Applied Machine Learning for Image Based Cattle Recognition and Counting

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Sangeeta Kumari

Graduate Program in Computer Science and Engineering

The Ohio State University

2017

Abstract

Object detection and localization in time series images is a challenging task because of uncontrolled variations in background, orientation and scaling of targets from one frame to another. The problem becomes worse with non-uniform interval in the series data. With the partial presence of target objects in some of the images no longer being a rare scenario, detection of all the target instances becomes extremely difficult because of insufficient features for classification. Although region proposal algorithms like Faster Region-based Convolutional Neural Network (R-CNN) detects targets with high precision, it gives significant number of false negatives when the threshold is set for high confidence of target. Since such models are not designed for temporality, they fail to recognize same targets detected in the previous frames. We propose a method that adds short-term memory to the model without having to train a complex network like Recurrent Neural Network. We test our approach on the image dataset we collected at a cattle farm at The Ohio State University and it shows an improvement in recall by 71.3% over the detections made by Faster R-CNN in the absence of memory.

Table of Contents

List of Figures

1 Introduction

1.1 Motivation

Data-driven classifiers have found use in many applications like image categorization [15], image matching [14], activity recognition [16], vegetation classification [13], video tagging [17], anomaly detections [18], etc. For any data-driven approach, it is not enough to collect large amount of data; the data is to be prepared in a way that models can use them to find patterns and inferences. This is done by adding relevant metadata to the collected dataset. With the huge amount of data present, manual annotation of the data with ground truth becomes a very tedious task.

In this study, we focus on automated ground truth estimation in the context of data-driven approach for human-animal discrimination application that realizes mote-scale (wireless sensor nodes) classifiers using micropower Doppler sensors and an attached microcontroller. To collect data and ground truth for Doppler radar training, we have a setup of camera traps and motes at a cattle farm at The Ohio State University. Camera and radar work independently while collecting data, lack the time synchronization and do not have same range of view, hence the results from both cannot be directly correlated. We take a step towards solving this problem by using machine learning for labelling all targets (in this instance, cow) in the images.

1.2 Related Work

A common approach for object detection has been an exhaustive search of the full image with sliding window technique using windows of different scales. Dalal et al. [7] use this approach with Support Vector Machines trained on Histogram of Oriented Gradients (HOG) features for pedestrian detection. Selective search [4], on the other hand, reduces the number of windows we need to consider by doing region segmentation first and giving only the regions with potential objects in it. Such kinds of region proposals have found use in many extant object detection models [1][2][3][5] [10].

R-CNN [3] uses ~2k region proposals using Selective Search and then classifies them with a pretrained convolutional neural network (CNN). This approach combined three important aspects of object detection, category-independent proposals, reduction in search space and rich CNN features. While the replacement of the hand-engineered features like HOG [7] or SIFT [12] with high level object representations obtained from CNN models has brought a significant improvement in the accuracy (30% relative improvement over the best previous result on PASCAL VOC 2012 [11] with VGG16 [9] model), it relies on Selective Search to provide the region proposals and feeds each one of them to the Convolutional Network (ConvNet) which makes the training as well as testing very slow.

Fast R-CNN [2] significantly improves the efficiency and accuracy of R-CNN by pooling and reusing the convolutional layers which allows for faster feature extraction on the proposal windows. Since it still relies on external region proposals, it is still a time-consuming step. Introduction of Region Proposal Network (RPN) in Faster R-CNN [1] gets rid of the external proposal method by making use of a fully convolutional network to output a set of rectangular object proposals, each with an objectness score (probability of an object present in the region). RPN with Fast R-CNN produces detection accuracy better than the baseline of Selective Search with Fast R-CNN. However, with high (Intersection over Union) IoU thresholds (> 0.7), the recall of RPN drops and since the model lacks temporality, it fails to exploit our time-series based image dataset. Although this can be solved to some extent by lowering the threshold, this introduces a new type of false negatives, i.e., cows get tagged as horses, sheep, dogs, etc.

1.3 Approach and Outline

To exploit the temporality in our dataset, we propose an algorithm that generates a history of detections made by Faster R-CNN and makes use of color histograms for template matching to find match between past and present detections. If a recent detection gets missed in the present frame with the target in a nearby region, we build upon the memory of the recent detection result.

Our validation of the algorithm is comparative, with respect to a Linear Support Vector Machine (LSVM) and traditional Faster R-CNN. We show that Faster R-CNN gives a much higher precision than LSVM trained with hand-crafted HOG features. We then compare the traditional Faster R-CNN with our modified approach and show an improvement in recall by 71.3%.

The rest of the report is structured as follows. Section 2 describes LSVM, a procedure for its training with HOG features for object detection, and shows its performance in the context of our application. Section 3 starts by a brief description of R-CNN and is followed by its successor models like Fast and Faster R-CNN that improved the performance of traditional R-CNN. Section 3.3 discusses the training steps and performance of Faster R-CNN with pretrained VGG16 CNN model. Our approach of adding temporality post VGG16 detections is introduced in section 4 with the pseudo code in Algorithm 4. The performance and potential issues are discussed in Sections 4.3 and 4.4 respectively. We end by summarizing the results and suggesting future work in Section 5.

2 Linear Support Vector Machine

In this section, we describe the LSVM, its application in object detection and a training procedure using HOG features.

Given a training dataset of n points which are in the form (x_i, y_i) where x_i is the d dimensional feature vector and y_i is the class label to which x_i belongs (+1 and -1 for two class classification), a linear SVM tries to find a hyperplane that maximizes the margin between both the classes. It gives a set of weights w for each feature so that their linear combination predicts the value of y.

