Object Detection and Counting with Low Quality Videos

Hao Jiang
Mechanical Engineering
Stanford University
jianghao@stanford.edu

Shiquan Wang
Mechanical Engineering
Stanford University
shiquan@stanford.edu

Abstract

Enabling intelligent devices to accurately detect and identify the environment with various disturbances is very important. In this final report, we present an object detection approach for low quality videos. Still images with motion blur, Gaussian blur and pixelation are trained in addition to clean images using both faster R-CNN and YOLO algorithms. The models are integrated with low resolution webcam videos and evaluated with an object counting agent. The final models demonstrate improvements in both low quality images and videos. The cause of errors are discussed.

1. Introduction

Computer vision has recently been greatly integrated into robotics and automation. From camera manufacturing [2] to quadcoptor SLAM and navigation [8, 6], image classification and object detection has been widely applied to achieve more accurate localization and faster computation. However, due to environmental disturbances such as wind gust, rain, and lack of lights, image quality can reduce to different extent. This problem is especially severe on webcam vision which is broadly used in many robots. Thus a desirable detection algorithm for robot vision should be robust enough to tolerate various image quality reductions and efficient enough for robot onboard real-time detection.

The application we are looking into is enabling robot to count objects in a comparatively fixed scene, such as counting number of people in certain store or number of cars in certain parking lot. It is a sophisticated enough problem that needs to resort to deep learning algorithms.

Low quality images can be caused by fast motion of webcam or target, unfocused lens, lack of light exposure and etc.. A fast moving scene can be simulated with a motion blur filter, which acts on the original image as a moving average along arbitrary directions. Unfocused lens can be modeled as Gaussian blur. Low resolution (small) images, when applied to convolutional networks, can be modeled as resized, larger images with pixelation. The above low quality filters are applied to still images for training. Test videos cover a range of qualities.

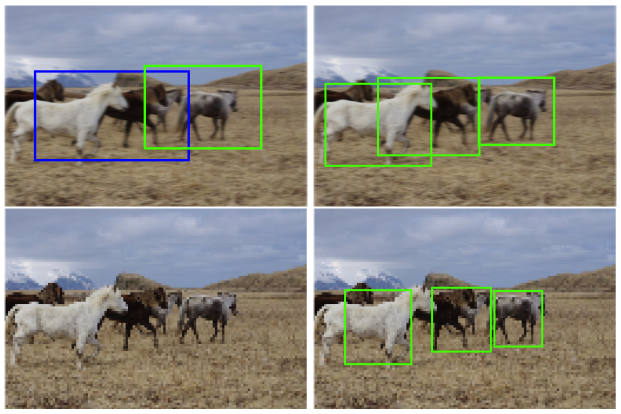
Figure 1. An original model VS a re-trained model on reduced quality images. The re-trained model improves detection accuracy and robustness. **Top left:** The original model on a motion-blurred image. **Top right:** The re-trained model on the same motion-blurred image. **Bottom left:** The original model on a pixelated image. **Bottom right:** The re-trained model on the same pixelated image.

2. Background

Object detection has been an active research field for decades. Recent approaches using convolutional neural networks (CNN) have improved the detection accuracy by a large extent [7, 13, 5]. Moreover, objection detection with reduced image qualities have also aroused people's attention [9, 10], but there is little literature on using deep CNN's for such problems.

Among the many choices of object detection with CNN's, faster R-CNN and YOLO [12, 11] are the two most promising ones that require comparatively low computational power but still achieve good enough accuracy, therefore are the main focus in this paper.

3. Approach

3.1. YOLO

YOLO stands for "You Only Look Once". It splits a source image into 7×7 grids, and for each grid it predicts two bounding boxes and a class association with probabilities. All layers before the final prediction layer are similar as regular CNN's in terms of architecture. The final detection layer instead is a regression that maps the output of the last fully connected layer to the final bounding box and class assignments. The illustration of the detailed structure is shown in Fig. 2. Because of the simple structure, YOLO is comparatively much faster than most region-proposal algorithms and suitable for real-time applications.

