Object Detection and Sharp Corner Detection Model for Baby Proofing

PROJECT REPORT

SUBMITTED BY:

Aasawaree Deshmukh
Nishant Priyam

UNDER GUIDENCE OF:

Prof. Vir Phoha

May 08, 2018

Object Detection and Sharp Corner Detection Model for Baby Proofing

Aasawaree Deshmukh, Nishant Priyam
Syracuse University, Syracuse, NY, USA, 13210
{asdeshmu, npriyam}@syr.edu

ABSTRACT - We are proposing a new model for classifying and identifying possible threats in the vicinity for a new born baby or a toddler via the concept of baby proofing using Machine Learning and Deep Learning. As of now, there is no significant work done in this field. However, there is some work done in the domain of object detection and corner detection. After detecting the objects which might be harmful for a baby, we detect what areas within the object would require proofing solutions. This will help parents take precautionary measures which are necessary to baby proof the surrounding.

I. INTRODUCTION

Having a baby for the first time offers so much joy-an unexplainable happiness that can never be equaled by anything in the world. However, this experience comes with a lot of responsibility and parents who are experiencing this for the first time may find it overwhelming. A child is full of energy but fragile. They require your constant love and attention, and safety always comes first.

Babies of the age group 0-3 years; i.e. new-born to toddlers, are totally unware of their surroundings. They are young and are naturally curious which sends them out to explore. This may lead to some serious injuries or even some life-threatening events. According to the Centre for Disease Control and Prevention (CDC) [21] approximately 12,000 children and young adults, risk their lives due to unintentional injuries that happen at home. There is a high risk of being injured at home because that's where they spend most of their time.

To prevent the child from getting hurt, one precautionary measure which can be taken is to access the surrounding for any lurking danger and baby proof the surrounding for the child. This can be achieved by doing it yourself or by hiring a professional for the task. However, there might always be a possibility of an object or an area which might get neglected and would require baby proofing. It's always better to be safe than sorry.

II. PROBLEM STATEMENT

Many people may refer to basic resources available online which provides guideline for different solutions used for an infant/child proofing. These baby proofing solutions vary based on different age groups. Based on this available information, they baby proof their space. Some may also seek help from professional childproofing services due to their lack of experience. Few of these hired professionals might charge separately for an inspection or consultation and installing the baby proofing solutions, or as part of a package deal.

Even if you've read books, researched online or taken an advice from everyone you know, and then taken all the precautions you can think of, dangers lurk below eye level. How can we ensure that everything is safe and sound? How can we ensure that there is no human error encountered while proofing the space? How can we ensure and avail a sure-shot cost effective mechanism for the safety of our children? All these questions define our problem statement, where we developed a bare minimum cost-effective mechanism to identify objects which may pose as a threat for a child and then evaluate these detected objects for any sharp edges or corners.

III. RELATED WORKS

Object detection comes under computer vision research world. The research area is basically a fusion of Machine Learning, Deep Learning and Neural Network. Machine learning is used for feature extraction of objects from images or any video that we feed in the model. It also uses the concepts of transfer learning. Deep learning and Neural Network are used for creating model as well as API where we pass an image or video as an input and receive an object identified or labelled or recognized with certain amount of accuracy as an output.

In past few years, there is a rapid industrial growth in the field of object detection since machine learning was introduced. In most of the computer vision systems, the first task implemented was object detection where system predicts or recognize the object (e.g. a chair or table) [8]. In 2018, smartphones adapted recognition system and call themselves as AI enabled smartphones. When Google introduced TensorFlow Object Detection API [11][14], it was a revolution in the world of developers as it simplifies identifying objects within an image along with improved performance. Convolution neural network-based models were introduced for handling tasks like facial detection and landmark recognition. Big names like Facebook, Apple and Google then released their own light and easy to implement open source version of object detection model and API which were focused for smartphones as well as heavy duty machines were being developed [19]. Since this is one of the burning topics in the field of Machine Learning, there are multiple research papers and open source projects repositories released on this.

Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image [6]. Corner detection is frequently used in motion detection, image registration, video tracking, image mosaicking, panorama stitching, 3D modelling and object recognition. A common low-level operation in many computer vision systems is the detection of "interesting" parts, or features, within an image. Corners are point-like features in an image, which have a two-dimensional structure, such as where two edges come together, describing high levels of curvature of image gradients in a region [1]. There exist several classical corner detection algorithms for estimating corner points. Such detectors are based on a local structure matrix which consists on the first partial derivatives of the intensity function. A clear example is the Harris feature point detector [2][3], which is based on a comparison: the measure of the corner strength - which is defined by the method and is based on a local structure matrix - is compared to an appropriately chosen concrete threshold. Another well-known corner detector is the SUSAN (Smallest Univalue Segment Assimilating Nucleus) detector which is based on brightness comparison [2][4]. It does not depend on image derivatives. The SUSAN area will reach a minimum while the nucleus lies on a corner point. The effectiveness of the above-mentioned algorithms is acceptable. Recent studies such as [2][5] demonstrate that the Harris corner detector performs better for several circumstances in comparison to the SUSAN Algorithm.

IV. Approach

A. Object Detection Module

The first thing that we did was to detect object in order to find the corner points within that object. For detecting the object, we created the data set. To limit the scope of the project in regards with the given time frame, we focused on detecting one object, i.e. table and finding it's corner using our own sharp corner detection algorithm (which is described in next module). There are already existing universally accessible prebuild data sets of some objects, such as image dataset on ImageNet [17], which gave us image dataset for a room, GitHub open images data set [18] etc. However, there was no prebuild data set available specially for table, chairs, electrical sockets, etc. Hence, we had to manually create our own data set. In order to achieve this, we manually scrapped images for table from Google Images and Yahoo Images, restricting our search for images between the dimension of 100x100 to 200x200. After scrapping approximately 250 images for tables, we did image verification and dimension deduction on the images within our dataset so that we have a uniform set of images within our data set.

The next step was to create bounding boxes and label these images. There are multiple open source software which create bounding boxes and label them such as LabelMe, RectLabel, labelbox, LabelImg [15] etc. The problem with LabelMe and Labelbox was, they are cloud-based software where our data gets saved after execution of our work and can be publically accessible by anyone without any permissions required. To avoid this, we decided to use LabelImg [16]. LabelImg is an open source software where we manually create bounding boxes and assign a particular label. The main advantage of using LabelImg was that we can label multiple objects within a single image. We created the bounding boxes for all the images and labelled them. This led to the generation of an .xml file for each image which was saved in the same folder as that of the image dataset. This file consisted information regarding the name of the label, bounding box co-ordinate (xmin, ymin, xmax and ymax) and height, width and depth of the source image and label of the object. To co-relate this file with the respective image, the name for the .xml file had to be same as the source image file. Bounding boxes also helped us to extract features of the object like height, width, classes of object and class text of the object. Later we changed the .xml file to .csv file in order to train and test our module.

After extracting the features of the object from an image, we needed to create a TFRecord file (TensorFlow's file format). When we used the training model on the image data set that we created, most of the batch operations were not directly done on images. Images are converted into numpy (python library) arrays and labels are converted into lists of string which are then used by them. TFRecord file contains these converted formats of images and labels. For creating the TFRecord file, we used pre-existing code written in python called generate_tfrecord.py. Before generating the TFRecord, we generated the class number and index integer for each object label. Index integer is used by neural network cross-entropy function which helps to manipulate the performance of the network while classifying. Class number is generated in the same order as the trainable file class is defined. We also had to check if the bounding boxes coordinate are relative to image coordinate, so that we don't face any error or exception while training the model. After this, we created a TFRecord file.

