

# COMP4801 Final Report

## Object Recognition with Videos

Student: Liu Bingbin (3035085748)

Supervisor: Dr. Kenneth Wong

April 16<sup>th</sup>, 2017

### Abstract

Object recognition is an interesting task in computer vision with a wide range of real world applications. While object recognition in still images has achieved impressive performance, object recognition in videos is yet to be explored. Due to motion blur and other complexities, still image methods directly applied to video frames usually cannot yield satisfying results, which calls for the need to better study and incorporate temporal information in videos. In this project, the baseline framework is chosen to be T-CNN [1], which leverages video's temporal and contextual information by post-processing techniques such as motion-guided propagation and tracking. However, temporal information is only incorporated after still-image detection results are obtained, and some of the post-processing processes are time consuming. This project therefore attempts to address these two issues by utilizing temporal information at the feature and proposal level, as well as improving the post-processing efficiency. The objective is to increase the mean Average Precision under the setting of ImageNet VID task [2].

## **Acknowledgement**

I sincerely thank my supervisor Dr. Kenneth Wong for providing valuable insights and instructions which have greatly assisted this project. I would also like to express gratitude to the PhD students in the computer vision lab – Mr. Wei Liu, Mr. Chaofeng Chen, Mr. Guanying Chen, and Mr. Zhenfang Chen, for their kind support regarding the implementation.

## Table of Content

1.	Overview .....	5
2.	Objectives .....	6
3.	Task Settings .....	6
4.	Methodology .....	8
4.1	Baseline T-CNN framework .....	8
4.2	Proposed modifications .....	9
5	Phase One & Two - Preparation .....	14
5.1	Project website .....	14
5.2	Environment setup .....	14
5.3	Reproducing the results .....	14
6	Phase Three – Experiments .....	18
6.1	Post-processing: selected MGP and NMS variants .....	18
6.2	Temporal information in Faster RCNN: volumetric convolution and propagated proposals .....	25
7	Future Work .....	37
8	Conclusion .....	38

## **List of Figures**

Figure 1.	.....	8
Figure 2.	.....	10
Figure 3.	.....	11
Figure 4.	.....	12
Figure 5.	.....	12
Figure 6.	.....	22
Figure 7.	.....	26
Figure 8.	.....	27
Figure 9.	.....	28
Figure 10.	.....	28
Figure 11.	.....	29
Figure 12.	.....	30
Figure 13.	.....	31
Figure 14.	.....	32
Figure 15.	.....	33
Figure 16.	.....	35
Figure 17.	.....	36
Figure 18.	.....	36
Figure 19.	.....	37

## **List of Tables**

Table 1.	.....	19
----------	-------	----

## 1. Overview

Object recognition, which refers to the task of identifying objects in image or video inputs, has become an active research topic over the recent years. There have been many models for object recognition on still images with impressive performance, while video inputs are much less experimented with. It has been shown that models trained on still images cannot achieve comparable results when directly applied to video inputs, which may be explained by the different characteristics of image data and video data. For example, video frames tend to be of lower resolution than still images, and the objects may experience motion blur, viewpoint changes or occlusion, all of which can make recognition more challenging.

Fortunately, videos are more than simple collections of images. Apart from information in a single frame, videos also contain contextual information across frames, which should be incorporated to help improving the performance. For example, objects from neighboring frames are expected to follow temporal consistency, suggesting that the locations and classification results for detections on different frames should not fluctuate dramatically. Consequently, the position of an occluded object may be inferred from detection results obtained in its neighboring frames.

To utilize these information and constraints, we chose to base our framework on T-CNN (Tubelet with Convolutional Neural Network [1]), the winning framework of the challenge “Object detection from video” in ILSVRC2015, which takes advantage of both temporal and contextual information. T-CNN addresses the problem by first using state-of-art detection frameworks, and employing temporal information through a set of post-processing techniques. Specifically, it uses motion-guided propagation to make up for missing detections to impose short-term temporal consistency, and uses tracking algorithms to generate “tubelets” followed a binary classification rescore step to exploit long-term temporal consistency,

Starting from T-CNN, we attempt to improve this baseline framework from two aspects, both on the single-frame detection results and the post-processing techniques. Regarding single-frame results, it was observed during our experiments that they could impose limitations on the effectiveness of T-CNN’s post-processing methods, which encouraged us to introduce temporal information early into the framework at the feature level so as to obtain more informative features, as well as add a temporal loss to encourage consistency between

neighboring frames. On the other hands, experiments have shown that some post-processing techniques may be the reason for the entire pipeline to be rather time consuming and computationally expensive. To address this, we propose to change the pipeline by rearranging the modules and trying to seek alternatives for tracking.