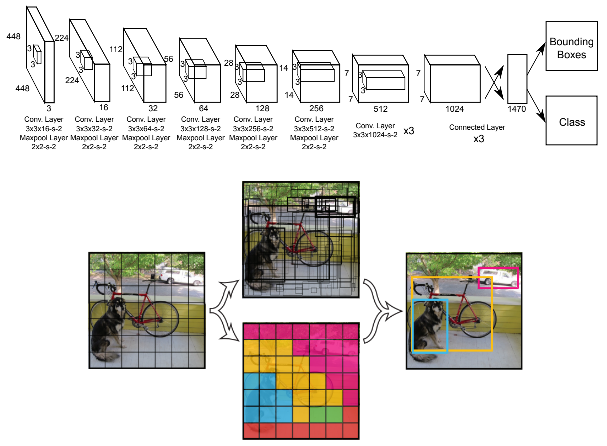
448
224
112
56
28
14
7
3
16
32
64
128
256
512
1024
1470
Conv. Layer 3x3x16-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x32-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x128-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x256-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x512-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x1024-s-2 x3
Connected Layer x3
Bounding Boxes
Class

Figure 2. **Top:** The original YOLO's architecture. **Bottom:** Illustration of YOLO dividing a source figure into grids and assigning bounding boxes and class to each grid.

We used YOLO-tiny model as the pre-trained model because it is the fastest model so far and can do detection at more than 100 fps [11]. Smaller than the original YOLO, YOLO-tiny has 9 convolutional layers, 6 max pooling layers, and 3 fully connected layers,which is shown in Fig. 2. The pre-trained model has already been trained on the VOC 2007 and 2012 for 40,000 iterations with each iteration looping over 64 images as one batch. Thus, this model is highly over-fitted to the VOC clean images. We fed reduce quality images into the pre-trained model for re-training and fine-tuning. For each category of reduced quality images, we re-trained the model for 500 to 1,000 iterations with 128 images (batch size) per iteration. Learning rate was controlled to be as low as $1e^{-4}$ to $1e^{-5}$ to prevent unstable gradients.

3.2. Faster R-CNN

R-CNN is a neural network architecture for object detection, with region proposal network (RPN) which extracts object candidates from raw image and have them processed by convolutional networks [12]. More specifically, a convolutional network (ConvNet) pretrained by ImageNet [4] has its last fully-connected (FC) layer removed and retrained for new set of classes required by the object detection task. All regions acquired by the region proposal methods are processed by the ConvNet into features that will be stored in the disc. Binary SVMs and regression for bounding boxes are then trained based on the features. This method is expensive in both space (storing huge size of features) and time (processing each region separately). Fast R-CNN has been developed to overcome these disadvantages [12]. It swaps the pipeline and process the images with ConvNet altogether first and then based on which use external region proposal for region of interests and pass them into layers for classifications and bounding box regression. Faster R-CNN [12] replaces the external region proposal methods with the region proposal network (RPN) which enable end-to-end training and testing, in order to achieve close-to-realtime inference speed. The architectures of R-CNN variants are shown in Fig. 3.

In this project, faster R-CNN are selected as one of the main approaches for its comparatively good training/testing efficiency and great accuracy. VGG16 is used for the ConvNet kernel. Approximate joint training method was used on the faster R-CNN to achieve 1.5x faster training time.

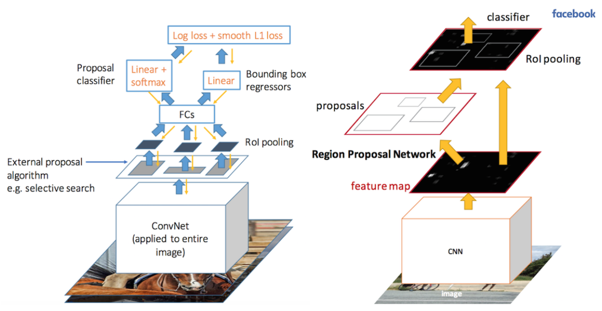

Figure 3. **Left:** Architecture of fast R-CNN and **Right:** architecture of faster R-CNN.