It defines the hyperplanes H such that:

$\mathbf{w \bullet x_i + b \geq +1}$ **when** $\mathbf{y_i = +1}$

$\mathbf{w \bullet x_i + b \leq -1}$ **when** $\mathbf{y_i = -1}$

where:

b/||w|| determines the offset of the hyperplane from the origin along the normal vector w.

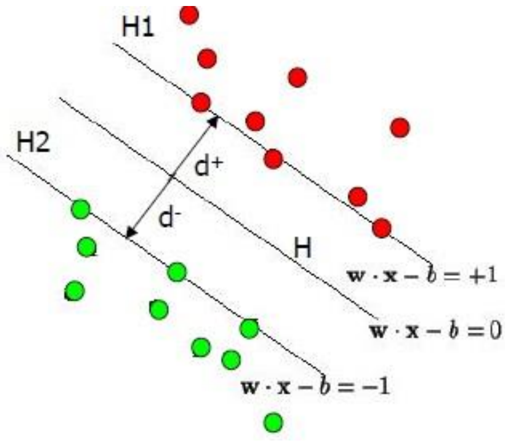
H1
H2
d+
d-
H
$\mathbf{w} \cdot \mathbf{x} - b = +1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$

Figure 1: Optimum Separation Hyperplane

To maximize the margin, we thus need to minimize ||w|| so that there are no data points between H1 and H2. For classification by Linear SVM, training involves the minimization of the error function:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i$$

subject to the constraints: $y_i\left(w^T\phi(x_i)+b\right)\geq 1-\xi_i$ and $\xi_i \geq 0, i=1,\ldots,N$

where C is the capacity constant, w is the vector of coefficients, b is a constant, and ξ_i represents parameters for handling non-separable data (inputs).

2.1 Histogram of Oriented Gradients

A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. Histogram of Oriented Gradients (HOG) is a feature descriptor that exploits the gradient information in localized parts of the image to give useful information about the object characteristic.

The image patch is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells, each having magnitude and angle. The magnitude of each angle is accumulated in the 9-bin histogram corresponding to the angles i.e. 0, 20, 40, 60 … 160. Since gradients of an image are sensitive to overall lighting, we L2 normalize the histogram with so they are not affected by lighting variations to get a 36 × 1 vector from 16×16 block. All the block features are then concatenated to form the end feature on which the SVM will be trained. The pseudo-code for computing HOG is described in Algorithm 1.

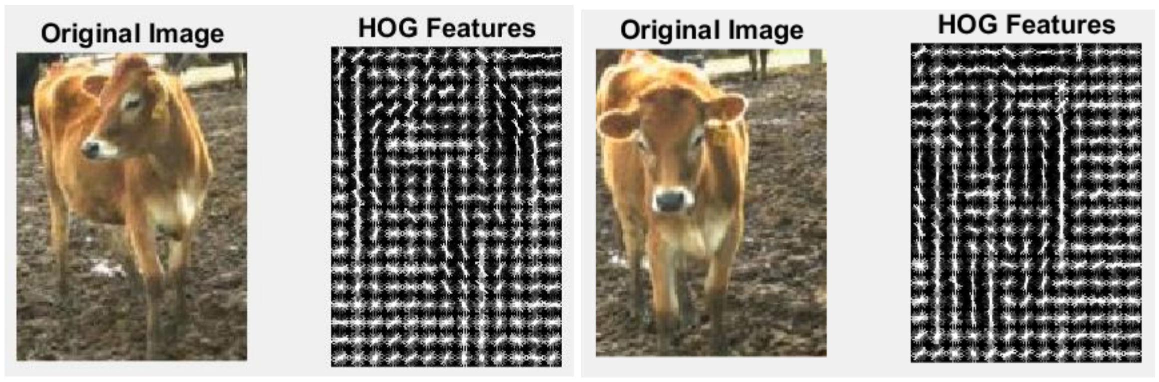

Figure 2: Examples of HOG features

Algorithm 1: Computing Histogram of Gradients on an image patch

```
img_patch ← 120 ×160 cropped image patch
feature ← []
for each 16×16 block in img_patch
    block_feature ← []
    for each 8×8  cell in block
        gx ← cell convolved with [-1 0 1] filter
        gy ← cell convolved with [-1 0 1]' filter
        hist ← []
        for each pixel p in cell
            g(p) ← sqrt(gx(p)^2 + gy(p)^2)
            angle(p) ← arctan (gy(p)/gx(p))
            hist(bin_index)← hist(bin_index) + g(p)

    block_feature ← append hist to block_feature
    normalize the block feature

    feature ← append block_feature to feature

normalize the feature vector using L2-norm
```

2.2 Training Linear SVM

2.2.1 Dataset Creation

Being a supervised learning algorithm, we need to have labelled data for training an SVM. Image patches of 120 ×160 dimension were cropped from our collected farm data set and from images downloaded from internet to form the positive and negative dataset. The dataset was increased by generating mirror instances and changing the orientation. A python script is written that takes backgrounds with no targets and randomly crops 120 ×160 image patches for negative training dataset.

2.2.2 Parameter Selection

In machine learning, two tasks are commonly done at the same time in data pipelines: cross validation and (hyper)parameter tuning. Cross validation is the process of training learners using one set of data and testing it using a different set. Parameter tuning is the process to selecting the values for a model's parameters that maximize the accuracy of the model.