This report will first specify the objective of this project and the task settings. It will then explain the proposed modifications, followed by experiment results, and briefly discuss the future works.

## 2. Objective

The objective for this project is to improve the mean Average Precision (mean AP) on the 30 classes in ILSVRC2015 VID dataset [2]. A better performance is indicated by a higher mean AP, which is defined as

$$\text{mean AP} = \sum_{i=1}^N \text{ap}@n_i / N, \text{where } \text{ap}@n_i = \sum_{k=1}^n P(k) / \min(m, n)$$

In the above formula,  $N$  denotes the number of classes,  $P(k)$  is the precision at cutoff  $k$ ,  $m$  is the number of ground truth objects of the class, and  $n$  is the correctly recognized objects.

Due to some modifications and limitations explained below, the performance of this project will not be directly compared to the mean AP as reported in the original T-CNN paper. Instead, the performance baseline will be set as the results obtained in this project. Performance gain over the baseline will be used as the success measure.

## 3. Task Settings

### 3.1 Evaluation protocol

The evaluation protocol in this project follows the specification of the video challenge in ImageNet ILSVRC2015 [2]. Each detection result is in the form of  $(f_i, c_i, s_i, b_i)$ , where  $f_i$ ,  $c_i$ ,  $s_i$ ,  $b_i$  denote the frame number, the class ID, the confidence score and the bounding box

coordinates, respectively. Performance is measured by mean AP, where missing and duplicated detections for an object are penalized.

This project uses the training and validation set in video dataset VID, which contains 3862 video snippets (1122397 frames) and 555 video snippets (176126 frames) respectively. Both sets are fully annotated with 30 classes. The VID training set is split for training and validation (split ratio = 9:1, similar to the original ImageNet setting), while the VID validation is used for performance testing.

### 3.2 Regression loss

The regression loss is defined using smooth L1. The proposal network used in this project namely RPN outputs four entries, and the regression loss is defined the sum of the four:

$$\mathcal{L} = \sum_i |\hat{y}_i - y_i| = \|\hat{y} - y\|_1$$

Smooth L1 differs from L1 by using quadratic loss for small errors  $d$ :

$$|d|_{\text{smooth}} = \begin{cases} 0.5d^2, & \text{if } |d| \leq 1 \\ |d| - 0.5, & \text{otherwise} \end{cases}$$

### 3.3 Classification loss

The classification loss for the detection framework is defined using negative-log likelihood (NLL). Likelihood refers to probability and adds up to 1 for all classes.

$$\begin{aligned} \mathcal{L}(\theta | x_1, \dots, x_n) &= f(x_1, x_2, \dots, x_n | \theta) = \prod_{i=1}^n f(x_i | \theta) \\ \log \mathcal{L}(\theta | x_1, \dots, x_n) &= \sum_{i=1}^n \log f(x_i | \theta) \end{aligned}$$

Assuming the independency among different samples, accumulating likelihoods over all the samples is equivalent to taking the product of the likelihoods over all samples. Then by taking logarithm, the product of likelihoods becomes the sum of log-likelihoods. Finally, since likelihoods are less than 1, negating their logs results in positive values.

The goal of training the classification network is to maximize the likelihoods of generating correct predictions, which is mathematically equivalent to minimize NLL. In practice, NLL is preferred for being computationally more stable.

For example, the probability to output each of 31 classes (30 object classes + 1 background class) at the beginning of the training is 0.032 (i.e. uniformly distributed over each class), which corresponds to a negative log likelihood of 3.43.