Based on the insights acquired from YOLO retraining, a faster R-CNN were also retrained to increase its performance on low quality images. The model was initialized with weights that have been trained on VOC2007 images. It was then trained with images processed by Gaussian blur, motion blur and pixelate filter. The image processing was implemented using OpenCV. The model was also retrained using VOC2012 images for better generality. Different sequences and combinations of successive training were experimented. The size of batch and learning rate were tuned to achieve better training efficiency and set to be 128 and 0.0002 for the final training.

3.3. Model Implementation

The code and pre-trained weights of YOLO are originally from [3]. This framework is called Darknet, which is implemented with C. Re-training of YOLO were accomplished on Stanford Terminal Cloud [14].

The implementation of faster R-CNN are adapted from codes [1] developed using Caffe. AWS EC2 g2.2xlarge server with GPU was used for model training. For ease of video detection and rendering, we used a desktop equipped with GeForce GTX 980 Graphic card for the testing.

3.4. Low Quality Image Filters

There are 3 filters used in the training data set to reduce the image qualities, known as motion blur, Gaussian blur, and pixelation. Motion blur acts as a moving average function applied to a sliding window traversing the whole image along a fixed direction. Gaussian blur acts as a Gaussian function convolving with the original image to reduce details. Pixelation acts as an average pooling across the whole image. They are used to get better tolerance to images with wrong focus, fast moving objects and small size objects respectively. In this work, a motion blur filter with window size of 8 pixels and orientation of 0^o is used. Gaussian blur of window size of 8 pixels and standard deviation of 2 is applied. For pixelation, a series of 2×2 through 8×8 average pooling filters are used to optimize robustness. Note that the pixelate images for faster R-CNN were obtained by down sampling and then up sampling the images using OpenCV. Example of an original image and the corresponding several filtered images are shown in Fig. 4.

3.5. Object Counting Benchmark

Based on the two aforementioned pretrained model, we developed an object counting agent for video input. The scripts were written in python with interface for the object detection model. Only two classes: human and car, is processed in the agent, for scenarios of counting people in indoor environment (restaurant etc.) and cars in parking lots. The numbers of persons and cars are rendered on the top left corner of the video in each frame.

For evaluating performance of the object counting agent, we chose 8 different videos, 4 for each class, for the benchmark. Snapshot of each video sample is shown in Fig. 5. Detailed descriptions of each video are as follows:

- Video1: small group (less than 10, sparsely distributed) of people in indoor environment, with dark background. Fast motions are involved.

- Video2: small group (less than 12, sparsely distributed) of people in indoor environment (restaurant), with bright color background. Motions are mostly slow. Object size ranges from large to very small (down to 1/10 of image height and weight).

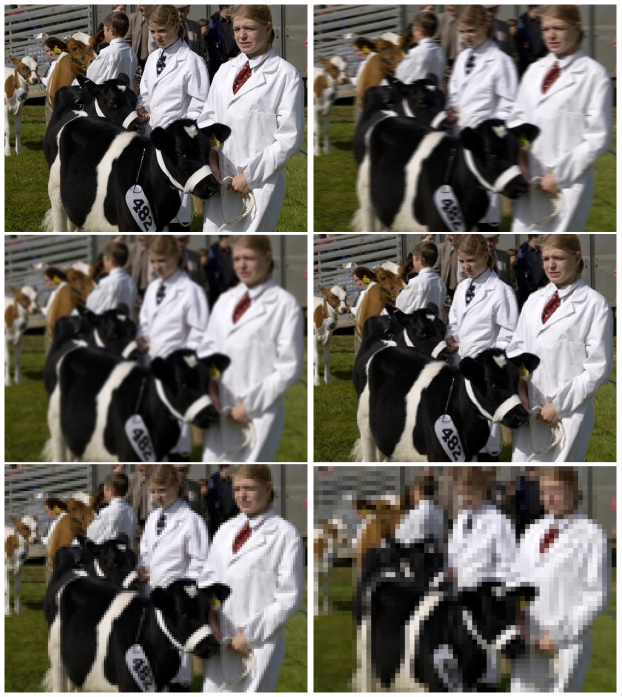
Figure 4. From left to right, top to bottom: original, motion-blur, Gaussian-blur, 2×2 pixelation, 4×4 pixelation, and 8×8 pixelation.