The next step was to set up the object detection API. We installed TensorFlow [12] and necessary system dependencies through pypi. Now, we will use the concept of transfer learning. Transfer learning is a concept as well as research area of Machine Learning which means that we can use the gained knowledge that we gathered while solving one problem to different but relatable problem. For example, if we train a model for detecting raccoons, we can use the same model for detecting some other objects by just passing another data set of the object we want the model to detect.

For this project, we decided to use different neural network configuration to train our model to analyze which configuration is suitable for the system. Due to the time restriction, we decided to go with only two configurations. They are

faster_rcnn_resnet50_lowproposals_coco_2018_01_28 and ssd_mobilenet_v1_coco_11_06_2017. Both of them are pre-trained open source configuration models existing on the GitHub repository. Now, we cannot directly start training our model. We have to make necessary changes in the configuration file according to our requirements. Before training our model, we changed the num_classes from the configuration file to 1 because we have only one label. num_classes define the total number of classification labels. In the rest of the configuration file, hyper-parameters of various layers are defined. After that, we checked the train_config file that is present in the model. We changed the batch_size to 10 (batch_size is number of elements that the model is using for training in a single instance). Since our machine has no GPU and has 4 GB RAM, we decided to go with batch_size of 10. We transferred all the training models and image data set folder which contains both train and test sets of images to the object detection model folder configuring our API.

The third step is to train the model using train.py (which was in object_detection folder of models from GitHub repository [20]). In order to do that, we ran the train.py for training the model and saved the checkpoints in the training_output directory, so that if we have to resume the training, we have previous checkpoints from where the model will reload and start training again. While training the model, we ran the tensorboard daemon on another terminal from the training folder. We can track the average precision using tensorboard and can also see how effectively our model is detecting the required object. We get the total_loss graph, configuration_loss graph, different histograms, etc. Training a model usually takes time. For our project, we used MacBook Air with 4GB RAM (no GPU) to train the model. While training our model with SSD configuration, it took approximately 2.5 hours to train until we started to get flat line of average precision on the Tensorboard. When we tried to train with faster_rcnn, it trained for 8 hours and stopped by itself because multiple exception of low memory in disk, system overload etc. were generated. Initially, to process a single step, SSD was taking approximately 35-50 seconds. But when we tried faster_rcnn code, it was taking 500-850 second to run a single step while training. This shows huge difference of time to run a single step between SSD and faster_rcnn. Since faster_rcnn is unable to run properly in a machine with low configuration and also takes much more time to then SSD to train, we decided to proceed with SSD for our project.

The final step was to check if our model is providing us the desired output or not. We ran a program written in python3 (which was present in the object_detection module [19]) on jupytor notebook to validate the same. When we checked for SSD model, it gave us 70%-95% accuracy. In few images, the bounding boxes were not accurate. For example, all the corner points of the table were not in the bounding boxes in some of the images. When we tried with faster_rcnn, the output was 80% to 95% but the bounding boxes for the object were much more accurate.

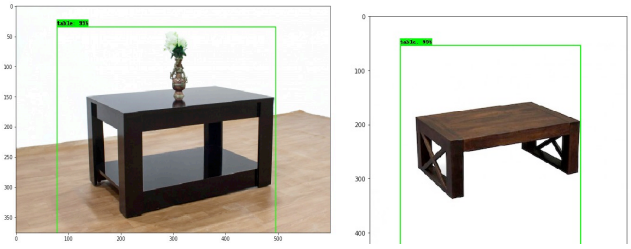


Figure 1: Model with SSD configuration output

B. Sharp Corner Detection Module

Once the object is detected within the image, we need to read the information within that object to deduce the corner. In order to achieve optimal results, we researched about various existing techniques for corner or sharp edge detection. One of the earliest corner detection techniques is the Moravec corner detection algorithm [6]. The algorithm tests each pixel in the image to see if a corner is present, by considering how similar a patch centered on the pixel is to nearby, largely overlapping patches. The similarity is measured by taking the sum of squared differences (SSD) between the corresponding pixels of two patches. A lower number indicates more similarity. The corner strength is defined as the smallest SSD between the patch and its neighbors (horizontal, vertical and on the two diagonals). As pointed out by Moravec, one of the main problem with this operator is that it is not isotropic: if an edge is present that is not in the direction of the neighbors (horizontal, vertical, or diagonal), then the smallest SSD will be large and the edge will be incorrectly chosen as an interest point.