## 4. Methodology

### 1. Baseline T-CNN Framework

Figure 1 shows the overview of the T-CNN framework [1], which consists of four parts:

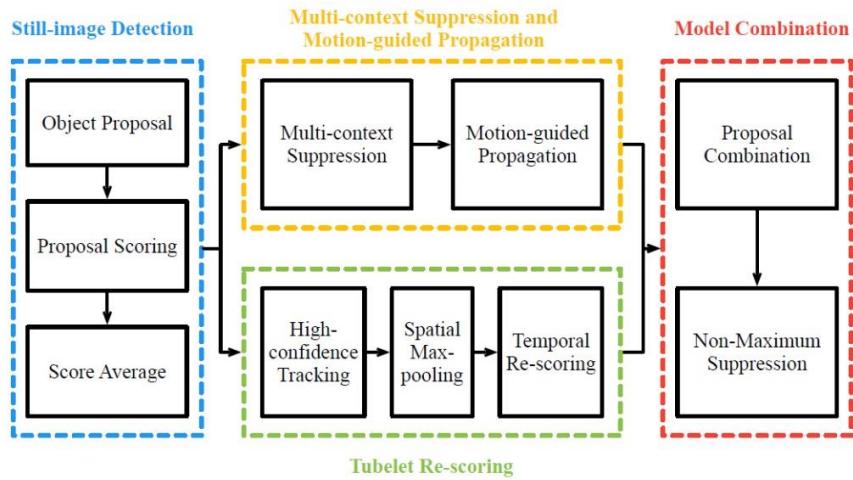


Figure 1. **T-CNN framework.** Reprinted from *T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos* by Kang K, Li H, Yan J, Zeng X, Yang B, Xiao T, Zhang C, Wang Z, Wang R, Wang X, Ouyang W. arXiv:1604.02532v3, 2016. With consent of the first author.

#### a) Still image detection

Each video is first parsed into a sequence of frames, on which T-CNN then uses DeepID-Net [8] and CRAFT [9] to perform still-image detection. Results from these two detectors will be treated separately in the second and third stage, and get combined in the final stage.

### **b) MCS and MGP**

The still-image detection results then enter the second module, which consists of two steps:

Multi-Context Suppression (MCS): MCS sorts all detections by their confidence scores, and sets the classes corresponding to the top detections to be “top classes”. Detections belonging to the top classes will then be awarded a bonus score, which gives them an advantage in the NMS process during score averaging. In this way, MCS essentially suppresses false positives.

Motion-Guided Propagation (MGP): First, an optical flow is calculated for each frame. Then for each frame, all detections in the current frame get propagated to neighboring frames, where the location of each detection is adjusted according to the optical flow. MGP can help make up for missing detections, hence mitigate the impact of false negatives.

### **c) Tubelet Re-scoring**

In addition to MCS and MGP, still-image detection results are fed into the third component to enforce long-term temporal consistency. For each video, a tracking algorithm first generates sequences of bounding boxes called “tubelets”, then refines the tubelet boxes by max-pooling over neighboring detection boxes. Each tubelet is then classified as either positive or negative, after which confidence scores of the tubelet boxes are re-distributed to increase the margin between positive and negative detections.

### **d) Model Combination**

Results from previous stages are combined by a non-maximum suppression (NMS) process to get the final scores.

## **2. Proposed Modifications**

Three modifications are proposed to reduce the computation cost of T-CNN and to improve single-frame results.

### **a) Selected MGP and NMS variants**

This modification is regarding the post-processing stage. During experiments it was found that MGP could lead to much longer running time, because it resulted in a larger number of detections which would affect later operations. This problem can

be mitigated by propagating a smaller number of detections, which is proposed to be achieved in two ways:

1. Move MGP to be after NMS: after MCS, perform non-maximum suppression (NMS) before MGP such that only a small number of high-confidence detections remain, which then get propagated to neighboring frames (as shown in Figure 2).
2. MGP only if above a threshold: instead of propagating all detections, only propagate detections whose confidence score is beyond a threshold.

Before:



After:



Figure 2: Selected MGP and NMS variants

Moreover, directly performing NMS for combining propagated results may not be appropriate, since the propagated locations could be inaccurate due to errors in the optical flows, and the confidence scores from different frames may not remain meaningful due to the change in image quality. We therefore experimented with several other methods to combine the propagated detections. Detailed explanations and results for these experiments are presented in section 7 – “Second Phase - Experiments”.

### b) Temporal information at the feature level

As discussed in the Progress section, the post-processing techniques have limitations, and the final results depend largely on the single-frame results. It is therefore necessary to improve the single-frame detectors. Inspired by MGP, we plan to use information from neighboring frames as contexts to complement the results on a certain frame. Instead of propagating boxes after the detections are fixed, we propose

to use contextual information at a feature level, such that the proposal network and the classifier can benefit from the enhanced feature maps.