- Video3: middle group (less than 18, densely distributed) of people in indoor environment, with dark background. Mostly slow motion and large size of objects. The video has poor resolution.

- Video4: large group (more than 20, densely distributed) of people with similar dressing in outdoor environment, with light color background and severe occlusion. Mostly slow motion and large size of objects.

- Video5: middle group of cars (10-20, densely distributed) in light background with mild occlusion, in 45deg (from horizontal) perspective. Small group of people occasionally occur in the video.

- Video6: small group of cars (less than 10, sparsely distributed) in light background with almost no occlusion, in 30deg perspective.

- Video7: large group of cars (more than 20, sparsely distributed) in dark background with almost no occlusion, in 60deg perspective. Video quality is poor.

- Video8: small group of cars in dark background and almost no occlusion, in 80deg perspective. A few people occasionally occur. Video quality is good.

Number of cars and persons in each frame were manually labeled with time stamps in 8 text files, which were then loaded into buffer for performance evaluation. Object

Figure 5. Snapshots of video samples for the benchmark.

counting with different models were ran through each video with predictions compared to the labels. Frame stride of 20 were applied to speed up the evaluation. The performance was evaluated using the mean normalized error ϵ, which is expressed by the equation as below:

$$\epsilon = \frac{1}{8}\sum_{n=1}^{8}\sum_{i=1}^{j_n}\frac{|N_{pred} - N_{label}|}{(N_{label}+1)j_n} \quad (1)$$

where j_n is the frame number (taking frame stride into account) of Video n. N is number of counts.

4. Results

Intuitions are first acquired through YOLO re-training optimization, and then applied to faster R-CNN for better image and video detection.

4.1. Model Re-training Optimization

As described before, for YOLO each image quality reduction category is trained for 500 to 1,000 iterations based on the pre-trained model. Fig. 6 shows the loss history of 1,000 iterations of re-training motion-blurred images. Because the pre-trained model can already tolerate motion-blurred images to some extent, the loss does not start at a large value. The loss decreases and eventually becomes stable, implying that the re-trained model tolerates motion-blurred image better than the original model. Similar trend has been found in the re-training of other types of blurred or pixelated images.

The mean average precision (mAP) is calculated after the re-training of every image quality reduction category. As shown in Fig. 7, The original model cannot tolerate motion-blurred, Gaussian-blurred and pixelated images well. After re-training each category, the mAP of the corresponding category increases and the mAP's of other categories including the original clean images decrease. For most of the cases, the gain of the increasing robustness defeats the loss of detection accuracy in other categories. If there is

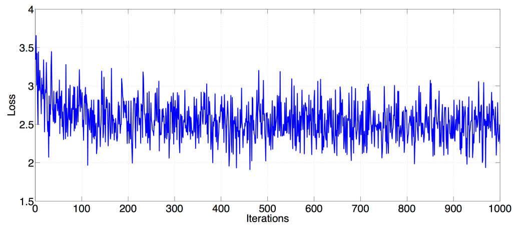

Figure 6. Loss history of 1,000 iterations on motion-blurred images.

little pixelation effect in test images and videos, then the re-training process should stop after re-training Gaussian blur, where there is a good balance among the performance of the original images, the motion-blurred images and the Gaussian-blurred images. However, if there are many small images in the test set which can result in great amount of pixelation after the resizing in the network, the final model should be used to improve pixelated image detection. After training for all 3 categories, the re-trained model is much less sensitive to image qualities.