Grid search is used for performing hyperparameter optimization and it refers to an exhaustive search though a pre-specified list of parameter values needed for the SVM learning algorithm. The search algorithm is guided by its accuracy which is measured by cross-validation on the training set or evaluation on a held-out validation set.

For example, a Linear SVM regression has a regularization constant C and width p that needs to be tuned for good performance on unseen data. Both parameters are continuous, so to perform grid search, one selects a finite set of valid values for each, for example:

$C \leftarrow \{0.1, 500, 5\}$

$p \leftarrow \{0.01, 7, 100\}$

where 1st value is min value, 2nd is the max value and 3rd is the logarithmic step for iterating. Grid search then trains an SVM with each pair (C, p) in the Cartesian product of these two sets and evaluates their performance by internal cross-validation on the training set. The algorithm outputs the settings that achieves the highest score in the validation procedure.

2.2.3 Hard Negative Mining

After the 1st iteration of training, the model had many false positives so hard negative mining was done by explicitly creating a negative example out of the falsely detected patch, and adding that negative to the training set. A similar procedure was repeated 3-4 times and the classifier was retrained for better performance.

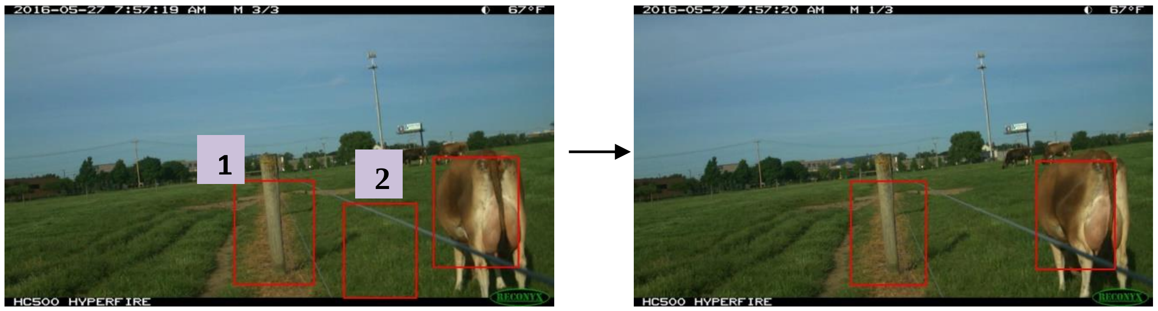

Figure 3: (Left) Image with false positives (Right) Image after hard negative training with patch 2

2.3 Performance

The model detection had false negatives and false positives (even after hard negative training) as shown in Figure 4. Since our dataset has lot of variations, like both sitting and standing cows, partial bodies etc., we could not train just one generic SVM for our target.

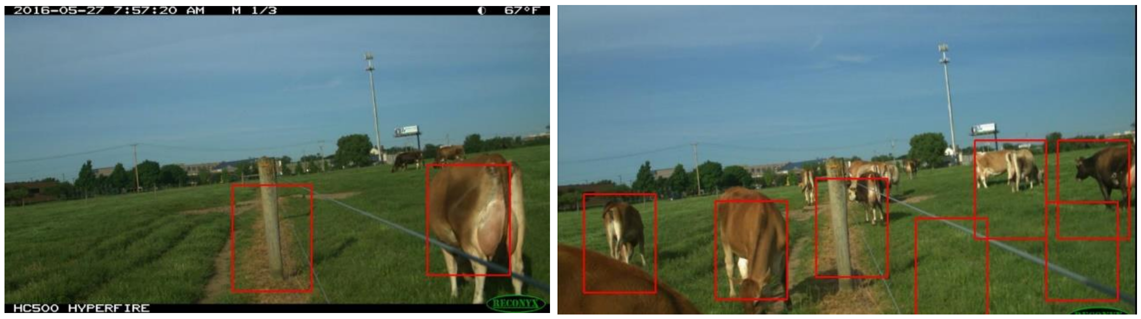

Figure 4: Detections by Linear SVM

3 R-CNN

As shown in the previous approach with Linear SVM, the shallow features provided by the histogram of oriented gradients is not enough to deal with the variations present in our dataset. Moreover, the sliding window search with different scaling does not give good localization. In this section, we use describe models that use CNN features and replaces the sliding window approach with region proposal algorithms. One such model is given by Ross Girshick et al., R-CNN [3]. As described in their paper, the model consists of three modules. The first module uses selective search [4] to generate category independent region proposals. The second is a large convolutional neural network that extracts a fixed-length feature vector from each region. The third module is a set of class specific linear SVMs.

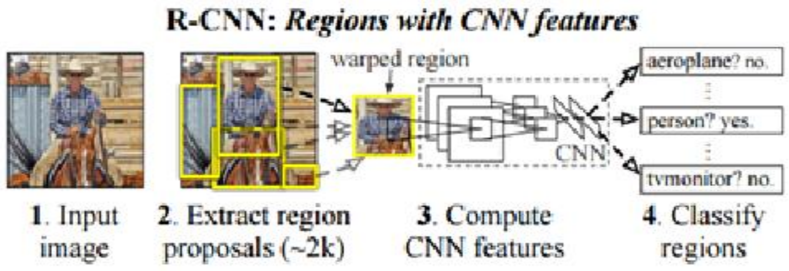

Figure 5: R-CNN architecture as shown in [3]

At test time, selective search is run on the test image to extract around 2000 region proposals. Each proposal gets warped and forward propagated through the CNN to compute features. Then, for each class, each extracted feature vector is scored using the SVM trained for that class. Given all scored regions in an image, a greedy non-maximum suppression (for each class independently) is applied that rejects a region if it has an intersection-over union (IoU) overlap with a higher scoring selected region larger than a learned threshold.