After Moravec algorithm, there were few improvements done using Harrison and Stephens algorithm. Harris and Stephens improved upon Moravec's corner detector by considering the differential of the corner score with respect to direction directly, instead of using shifted patches. It measures the change in intensity for the shift in all directions.

There are many other corner detection techniques available, however these algorithms didn't meet our requirement for finding relevant sharp corners as per our project. We needed only those interest points which would pose as a threat for a baby and needs proofing. In order to optimize the already existing algorithms, we carved out our own approach by combining already existing methods / algorithms.

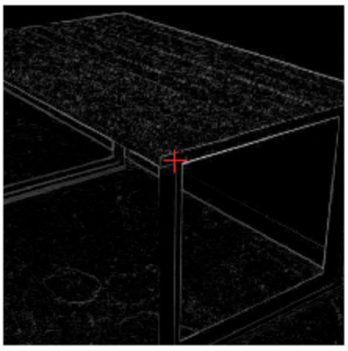
Firstly, we needed to reduce the noise within the image. A colored image consists of 3 channels - RGB channels. For our project, color information doesn't help us identify sharp edges or corners. Hence, we convert the image to single channel, i.e., gray scale and read its information. This is achieved using the skimage python library. Using TensorFlow, we apply two filters to extract the horizontal and vertical edges using the emboss horizontal and vertical filter namely, kernel_h = [1, 2, 1], [0, 0, 0], [-1, -2, -1] and

kernel_v = [1, 0, 1], [2, 0, -2], [-1, 0, -1] [11]. We then activated the tensor flow session using tensor flow name_scope and using 1-D shape array - (3, 3, 1, 1) and strides array - [1, 1, 1, 1] to generate the output vectors. Using trigonometric functions [11], we then calculate the strength and angle for the result output vectors using both horizontal and vertical vectors. Within an image, if there is a low strength area, it indicates that there is noise within that region as the ratio between x and y axis swings easily. To correct the same, we added a small value so as to avoid zero error issues. These vectors are then used to generate respective normalized form vectors. We then plot these result vectors using python matplot library. The resultant image consists the outline or edge of the detected object [11].

Using the skimage python library, we then make use of predefined corner detection algorithms. Here, we needed to compare and validate which algorithm would give the correct set of interest points. We made use of fast corner detection algorithm. FAST (Features from Accelerated Segment Test) algorithm was proposed by Edward Rosten and Tom Drummond in their paper "Machine learning for high-speed corner detection" in 2006 (Later revised it in 2010) [13]. However, with this algorithm we observed that it detected multiple corner points. These interest points were not highly relevant as per the requirement of our project. For example, if a table is detected then as per our project, the interest points should only consist of those 4 corners.

We then made use of Harris corner detection algorithm [3]. Here also there were other corner points detected apart from corner of the table, but ratio of irrelevant interest point was less as compared to Fast corner detection [9]. After combining the edge detection and the corner detection algorithm, we get an optimum result in terms of the desired corner point detection. Below are the images of a table which shows comparison of Fast and Harris corner detection after applying edge detection algorithm.

(a) Original Image (b) Using Fast Corner



(c) Using Harris Detection

Figure (2)

C. *Combination of above Modules*

Now we have two separate modules which are detecting objects from an image in a bounding box and corners of the object in the image. Once we get the output from the object detection module, we are feeding them as the input of the second module, i.e. sharp corner detection module. In that module, only objects within the bounding boxes are considered for finding the sharp corner. The main reason of creating bounding boxes was to eliminate irrelevant area and only consider objects that are within the bounding boxes. This reduces our area of interest and gives us a particular target to focus for corner detection algorithm. Hence, after getting the image within the bounding box, we are running a python notebook file which gives us corner points marked in the red cross for that particular object which requires baby proofing solutions.