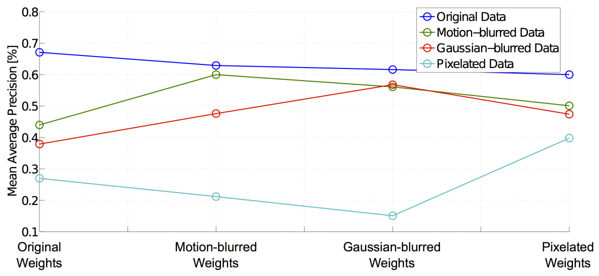

Figure 7. Mean average precision of YOLO after re-training different image quality reduction categories. The final model exhibits more robustness than the original model.

Similar re-training process is applied to different amount of pixelation. Based on the re-trained model after the motion-blur training, 4 additional groups of pixelated images are fed into the network to optimize the performance across all image categories. Instead of calculating mAP, we use an accuracy indicating correct class assignment and over 50% IOU as the metric. As shown in Fig. 8, after more severely pixelated images being fed into the network, the model generally tolerates pixelation better with the sacrifice of decreasing detection accuracy in the original, motion-blurred and Gaussian-blurred images. In YOLO the input images are resized into 448×448 pixels. If the input image is larger than 112×112 pixels, which corresponds to a worst case of 4×4 pixelation, then the re-training process should stop after the 4×4 pixelation training, because the 4×4 pixelated images have already gained enough accuracy boost and approaching a plateau while the accuracy of the original and blurred images have not lost much. However, if the input images can be even smaller, then further training with 6×6 or even 8×8 pixelated images are favored.

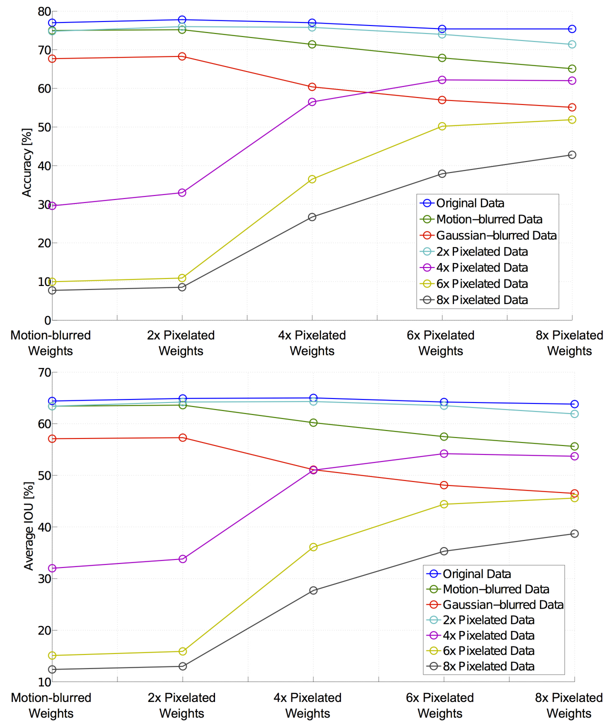

Figure 8. **Top:** Validation accuracy of different re-training stages. **Bottom:** Validation average IOU's of different training stages. As the model being trained with all pixelated images, it tolerate pixelation much better but with the sacrifice of accuracy loss in the original and blurred images.

The above intuitions of re-training optimization with different categories of blur and different amount of pixelation with YOLO can also be applied to the faster R-CNN model. Table. 1 shows the statistical results of the re-training of the faster R-CNN model. The mAP of people and cars are also listed in addition to the overall mAP as they are important for object counting in videos.