3.1 Slow R-CNN

The traditional R-CNN method is very slow because it performs a ConvNet forward pass for each object proposal, without sharing computation. Since the features are written to the disk, it also takes a lot of disk space.

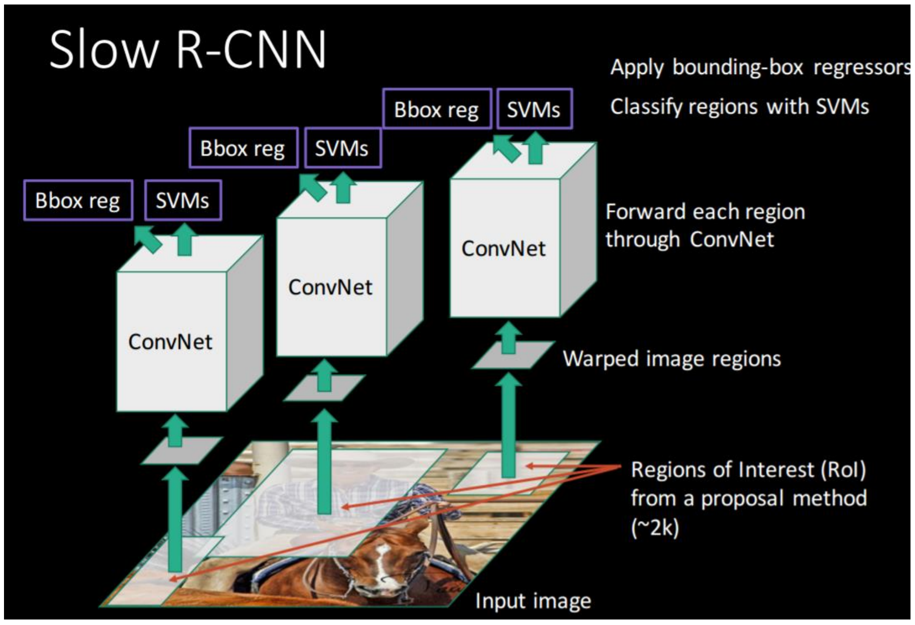
Slow R-CNN
Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Figure 6: Training of Slow R-CNN as shown in [3]

3.2 Fast R-CNN

Fast R-CNN [2] by Ross Girshick reduced the training to a single-state process by using multi-task loss. It also got rid of the disk storage disadvantage because features are not cached in this model.

As described in [2], Fast R-CNN network takes as input an entire image and a set of object proposals. It processes the whole image with several convolutional and max pooling layers to produce a convolutional feature map. Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all "background" class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

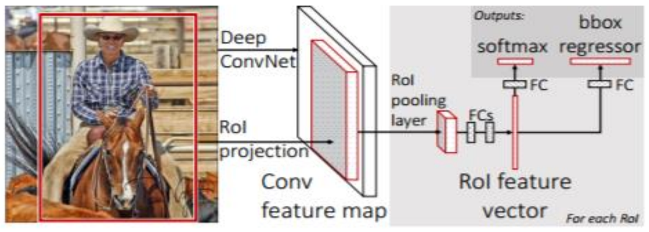

Figure 7: Fast R-CNN architecture as shown in [2]

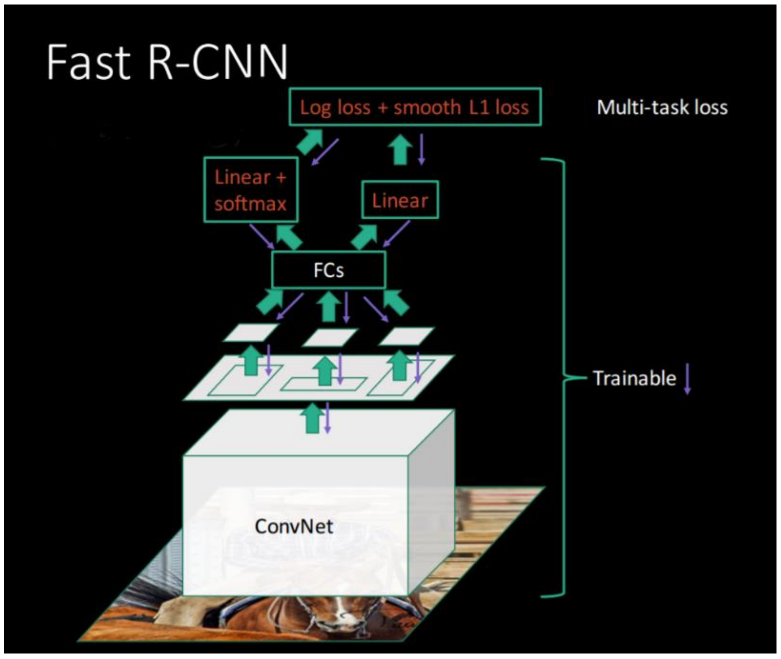

Figure 8: Training of Fast R-CNN as shown in [2]

3.3 Faster R-CNN

Till Fast R-CNN, the region proposals were external. Shaoqing Ren et al. found that the convolutional feature maps used by Fast R-CNN can also be used for generating region proposals. They introduced Region Proposal Networks (RPNs) that shares convolutional layers with the object detection networks [2], [5] that use region proposals. This reduced the cost for computing region proposals. RPNs are designed to efficiently predict region proposals with a wide range of scales and aspect ratios.

