has six years of teaching experience and three and half years of research experience. Her major research interests include computer vision, deep learning, and the Internet of Things. Other areas of interests include artificial intelligence, computer graphics, embedded application development, and web designing.

RENUKA A. (Member, IEEE) received the B.E. and M.Tech. degrees from Mysore University, and the Ph.D. degree from NITK Surathkal, India.

She is currently a Professor with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India. She has published several papers in international journals and conferences. Her research interests include cryptography and network security, crypt-analysis, image processing and pattern recognition, machine learning, deep learning, 4G LTE systems, and mobile ad-hoc networks.

Dr. Renuka is a reviewer of articles submitted to various reputed journals and conferences.

VISHWAS G. KINI received the B.Tech. degree in computer science from Manipal University, India, in 2017, and the Master of Information Technology degree from the School of IT, Deakin University, Australia, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India.

He has one year of industry experience, one year of teaching assistant, and two years of research experience. His research interests include human–computer interaction, extended reality, spatial computing, affective computing, applied AI, and android app development.

SHWETHA PRABHU received the B.Tech. degree in electronics and communication engineering from Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, India.

She has presented a few papers at international conferences. Her research interests include medical image processing and cipher systems.