The pretrained model achieves a loss of about 0.8 when initialized. In cases of training with blurred images, the loss fluctuates and decreases down to about 0.4 after 100 iterations but is not necessarily significant smaller than the initial loss at the end of training. During testing, each image takes 0.565 sec to process. The results are shown in the table below. Performances of human and car detection among our main interest were evaluated individually. Mean AP was calculated over 20 classes.

	AP (Human)	AP (Car)	Mean AP
Retrain: None Test: Test: None	0.7062	0.8798	0.6852
Retrain: None Test: Motion Blur	0.5173	0.7257	0.4329
Retrain: Motion Blur Test: Motion Blur	0.6664	0.7384	0.6222
Retrain: Motion Blur Test: None	0.6643	0.8532	0.6428
Retrain: None Test: Gaussian	0.5627	0.7066	0.4723
Retrain: Gaussian Test: Gaussian	0.6235	0.7333	0.5876
Retrain: Gaussian Test: None	0.7160	0.8576	0.6935
Retrain: None Test: Pixelate	0.4950	0.5553	0.4404
Retrain: Pixelate Test: Pixelate	0.5878	0.8258	0.5511
Retrain: Pixelate Test: None	0.6613	0.8221	0.6527

Table 1. MAP performance on faster R-CNN using intuitions from YOLO re-training optimization.

As shown in Table. 1, both motion blur and Gaussian blur filters decrease the overall performance significantly by up to 0.2AP. The accuracy is more vulnerable to pixelate with a performance drop of almost 0.3AP. Retraining the network with processed images does improve the robustness of detection on low quality images. It is interesting that the performance of retrained model doesnt show noticeable drop when tested on original high quality images. In general, the car is easier to detect compared with human detection due to its comparatively invariant structure. The results also show that retraining the model with low quality images gains better performance improvement on human over on car detection for blurred images. However on pixelate images, car detection obtains better enhancement than human detection by retaining. Models trained on combinations of differently processed images will be evaluated in next section.

4.2. Performance of Re-trained Models

4.2.1 Still Images

Using the optimized models we can compare object detection performance in still images. As shown in Fig. 9, every image sequence denotes the input image, the neuron's output of the 1st, 9th, and 15th layer. The original model cannot tolerate low quality images well. As the image quality reduces, the neurons get less excited and thus less sensitive to objects. After the re-training and fine-tuning process, the model can detect objects well again. Even if the re-trained model cannot detect objects in the low quality images as well as the original model detecting objects in the clean images, the re-trained model still exhibits much better robustness than the original model.

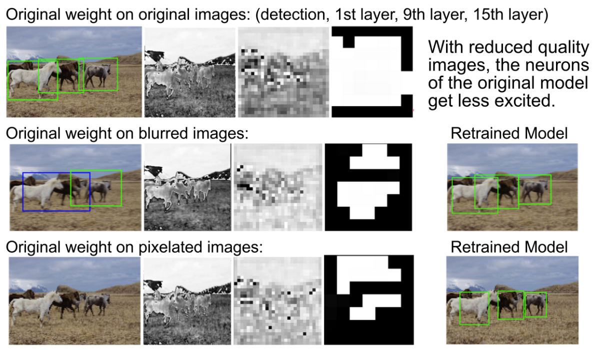
Original weight on original images: (detection, 1st layer, 9th layer, 15th layer)
With reduced quality images, the neurons of the original model get less excited.
Original weight on blurred images:
Retrained Model
Original weight on pixelated images:
Retrained Model

Figure 9. Visualization of neuron's outputs given different image qualities. The re-trained model can identify objects that the original model fails to recognize.

4.2.2 Videos

Object counting agents based on different models that were trained using aforementioned methods are evaluated with our benchmark. Results for YOLO-based agent are shown in the following table Table 2. After training all motion-blurred, Gaussian-blurred and pixelated images, the performance is slightly improved. For object counting, YOLO is very sensitive to prediction probability threshold. Low threshold correspond to more bounding box predictions, which can lead to better object counting performance. However, very low threshold can contribute to redundant predictions over the same object, and thus the performance increase can be fake. Moreover, YOLO is generally bad at differentiating among a group of closed objects such as a group of parked cars and a group of closely standing people. This also contribute to the non-obvious performance increase.