Thus, Faster R-CNN consists of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector.

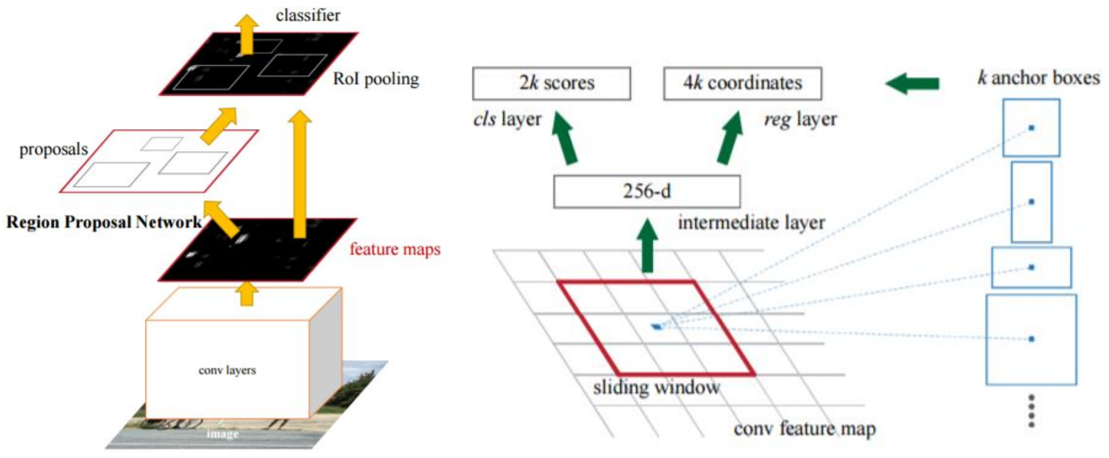

Figure 9: (Left) Faster R-CNN (Right) Region Proposal Network (RPN) as shown in [1]

We use Faster R-CNN written in Python [6] with the pre-trained Caffe model (VGG16) with the PASCAL VOC 2007 categories which has ‘cow’ as one of the classes.

3.3.1 Training

Algorithm 2: Training Faster R-CNN

Step 1: Finetune Region Proposal Network (RPN) from pre-trained ImageNet network

Step 2: Finetune fast RCNN from pre-trained ImageNet network using bounding boxes from step 1

Step 3: Keeping common convolutional layer parameters fixed from step 2, finetune RPN (post conv5 layers)

Step 4: Keeping common convolution layer parameters fixes from step 2, fine-tune fast RCNN fc layers

3.3.2 Performance

With the score threshold at 0.7 (a value we selected for empirical testing), the model did not give any false positives (as shown in Figure 10) thereby giving a high precision but with significant false negatives: missed instances as well as cows getting classified as other categories are illustrated in Figure 11.

The first problem occurred more than the second one. The targets with a score close to 0.7 showed a high probability of being detected in some frames and getting missed in the subsequent ones even with slight variation in orientation.

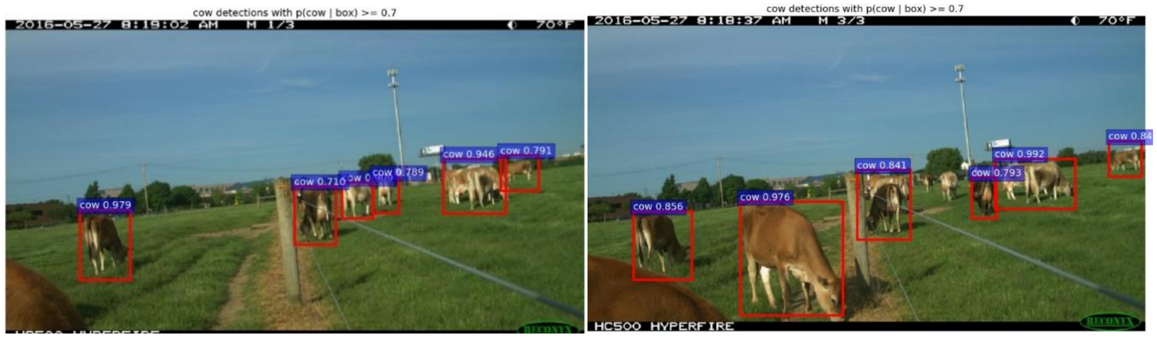
cow detections with p(cow | box) >= 0.7
2016-05-27 8:19:02 AM M 1/3 70°F
cow 0.979
cow 0.710
cow 0.789
cow 0.946
cow 0.791
cow detections with p(cow | box) >= 0.7
2016-05-27 8:18:37 AM M 3/3 70°F
cow 0.856
cow 0.976
cow 0.841
cow 0.793
cow 0.992
cow 0.84
HC500 HYPERFIRE

Figure 10: Correct detections by Faster R-CNN (with VGG16)