The single-frame detector currently used in this project is Faster R-CNN [3], whose structure overview is shown in Figure 2.

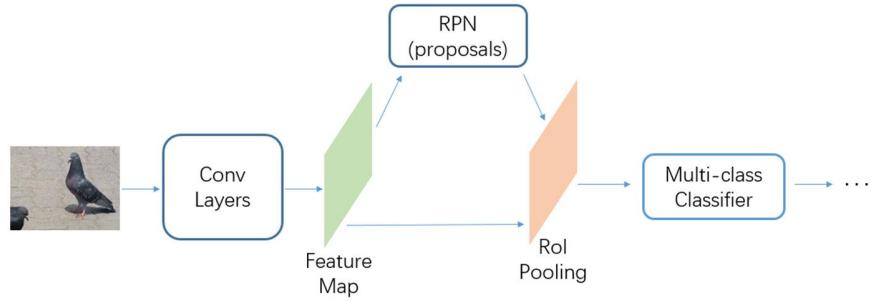


Figure 3. Faster R-CNN [3]

We propose two approaches to incorporate information from neighboring frames in faster R-CNN:

- 1. 3D convolution to combine feature maps from neighboring frames.** In faster R-CNN, feature maps are produced from one input image after going through a stack 2D convolutional layers. For this modification, we propose to use a set of 3D convolutional layers, which produce the same feature maps while using three frames as input (i.e. the previous, current and next frame). The weights for convolutional layers are shared to make sure to features extracted from different frames are the same and to keep the number of parameters moderate. The structure is shown in Figure 4.

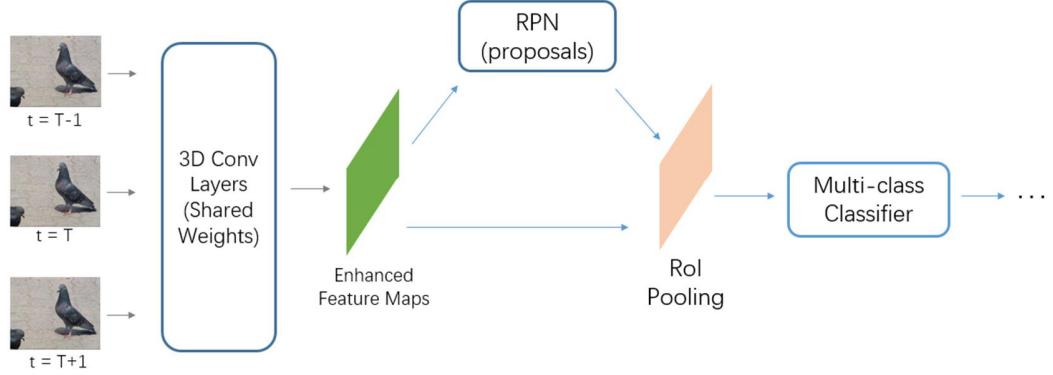


Figure 4. 3D convolution to combine feature maps

2. **Warp adjacent feature maps using optical flow.** This approach differs from approach 1 in that the combination of feature maps is guided by the optical flow, rather than being uncontrolled as in approach 1. After the first convolutional layer where low level features are extracted, feature maps from the previous and next frame are warped following the optical flow and merged with the feature map from the current frame. The combined feature maps then go through the rest of the convolutional layers as in Faster RCNN.