Video Index	YOLO Original	YOLO Motion	YOLO Gaussian	YOLO Pixelation
1	**0.711**	0.759	0.766	0.714
2	0.669	0.682	0.674	**0.650**
3	0.690	0.686	0.666	**0.637**
4	0.420	0.425	0.452	**0.411**
5	1.390	**1.374**	1.400	1.408
6	**0.796**	0.982	0.988	0.929
7	0.955	0.955	0.955	**0.954**
8	0.939	**0.909**	0.937	0.912

Video Index	R-CNN Original	R-CNN Motion	R-CNN Gaussian	R-CNN Pixelation
1	**0.508**	0.562	0.546	0.540
2	0.558	**0.541**	0.576	0.587
3	0.578	0.554	**0.535**	0.559
4	0.623	0.566	**0.534**	0.577
5	1.324	1.000	**0.903**	0.941
6	0.358	0.246	0.377	**0.241**
7	0.847	0.792	**0.713**	0.731
8	0.949	0.892	0.807	**0.753**

Table 2. Error comparison of the original models and the re-trained models. All models are re-trained based on the model trained before (Original – Motion – Gaussian – Pixelation).

Compared with YOLO, faster RCNN-based counting agent achieves better overall performance and less flickering of bounding boxes on successive frames. Normalized Errors of agents based on differently-trained faster-RCNN are compared in Fig.10. Faster RCNN trained by only one type of images achieves different degrees of improvement, among which training with Gaussian-filtered images get the best performance. Training the agent with VOC2012 (larger set of images) reduces the performance. But if training it with motion blurred, Gaussian blurred and pixelate images one by one successively, the agent gains continuous improvement. Details of normalized errors on different video samples are listed in Table. 123.

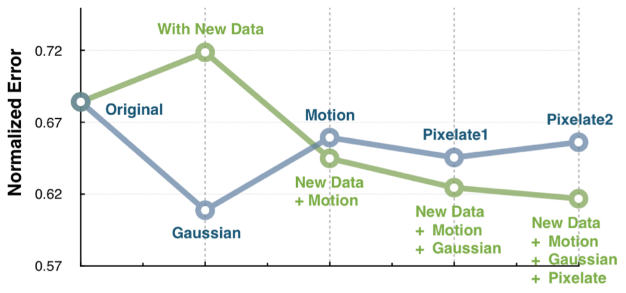

Figure 10. Performance of agents trained with different methods. Green line is about agents separately trained by different type of low quality images. Blue line is about agent trained successively by different type of images and expanded training data set.

Our final counting agent is based on faster R-CNN that has been retrained on all methods successively (last dot of green line in Fig. 10). Several snapshots are provided to show its performance, as shown in Fig. 11.The agent is able to detect fast and blur motion (Picture 1 and 2), very small object (Picture 4 and 5), occlusion (Picture 5) and object counting. It fails when the occlusion become severe (Picture 7 and 9), low quality image and objects with similar background (Picture 8), and lack of training data (Picture 3, 8 and 9 where some cars are shot from the top and the wheels become invisible).

5. Conclusion and Future Work

In conclusion, we present object detection and counting in low quality images and videos using two cutting-edge deep learning models, YOLO and faster R-CNN. The models are re-trained and fine-tuned with low quality images to increase robustness on motion-blurred, Gaussian-blurred and pixelated images. The re-trained model demonstrate improved performance of object counting agent on a wide range of videos.

Next step is to further improve the model by adding a counting layer for end-to-end training and testing. More photos about cars and humans taken from different perspectives are required to achieve more robust performance. To deal with densely grouped objects with severe occlusion, methods such as texture analysis may be incorporated to the model.