V. COMPARISONS

We used two neural network configuration files to check which one is best for a normal configured machine (with at least 4 GB RAM and Intel Graphics Card) to predict and recognize a particular object. The first one was SSD configuration model and the second was faster_rcnn configuration. While SSD configuration was taking 35-50 seconds per steps during training, faster_rcnn was taking 500-850 seconds per step.

```
INFO:tensorflow:global step 79: loss = 1.7855 (33.181 sec/step)
INFO:tensorflow:global step 80: loss = 1.0411 (25.682 sec/step)
INFO:tensorflow:global step 81: loss = 1.6526 (24.977 sec/step)
INFO:tensorflow:global step 82: loss = 1.7517 (22.612 sec/step)
INFO:tensorflow:global step 83: loss = 1.1813 (23.565 sec/step)
INFO:tensorflow:Recording summary at step 83.
INFO:tensorflow:global step 84: loss = 2.8032 (25.602 sec/step)
INFO:tensorflow:global step 85: loss = 5.4917 (31.108 sec/step)
INFO:tensorflow:global step 86: loss = 4.2198 (26.381 sec/step)
INFO:tensorflow:global step 87: loss = 1.2475 (21.170 sec/step)
INFO:tensorflow:Recording summary at step 87.
INFO:tensorflow:global step 88: loss = 1.1407 (25.430 sec/step)
INFO:tensorflow:global step 89: loss = 1.8902 (22.037 sec/step)
INFO:tensorflow:global step 90: loss = 1.7079 (22.074 sec/step)
INFO:tensorflow:global step 91: loss = 1.0071 (22.054 sec/step)
INFO:tensorflow:global step 92: loss = 2.2269 (20.247 sec/step)
INFO:tensorflow:global step 93: loss = 2.1500 (21.497 sec/step)
INFO:tensorflow:Recording summary at step 93.
INFO:tensorflow:global step 94: loss = 1.1447 (26.694 sec/step)
INFO:tensorflow:global step 95: loss = 0.9131 (22.735 sec/step)
INFO:tensorflow:global step 96: loss = 1.1507 (23.241 sec/step)
INFO:tensorflow:global step 97: loss = 1.0641 (23.727 sec/step)
INFO:tensorflow:global step 98: loss = 2.2765 (27.299 sec/step)
INFO:tensorflow:Recording summary at step 98.
```

Figure 3: Time Required per Step (SSD configuration)

```
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global step 1: loss = 6.2972 (521.588 sec/step)
INFO:tensorflow:Recording summary at step 1.
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:global_step/sec: 0.00843946
INFO:tensorflow:global step/sec: 0
```

Figure 4: Time Required per Step
(faster_rcnn configuration)