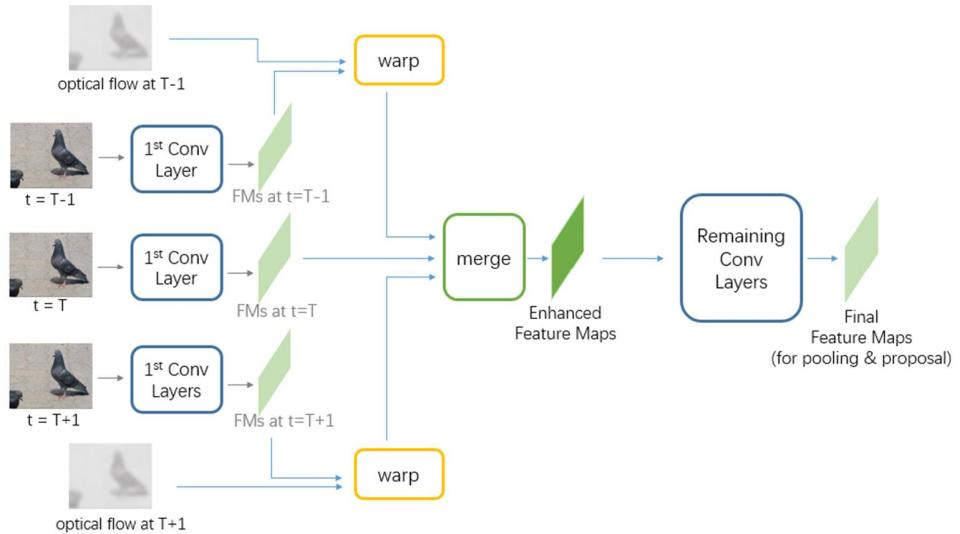


Figure 5. Merged feature maps using optical flow

### c) Motion-guided proposals

In MGP, propagating boxes helps to recover locations that have been previously missed out by the detection framework thus reducing false negatives. However, since both the location and the confidence score of a propagated box could be inaccurate, it would be desirable if they could be refined by the detection network.

Therefore, instead of propagating bounding boxes after the detection phase is completed, we propose to propagate the proposals before entering the classification phase, which would allow the last two fully-connected layers of Faster RCNN to refine the propagated locations and examine the class labels. This would not introduce much computation overhead to the detection framework since other parts of the framework are kept unmodified.

#### d) Temporal loss

Detection results on adjacent frames should be temporally consistent. Therefore, when training the detector, in addition to the classification loss and regression loss used in faster R-CNN, we propose to add an additional term for temporal loss, which penalizes the inconsistency between the current frame and the previous frame.

Each frame is associated with a heat map on which the temporal loss is calculated. The pixel value on the heat map is defined as the sum of the confidence scores of detections whose bounding box covers this pixel, and the temporal loss is defined as the difference between two heat maps:

$$L_{temp} = \frac{1}{W*H} \sum_{(i,j)}^{(W, H)} (X_{(i,j)} - Y_{(i,j)})^2,$$

where  $X^t$  is the heat map at time t and Y is the heat map warped from time (t-1) using the optical flow:

$$X_{(i,j)}^t = \sum_{b=(x_1, y_1, x_2, y_2)}^N s_b I_{x_1 \leq i < x_2, y_1 \leq j < y_2} \quad Y = w_{i-1}^i(X^{(i-1)})$$

Therefore the total loss now becomes

$$L = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) + \alpha L_{temp}(X^{(i)}, w_{i-1}^i(X^{(i-1)}))$$

where  $\alpha$  is an adjustable weight. The formula was inspired by [15].

## 5. Phase One & Two - Preparations

### a) Project Website

A project website is set up at <http://i.cs.hku.hk/fyp/2016/fyp16033/> which is updated with the latest project progress and documentations.

### b) Environment setup

This project runs on a server with 2 GPUs. Following the prerequisites for T-CNN, the server is installed with Caffe [4] for networks in the single-frame detectors and the tracker, GNU Parallel [5] for parallel computing, “matutils” [6] for easy file manipulation, and MATLAB Engine API [7] to support invoking MATLAB functions from Python.

### c) Reproducing the results

We reproduced the T-CNN framework and obtained a performance baseline. Below are the progress details and some major decisions made during this process:

#### i) Feasibility test

The entire T-CNN framework has been executed successfully on the server. Feasibility tests were conducted on one training snippet and one validation snippet. For each snippet, the framework correctly produced a set of tubelets, each tracking an object in the snippet.

Below are some major results obtained during the feasibility test:

#### a) Replace CRAFT with Faster R-CNN

CRAFT [9] is based on faster R-CNN, however is significantly more complex and more computationally costly due to the additional classifiers it uses. After trading off

between the performance gain and the computation cost, we plan to replace CRAFT [9] with Faster R-CNN.

**b) Reduce noisy proposals by EdgeBox**

Object proposals generated by EdgeBox [10] were examined through a simple GUI tool we built. It was shown that a large proportion of the proposals were about background regions, which did not contribute to the detection. Therefore, thresholding can be used to eliminate candidate proposal regions and thus help reduce the computation cost for detection.