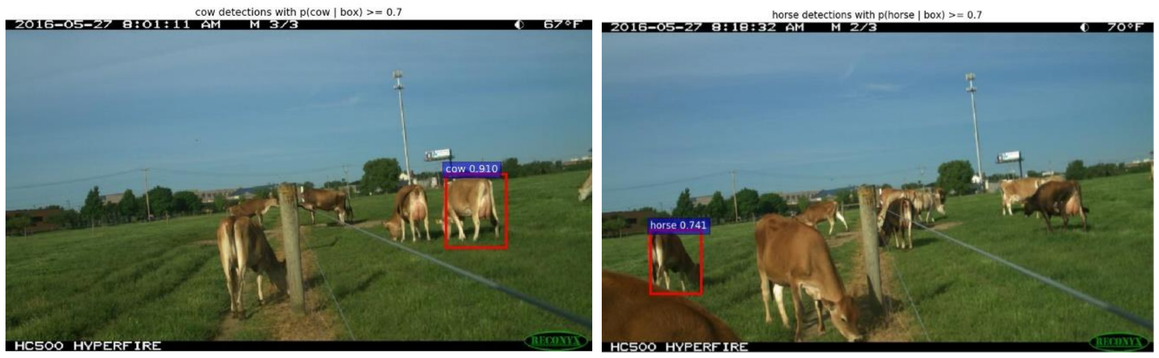
cow detections with p(cow | box) >= 0.7
2016-05-27 8:01:11 AM M 3/3 67°F
cow 0.910
HC500 HYPERFIRE
RECONYX
horse detections with p(horse | box) >= 0.7
2016-05-27 8:18:32 AM M 2/3 70°F
horse 0.741
HC500 HYPERFIRE
RECONYX

Figure 11: False negatives by Faster R-CNN (with VGG16)

4 Faster RCNN with Short-Term Memory

As mentioned in Section 3.3.2, Faster R-CNN with VGG16 had significant false negatives. There are two possible solutions to the problem of same targets getting missed intermittently. Either we decrease the threshold or we somehow create history of the detections so that if the same instance gets missed in subsequent frames, we can propagate the information we have from the temporal information we have. The first solution increased lot of false negatives in form of incorrect classifications so we chose the second approach.

4.1 Data Structures

To create a temporal history for the detections, we need to map all the detections with their corresponding bounding boxes, color histograms of the patch (inside the bounding box) and weights. In order to optimize the lookup time, we chose dictionaries to meet our above requirement. We used 4 dictionaries:

1) Image file mapped to a set of indicators (indicating each detection window in the image and the ones propagated from history)

2) Indicators mapped to upper left co-ordinates, width and height

3) Indicators mapped to histograms corresponding to its window

4) Indicators mapped with its weight

Here, indicators are just global counters for detection windows. Algorithm 3 describes the changes made in each dictionary after an image is processed for target(s) detection by Caffe (VGG16 model). For example, let us say that Caffe detected 3 targets in ‘Image1.jpg’ with indicators 1, 2, 3, the data structures get updated as mentioned in algorithm 3 below.

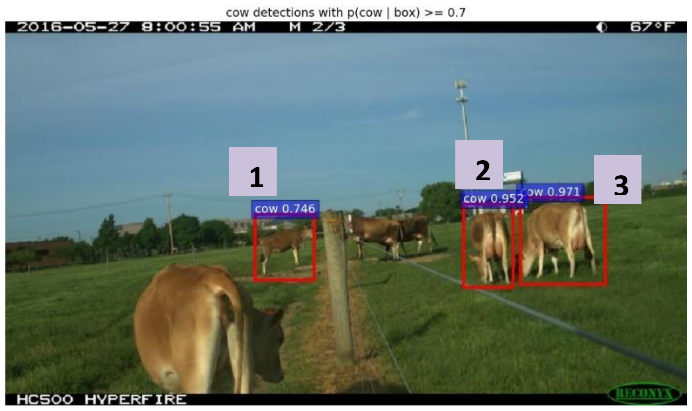
cow detections with p(cow | box) >= 0.7
2016-05-27 8:00:55 AM M 2/3
67°F
1
2
3
cow 0.746
cow 0.952
ow 0.971
HC500 HYPERFIRE
RECONYX

Figure 12: VGG16 detection on 1st frame

Algorithm 3: Temporal Data Structure

```
imgDict ← (Image1.jpg, [1, 2, 3])
**for all** indicators:
    coordinateDict ← (indicator, [x, y, wt, ht])
    histDict ← (indicator, color_histogram(image_patch(indicator)))
    wtDict ← (indicator, 1)

where (1, 2) shows key value pair added in dictionary
```

4.2 Algorithm

The algorithm described below is based on a few domain specific assumptions:

1) Caffe does not give false positives with the threshold at 0.7

2) Since the frame sequence has interval of 5 seconds, there is insignificant displacement in the target location in between two frames
3) The target (cow) and the environment (a farm background) do not share color information

With each frame, say at time *t*, we save all the VGG16 detections of the 'cow' class and map them as indicators with the bounding box co-ordinates and color histograms of the patch as described in Section 4.1. We compare the current detections against the list of indicators from the frame at time *t-1* and if there is any past detection which is not present in present frame, we compare the patch from the previous detection with patches around a surrounding area in the present frame to check the presence of cow. If an instance is found in the current frame, we add this instance also in the list of detected instances.

This method might introduce false positives if the color histograms match for non-target patches and it might propagate the incorrect prediction indefinitely through all subsequent frames so we maintain a weight dictionary in which we map each detection with a weight (initialized with 1). We progressive decrease the weights with time, thereby giving less importance to older detections and eventually forgetting them after a bounded number of frames. The pseudo code is described in Algorithm 4, below.