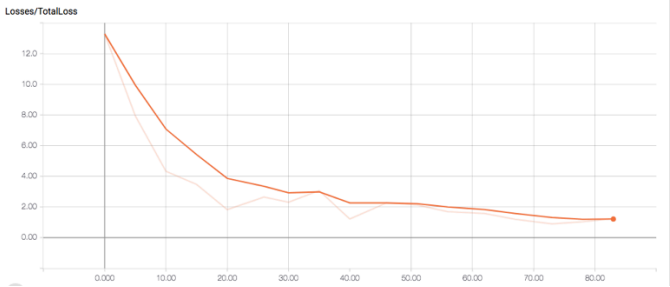


Figure 5: SSD model total loss graph

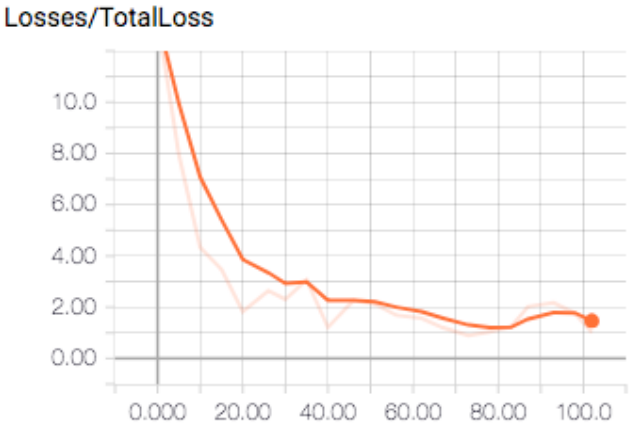


Figure 6: faster_rcnn model total loss graph

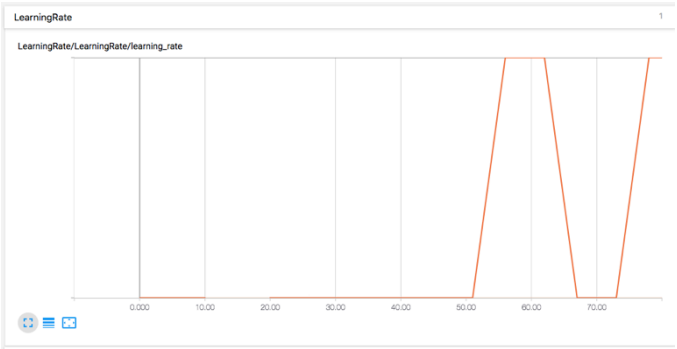


Figure 7: Learning rate of SSD model

Figure 8: Learning Rate of faster_rcnn model

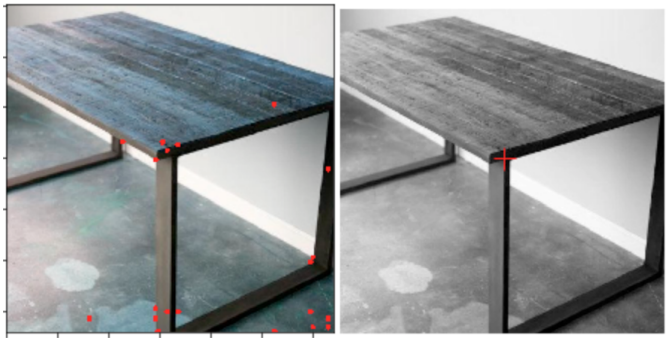
We can clearly see from the graph that the learning rate of faster_rcnn is slow in comparison to SSD configuration. Also, the time taken to achieve straight line in total loss graph is more in faster_rcnn as compared to SSD configuration. This signifies that it takes more time and machine power to train the model using faster_rcnn configuration then SSD configuration. On the other hand, faster_rcnn gives more accurate and precise recognition of object then SSD configuration. But the accuracy difference is less then 10% to 20%. While faster_rcnn gives us an accuracy of 85% to 95%, model trained with SSD configuration gives us accuracy of near about 80% to 90%. When we compare the ratio of required time to train the model with the accuracy, SSD is more suitable to use for a machine with normal configuration for object detection purpose. That is why, for this project, we are using model trained with SSD configuration for object detection and recognition.

Various existing corner detection techniques were tested to validate proposed detection of sharp corner points. We started this by simply implementing and testing multiple corner detection algorithms using python. The first algorithm we experimented with was Harris Corner Detection Algorithm [6]. In Figure (9) (b) and Figure (10) (b), we can observe that the algorithm recognized not only valid sharp corner interest points but also other various non-sharp corner interest points which are marked using a blue color blob. The second experimentation was carried using the Shi-Tomasi Corner Detection Algorithm [6]. The results for this algorithm can be observed in Figure (9) (c) and Figure (10) (c). The difference with this algorithm is number of relevant sharp points detected. However, using our proposed algorithm where we combine edge detection and corner detection algorithms with some modifications, we can observe that there is some improvement with regards to the number of relevant interest points detected required for baby proofing. Having said that, we also observe that our proposed algorithm has an error factor. We can see the variation while comparing Figure (9) (d) and Figure (10) (d). In Figure (9) (d), it was accurately able to