In general, highly scored proposals are more likely to contain a target object, however this relation was shown to be not strict. When raising the threshold from 0 to 0.05, number of proposals on a sample image decreased from 1798 to 818, which reduced the detection time by around a half as expected. However, when the threshold was raised to 0.2, only 21 proposals were left, which all turned out to be highly scored background regions, suggesting that an overly strict threshold can be harmful. On the other hand, the lowly scored ones were safe to eliminate. After some experiments, the number of proposals was set to be between 100 and 300, i.e.  $\max(100, \min(score > 0.1, 300))$

**c) DeepID-Net was removed**

Object detection by DeepID-Net [8] seemed to be extremely slow. When directly using EdgeBox proposals, it took about 17 minutes for DeepID-Net to detect all proposal regions in a single frame, which would make the cost for testing prohibitive given the large number of frames.

After setting threshold to eliminate some proposals as discussed above, the detection time was reduced to around 1/6 of the original time.

We also tried to eliminate absent classes at an early stage, which resulted in a 5 time speedup. Because of the context information in videos, a class's failing to be detected in a large number of frames is a strong indicator that objects of this class are absent in this video snippet, in which case the detector can skip detecting for this class in the remaining frames. To avoid being too aggressive, the elimination would be

stopped when there were less than 5 classes remaining. 5 was considered to have enough buffer reserved for false negatives based on our observations on the video snippets. Typically, absence can be confirmed within 10 frames; however for long snippets with more than 100 frames, class counters were reset at every 100 frames to avoid being overly aggressive.

These two approaches resulted in a roughly 30 time speedup that reduced the detection time per frame to around 30 seconds, which however was still significantly more expensive compared to faster R-CNN's mean detection time of less than 0.05 second per frame. In addition, in the original T-CNN, DeepID-Nets produced worse results than CRAFT, and combining the results from two detectors only gave a slight performance gain over CRAFT of 0.2.

Therefore, after trading off between a considerable computation cost and a slight performance gain, DeepID-Net was removed from our framework. Though DeepID-Net is no longer in use, the techniques attempted to speed up and GUI tools we built to examine the results could be helpful in later experiments.

#### d) **Expensive tracking**

Tracking on a single video snippet took around 45 minutes, which was not too surprising since trackers are known to be computationally costly, but would be too time consuming to be used in each test. Therefore, we decided to temporarily remove it from the framework, and add it back after other parts of the framework are tested and fixed. In the meantime, we will seek for different tracking methods as replacement, which can be tested separately from the rest of the framework.

### ii) **Coarse baseline using Faster R-CNN trained on VOC dataset**

After the feasibility test, we ran T-CNN on the entire VID validation set to get a coarse baseline performance, and to estimate the cost for each step.

#### a) **Mean AP: 37.8 on 12 classes**

We used a faster R-CNN trained on PASCAL VOC, and took the results on the 12 classes common in VID and VOC. VID classes that are absent in VOC were padded

with zeros. The mean AP on these 12 classes is 37.8, which is much lower than the original T-CNN's performance of 67.8.

There may be two reasons that mainly account for this performance gap. First is the difference in faster R-CNN's training VOC and the testing data VID. For example, VID frames tend to be more blurry than VOC images, while blurriness is of major impact on performance [11]. Second, our faster R-CNN is implemented using ZF net [12], while the original T-CNN used VGG net [13]. The latter yields a better result, however it is 3.2 times slower than ZF and much more memory consuming [3]. Though being less accurate, ZF is chosen in this project due to limited time and computation resources.

**b) Post-processing techniques are effective but have limitations**

Results from faster R-CNN alone gave a mean AP of 30.2, while MCS & MGP and tubelet tracking altogether helped raise it to 37.8. The increase is 7.6, which is comparable to the performance gain of 5-7 as reported in T-CNN paper.

However, post-processing techniques have their limitations. The performance gains obtained in both the original T-CNN and our experiments are similar, suggesting that these techniques can only contribute to the performance to a certain extent. Moreover, their effectiveness relies on an assumption that false detections are not dominant, which may not always hold. For example, it was found that in some videos, false positives took up the better part of the detections and had some of the highest scores, leading to the correct class being suppressed in MCS and NMS. In other words, though post-processing techniques were effective, the final results are limited by the single-frame results. It is therefore necessary to improve the single-frame results, which has led to the second and third modifications in the Methodology section.