Algorithm 4: Update Temporal History

```
indicator_list ← caffe_detect(image)
add image in imgDict

if indicator_list is empty
    temp ← all previous indicators
else
    for all ind in indicator_list
        update the dictionaries as in Algorithm 3
        for key in coordinateDict.keys():
            [x2, y2, wt2, ht2] ← coordinateDict[key]
            **if** past detection p does not correspond to any new detections
                temp[key] ← coordinateDict[key]

**if** temp is not empty
   **for** k in temp.keys()
        [x1, y1, wt1, ht1] ← temp[k]
        half_ht ← ht1/2
        half_wt ← wt/2
        midx ← x1 + half_wt
        midy ← y1 + half_ht
        match_list ← {}
        p_list ← {}
        j ← 0
        hist_prev ← h_dict[key]

        **for** all y in (midy-half_wt*0.25) and (midy+half_wt*0.25)
            **for** x in (midx-half_ht*0.25) and (midx+half_ht*0.25)
                im2 ← im[y-half_wt:y+half_wt,x-half_ht:x+half_ht,:]
                hist_curr ← color_histogram(im2)
                match_list[j] ← compare_hist(hist_prev, hist_curr)
                p_list[j] ← [x-half_ht, y-half_wt, wt1, ht1]
                 j ← j +1

                [min_key, min_value] ← min value in match_list.items()

                **if** min_value < limit and wt[k] > max_prob
                       [x, y, w, h] ← p_list[min_key]
                       img2 ← im[y:y+h, x:x+w]

                       **if** imgName is in imgDict.keys()
                              imgDict[imgName].append(k)
                       **else**
                              imgDict[imgName] ← [k]

                       histDict ← color_histogram(img2)
                       coordinateDict[k] ← [x, y, w, h]
                       wtDict[k] ← wtDict[k] - (α* wtDict[k])
                       delete the previous key for this indicator

to_remove ← imgDict[prev_frame]
**for** value in to_remove
    **if** value in coordinateDict.keys()
        delete coordinateDict[value]
    if value in histDict.keys()
        delete histDict[value]
    if value in wtDict.keys()
        delete wtDict[value]
    delete imgDict[prev_frame]
```

4.3 Performance

Our evaluation of the model with the short-term memory shows that no target detected in past is missed, thereby improving recall from 38.8% to 66.5%. Table 1 below compares the count of true positives between Faster R-CNN and Faster R-CNN with temporality.

	Ground truth	**Faster R-CNN**	**Faster R-CNN with temporality**
Number of instances	314 (manually annotated)	122	209
Recall		38.8%	66.5%

Table 1: Comparison of results by Faster R-CNN and Faster R-CNN with temporality

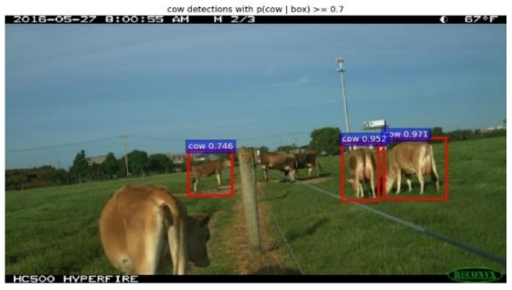
cow detections with p(cow | box) >= 0.7
2016-05-27 8:00:55 AM M 2/3 67°F
cow 0.746
cow 0.952
cow 0.971
HC500 HYPERFIRE

Image1

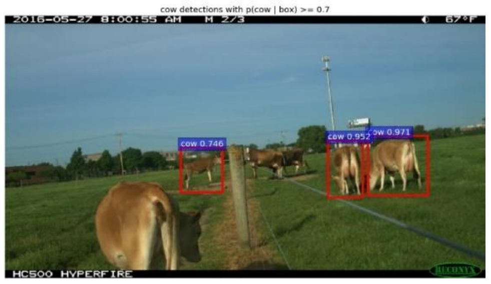
cow detections with p(cow | box) >= 0.7
2016-05-27 8:00:55 AM M 2/3 67°F
cow 0.746
cow 0.952
cow 0.971
HC500 HYPERFIRE

Image1

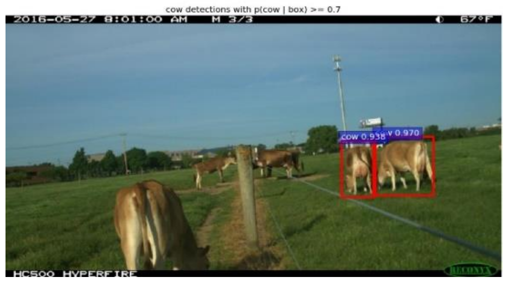
cow detections with p(cow | box) >= 0.7
2016-05-27 8:01:00 AM M 3/3 67°F
cow 0.938
cow 0.970
HC500 HYPERFIRE

Image2

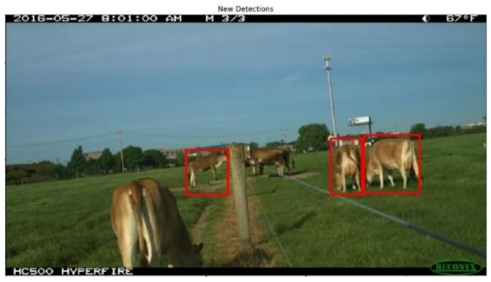

Image2

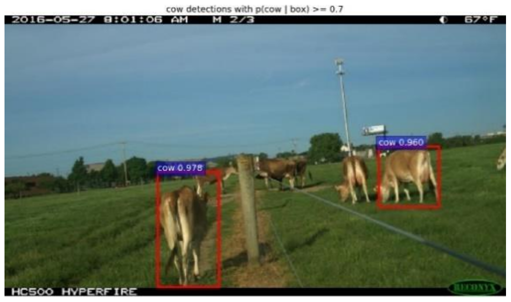
cow detections with p(cow | box) >= 0.7
2016-05-27 8:01:06 AM M 2/3 67°F
cow 0.978
cow 0.960
HC500 HYPERFIRE