detect the relevant point which required baby proofing, but in Figure (10) (d) we can see that our proposed algorithm detected a few other points in addition to the relevant sharp corners points. But one improvement we perceive from Figure (9) (d) and Figure (10) (d) is that we are now able to detect normal sharp corners with more accuracy in comparison with just implementing Harris and Shi-Tomasi Corner detection algorithm. While Moravec takes sum of squared differences (SSD) between the corresponding pixels of two patches, Harris takes the second derivative of the SSD and Shi-Tomasi takes the smallest Eigen values.

(a) Original Image

(b) Harris Corner Detection Algorithm



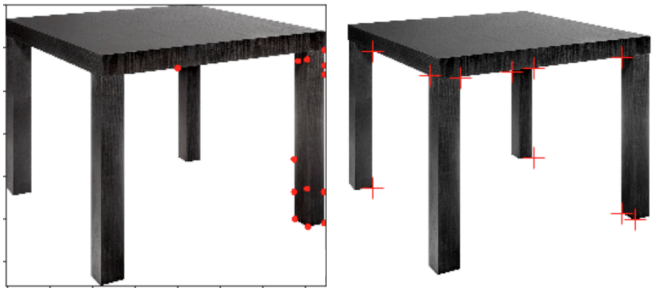
(c) Shi-Tomasi Corner Detection Algorithm

(d) Our Proposed Algorithm

Figure (9)

(a) Original Image

(b) Harris Corner Detection Algorithm



(c) Shi-Tomasi Corner

(d) Our Proposed Algorithm Detection Algorithm

Figure (10)

VI. Accomplishments

With the help of this problem statement, we were able to achieve some improvement while detecting sharp corners which needed baby proofing. In the due course of this project, we learned about various image processing concepts, multiple image analysis and corner detection techniques along with differences and improvements in each of those algorithms, namely, Moravec Algorithm, Harris and Stephen Corner Detection, Shi-Tomasi/Fast Corner Detection Algorithm [6] etc. We also learned about how to extract features for a particular object and then evaluate them based on the extracted features. Another learning we acquired was TensorFlow – an open source machine learning framework [12]. This was mainly used in applying sobel emboss filters to extract the horizontal and vertical edges within an object detected. Based on the output of this step, we then used the existing corner detection algorithm to extract the relevant points needed for baby proofing. We learned about TensorFlow object detection API [7] which can be implemented using various neural network configuration. We also learned about Keras which is a high level neural network API, written in python and can run on the top of TensorFlow [10]. Apart from that we got an insight about how different convolution neural network works on an image, how they divide them into grids and arrays and individually analyze the layers according to the hyper parameters defined within that configuration. We also came across many libraries of python like numpy, pandas and matplotlib etc.

VII. Challenges

There were a lot of challenges we faced while execution of our project. During implementation of the first module, setting up environment was one of the biggest challenge. We had to make sure that we had every required file, libraries and API installed. Since we were not having a prebuild labelled data set, scrapping appropriate image manually was also one of the challenges we faced. Resizing and labelling of images was the next problem. As we were having about 250 images of the object, we had to manually do the labelling and resizing. After resizing, we verified the

images which was a time-consuming process. Another major challenge was to train the model. Even though we had a pre-tuned and pre-trained model, different configurations took different time. If we consider training time for both the models together, it took more than twenty hours in one single run. Machine power was an additional obstacle we had to deal with since object detection requires a lot of GPU and CPU power. The machine on which we worked had a configuration of 4 GB RAM with Intel Graphics Card. While training the model we encountered system crashes due to insufficient machine power. However, we managed to train both models for object detection in order to compare and choose which model works best for a normal machine. We also faced a few challenges while detecting the right set of sharp corner points. Not only we had to research and experiment with various algorithms, but also evaluate the differences in each of them. When we finalized on one algorithm we tested the same by changing filters applied for edge detection, minimum window size and threshold values to extract the relevant interest points in corner detection algorithm.