**c) MGP results in too many duplicates**

By duplicating detections, MGP results in larger detection files. Consequently, later stages such as tracking and score averaging take longer. For example, score averaging could take up to 10 hours on the results after MGP for one video, while without

MGP it only took about 4 hours on all 555 testing videos. Possible solutions such as changing the order of MGP or propagating only high confidence detections were discussed in the methodology section.

### iii) Training Faster R-CNN with VID

We have trained faster R-CNN using a subset of VID training set, which consists of 1000 randomly sampled images per class for training and 150 for validation. The resulting mean AP is 21.4, even lower than the result using faster R-CNN trained on VOC.

This poor result may be due to overfitting and large learning rate. The performance on validation set was 52.6, much higher than the testing error, suggesting the need of using a larger training set. The error plateaued after 40k iterations, therefore decreasing the learning rate afterwards may help drive down the error. Since we plan to switch from Caffe to Torch for faster R-CNN, we decided to perform hyper parameter tuning after the Torch implementation is ready.

## 6. Phase Three - Experiments

### 6.1 Post-processing: selected MGP and NMS variants

#### 6.1.1 Selected MGP

T-CNN propagates all detections from neighboring frames to the current frame in an attempt to make up for false negatives, then conducts NMS to combine these detections. However, this approach may lead to redundant computation. Since optical flows do not affect the relative positions among detections, if a bounding box is going to get suppressed in NMS on its own frame, then it is very likely for this box to get suppressed again after getting propagated to some neighboring frame. Therefore, T-CNN's way of conducting MGP may result in too many unnecessary duplicates which lead to redundant computation and can hinder the performance. It is therefore natural to attempt to reduce the number of detections being propagated, to which we proposed two ways:

- a) Adjust the order in which NMS and MGP are performed

We attempted to perform NMS twice: the first NMS is performed separately on each neighboring frame; then after the resulting highly scored detections get propagated, the second NMS is performed on the current frame to get the final results. Performing NMS twice may sound to require repetitive efforts, but it has been shown in our experiments that this can significantly reduce the time for MGP. As a comparison, the original MGP over the VID validation set would require around 11 hours, while our modification reduced the time to less than 4 hours.

It was also discovered that splitting NMS into 2 stages helped improve the overall results. As shown in Table x, the mean AP increased from 25.1 to 26.6. A possible explanation to this may be that some relatively highly confident detection boxes from neighboring frames can get eliminated early when performing NMS within each frame. As explained in 6.1.2 (b), the confidence score may not be very meaningful for propagated boxes, therefore NMS within each frame can get rid of some neighboring detections that are otherwise going to suppress the more accurately located boxes from the current frame.

### b) Propagate only high-confidence detections

This approach gave the same results as the original MCS & MGP, which is understandable since it simply avoids duplicating low confidence detections which are going to be ruled out in the final NMS, and does not help eliminate the error caused by highly confident propagated detections.

Results for (a) and (b) are presented in the following table:

Class	Airplane	Antelope	Bear	Bicycle	Bird	Bus	Car	Cattle	Dog	Cat
original MGP	63.2	20.2	19.0	36.9	16.1	28.8	32.7	26.5	12.7	5.4
selected MGP (a)	65.1	20.5	19.8	40.9	18.0	29.1	34.4	27.2	13.5	5.7
selected MGP (b)	63.2	20.2	19.0	36.9	16.1	28.8	32.7	26.5	12.7	5.4
Class	Elephant	Fox	Giant-panda	Hamster	Horse	Lion	Lizard	Monkey	Motorcycle	Rabbit
original MGP	33.0	33.8	53.0	10.9	40.6	0.0	6.5	12.9	56.9	2.0
selected MGP (a)	35.5	35.5	55.7	11.6	45.5	0.0	7.9	14.8	57.9	1.8
selected MGP (b)	33.0	33.8	53.0	10.9	40.6	0.0	6.5	12.9	56.9	2.0

Class	Red panda	Sheep	Snake	Squirrel	Tiger	Train	Turtle	Watercraft	Whale	Zebra
original MGP	5.3	25.8	4.0	1.6	33.9	34.0	13.6	35.7	25.2	63.1
selected MGP (a)	6.2	26.7	4.3	2.1	36.9	36.4	13.5	38.1	27.1	65.0
selected MGP (b)	5.3	25.8	4.0	1.6	33.9	34.0	13.6	27.1	25.2	63.1