Image3

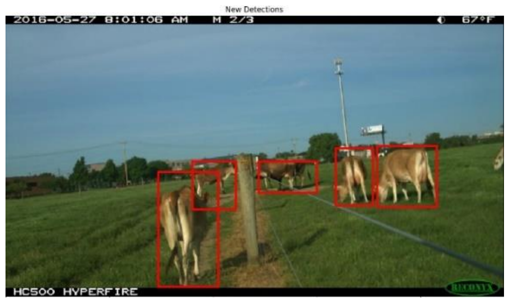

Image3

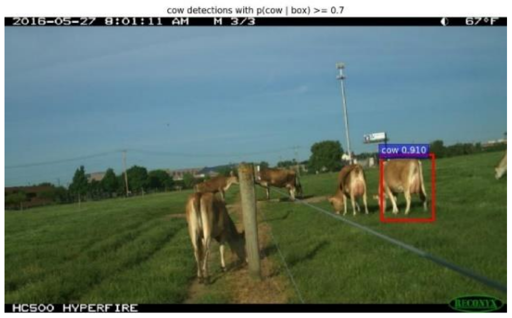
cow detections with p(cow | box) >= 0.7
2016-05-27 8:01:11 AM M 3/3 67°F
cow 0.910
HC500 HYPERFIRE

Image4

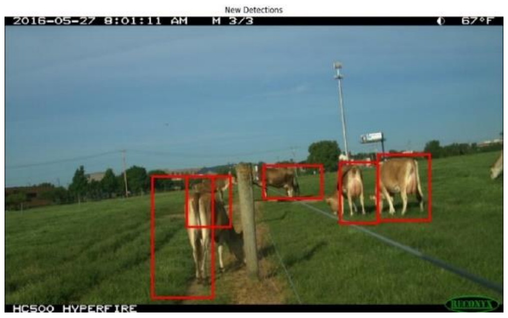

Image4

Figure 13: (Left) Detections from VGG16 (Right) Detections after short-term memory added

4.4 Potential Issues

If the model introduces false positive, then with the current setting of α (0.1), the detection gets propagated till 3 frames before model decides to forget it. Since we search for the potential missing target in multiple patches, the localization of the propagated detections is not always precise. Images in which the targets are not sparsely located give an incorrect count of the targets because multiple instances get captured within a common bounding box.

Moreover, if the target and the environment share color information, color histograms would not be a good feature descriptor for comparing the similarity.

5 Conclusion and Future Work

We compare the detections from Faster R-CNN with that of Linear SVM and find that Faster R-CNN gives a higher accuracy and improves localization. But at the same time, it gives a significant number of false negatives so we propose an algorithm to combine temporal information with the model predictions and show that it improves recall by 71.3%.

Re-training of the VGG16 model might have improved the detections. Also, the Caffe implementation of Faster R-CNN uses the PASCAL VOC 2007 categories so the model is not quite generic. As mentioned above, we can fine-tune VGG16 with our own training set to incorporate all the categories of interest. Although our problem allows us to make the assumptions as stated, color histograms may not work for all scenarios. If better descriptors are used, it will further increase the model's robustness.

6 References

[1] Shaoqing Ren, Kaiming He, R Girshick and Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", in IEEE International Conference on Computer Vision (ICCV), 2016

[2] R. Girshick, "Fast R-CNN," in IEEE International Conference on Computer Vision (ICCV), 2015

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014

[4] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders, “Selective search for object recognition,” International Journal of Computer Vision (IJCV), 2013

[5] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in European Conference on Computer Vision (ECCV), 2014

[6] https://github.com/rbgirshick/py-faster-rcnn

[7] N Dalal and B Triggs, “Histogram of oriented gradients for human detection”, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005

[8] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, “Caffe: Convolutional Architecture for Fast Feature Embedding”, in arXiv preprint arXiv:1408.5093, 2014

[9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, in ICLR, 2015

[10] S. Gidaris and N. Komodakis, “Object detection via a multiregion & semantic segmentation-aware CNNs model”, in ICCV, 2015

[11] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge: A retrospective”, in IJCV, pages 98–136, 2015

[12] D. G. Lowe, “Distinctive image features from scale-invariant keypoints”, in IJCV, 60(2):91–110, 2004

[13] Sah, Jay P., Ross, Michael S., and Stofella, Susana, "Developing a Data-Driven Classification of South Florida Plant Communities", in SERC Research Reports. 93, 2010

[14] Rodriguez-Serrano J.A., Sandhawalia H., Bala R., Perronnin F., Saunders C., “Data-Driven Vehicle Identification by Image Matching”, in Computer Vision – ECCV 2012 Workshops and Demonstrations, ECCV 2012

[15] Hu J., Sun Z., Li B., and Wang S., “PicMarker: Data-Driven Image Categorization Based on Iterative Clustering”, in Computer Vision – ACCV 2016

[16] Minor, B., Janardhan Rao Doppa, and Diane J. Cook, “Data-Driven Activity Prediction: Algorithms, Evaluation Methodology, and Applications”, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015

[17] Lamberto Ballan, Marco Bertini, Tiberio Uricchio, Alberto Del Bimbo, “Data-driven approaches for social image and video tagging”, Multimedia Tools and Applications, 2015, Volume 74, Number 4, Page 1443

[18] Eliahu Khalastchi, Meir Kalech, Gal A. Kaminka and Raz Lin, “Online data-driven anomaly detection in autonomous robots”, in Knowledge and Information Systems, 2014