VIII. Acknowledgment

We would like to thank Prof. Vir Phoha for his continuous guidance through the challenges we faced and supported us in the same. We would also like to thank Anirudhda Deshmukh for his help and guidance in brainstorming different ideas for machine learning project and research proposal.

IX. Conclusion

Thus, using the above proposed algorithm, we are successfully able to detect objects within a given image and extract or detect the interest points which might be harmful for a baby and requires proofing solutions.

X. References

[1] "A Machine Learning Approach to Corner Detection," [Online]. Available: http://mcclanahoochie.com/blog/wp-content/uploads/2011/05/main3.pdf.

[2] *. Z. ,. *. P.-C. +. S. A. -M. R.-O. ERIK CUEVAS, "ROBUST FUZZY CORNER DETECTOR," [Online]. Available: https://arxiv.org/pdf/1405.5422.pdf.

[3] "Harris, C. and Stephens, M., A combined corner and edge detector, in: Proceedings of the 4th Alvey Vision Conference, 1988, pp. 147–151. 11. Smith, S. and Brady, M., "A new approach to low level image processing", Int. J. Comput. Vision 23 (1) (1997) 45–78. 12. Zou, L., Chen, J., Zhang, J., Dou L.: The Comparison of Two Typical Corner Detection Algorithms, Second International Symposium on Intelligent Information Technology Application ISBN 978-0-7695-3497 (2008).," [Online].

[4] " Smith, S. and Brady, M., "A new approach to low level image processing", Int. J. Comput. Vision 23 (1) (1997) 45–78.," [Online].

[5] "Zou, L., Chen, J., Zhang, J., Dou L.: The Comparison of Two Typical Corner Detection Algorithms, Second International Symposium on Intelligent Information Technology Application ISBN 978-0-7695-3497 (2008)," [Online].

[6] "Corner_detection," [Online]. Available: https://en.wikipedia.org/wiki/Corner_detection.

[7] "Object detection API," [Online]. Available: https://techcrunch.com/2017/06/16/object-detection-api.

[8] "www.frontiersin.org," [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2015.00029/full.

[9] "FAST Corner Detection Algorithm," [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_fast/py_fast.html.

[10] "Keras," [Online]. Available: https://keras.io/.

[11] "TensorFlow Image Convolution Edge Detection," [Online]. Available: http://thegrimm.net/2017/12/14/tensorflow-image-convolution-edge-detection/ .

[12] "TensorFlow," [Online]. Available: https://www.tensorflow.org/.

[13] "Edward Rosten Machine Learning Approach," [Online]. Available: https://www.edwardrosten.com/work/rosten_2006_machine.pdf.

[14] "TensorFlow Image Recognition," [Online]. Available: https://www.tensorflow.org/tutorials/image_recognition.

[15] "Suggest Image Labeling Tool for object detection," [Online]. Available: https://www.researchgate.net/post/Can_anyone_suggest_an_image_labeling_tool_for_object_detection.

[16] "LabelImg," [Online]. Available: https://github.com/tzutalin/labelImg.

[17] "ImageNet," [Online]. Available: http://www.image-net.org/.

[18] "Github Image Dataset," [Online]. Available: https://github.com/openimages/dataset.

[19] "Object Detection TensorFlow," [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb.

[20] "TensorFlow Object Detection Github," [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection.

[21] N. C. f. I. P. a. C. D. o. U. I. P. Centers for Disease Control and Prevention, "CDC Childhood Injury Report," 2008. [Online]. Available: https://www.cdc.gov/safechild/child_injury_data.html.