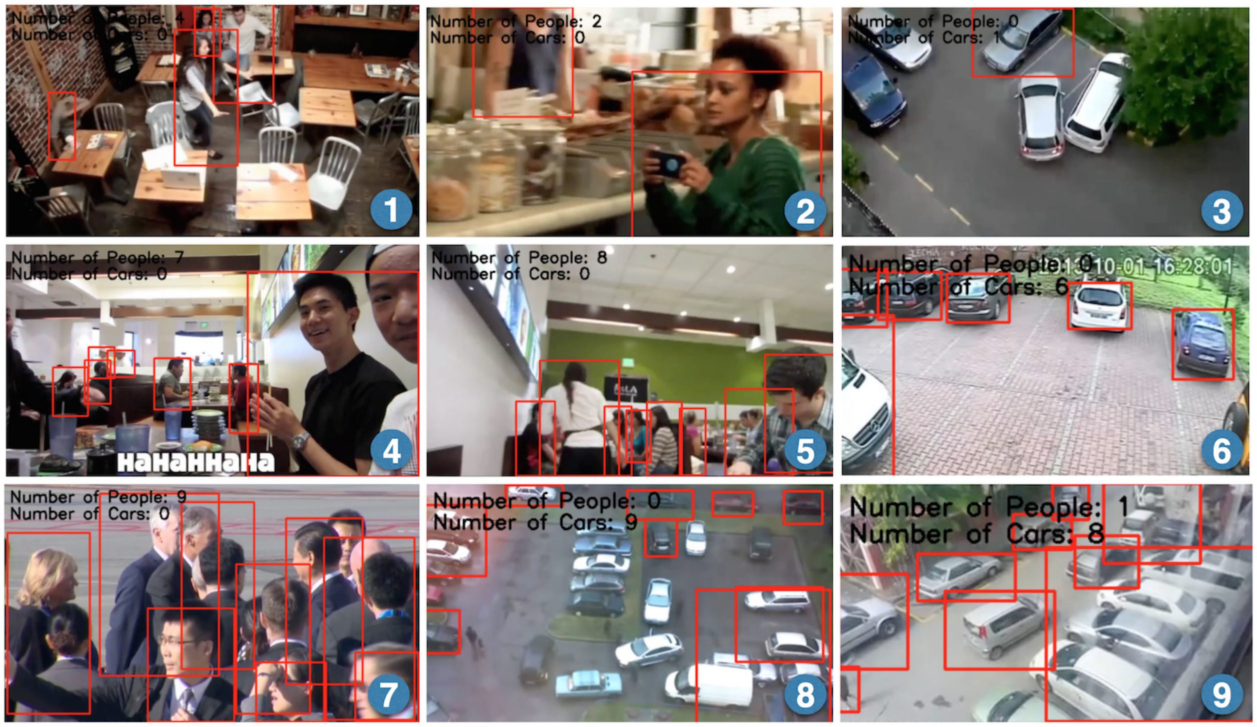

Figure 11. Snapshots of object counting on videos with agent based on faster-RCNN trained by different type of images successively.

References

[1] Caffe. `https://github.com/rbgirshick/caffe-fast-rcnn`.

[2] Canon. `http://www.canon.com/technology/canon_tech/explanation/dc.html`.

[3] Darknet. `http://pjreddie.com/darknet/yolo/`.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[5] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2154, 2014.

[6] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4557–4564. IEEE, 2012.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[8] S. Grzonka, G. Grisetti, and W. Burgard. A fully autonomous indoor quadrotor. *Robotics, IEEE Transactions on*, 28(1):90–100, 2012.

[9] P. Hsu and B.-Y. Chen. Blurred image detection and classification. In *Advances in multimedia modeling*, pages 277–286. Springer, 2008.

[10] R. Liu, Z. Li, and J. Jia. Image partial blur detection and classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2015.

[12] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[14] TerminalCloud. `https://www.stanfordterminalcloud.com/`.