Table 1. Results of MGP

Mean AP:

Original	25.1	Selected (a)	26.6	Selected (b)	25.1
----------	------	--------------	------	--------------	------

### 6.1.2 Different methods to combine detections

In T-CNN, propagated detections and detections from the current frames are collected, over which NMS is performed where propagated detections and detections directly obtained from Faster RCNN are treated equally when getting combined. This method may sound questionable, since detections directly obtained from the detection framework are supposed to be more reliable than propagated detections which are likely to be negatively affected by inaccurate optical flows. Therefore, we tried to assign different priority to the detections, and proposed with different methods to combine the propagated detections in addition to NMS.

Following is a list of methods we have experimented with:

- a) Plain NMS (baseline)

This is the method used in T-CNN, where the final results are given by performing NMS on detection boxes from both the current frame and neighboring frames.

- b) Keep current-frame results with high confidence scores

The plain NMS treats all detections equally, which means a propagated detection can suppress a detection on the current frame because of a higher score. However, it may not be appropriate to directly compare the scores of detections from different frames, since the frames may not be of comparable image quality. For example, an object may be detected with high confidence in frame at time  $t$  but much less so in frame at time  $t+1$  because the latter is more blurry. Then for frame at time  $t+1$ , its own best detection will

be suppressed by a propagated one, which has a not very meaningful high score and is at a worse location.

Therefore, it is the bounding box locations rather than confidence scores that is more likely to remain meaningful after the propagation. Since imperfect optical flows will make detections propagated from neighboring frames less accurate, more priority should be given to detections in the current frame. This method therefore tries to keep the highly confident detections from the current frame, and only takes consideration of the propagated boxes when none of the current detections has a confidence higher than a threshold.

c) Weighted NMS

This shares the same idea with (b) to give more confidence to detections on the current frame, where different weights are assigned to the bounding boxes such that detection from a closer frame will have a better impact. Weights decreasing linearly with the frame difference and weights following 1D Gaussian distributions were tried out.

d) Averaging

This method uses the weighted average of detection boxes as the combination result. In NMS, boxes get suppressed when there exists a box with higher score and an IoU beyond a certain threshold. Hence equivalently, NMS can be considered to being selecting the most highly scored one out of a cluster consisting of the most highly scored and the suppressed detection boxes.

In this method, clusters of detections are formed following the same IoU criteria, but instead of picking the one with maximum confidence score, it outputs the weighted average of the cluster. To form clusters, all detection boxes are first sorted according to their confidence scores; the list of boxes then gets traversed starting from the most highly scored detection, and each remaining box (i.e. with a lower confidence score) is added to the cluster if its IoU is beyond a threshold. Similar to Kalman filter [14], propagated boxes correspond to predictions, and proposals in the current frames are present measurements directly obtained from the detection framework. Karlman filter has shown that weighted

average considering both the predictions and the measurements can yield better results, therefore we expect proposals generated in this way to be more reliable.

e) Remove detections corresponding to parts of a larger detection

It was discovered that some false positives are caused by labeling some part of an object. For example, in the left image of Figure 6, the chimney, wheels, and the roof of the train each gets detected separately, which is understandable since these areas all contain rich features.

In circumstances like this where the detection corresponds to a part of a larger object, the IoU between the part's bounding box and the bounding box for the object is approximately the portion of the part with respect to the entire object. Consequently, the IoU can be low when a part is small, in which case NMS may fail to suppress the part's bounding box with the object's and thus result in false positives.

Unlike IoU, such false detections usually has a high “Intersection over Minimum” (“IoM”), which is defined as the area of intersection over the area of the smaller bounding box. As an example, the IoU for the chimney, wheels, and the roof are 0.06, 0.1662, 0.1444 respectively, all of which are below the IoU threshold 0.5; but their respective IoM are 0.8312, 0.7588, and 0.9515, which are far beyond the threshold (this could be a different threshold) and thus can be eliminated safely. Figure 6 shows a comparison between using an IoU of 0.5, versus using an IoU of 0.5 and an IoM of 0.7 when the IoU is below the threshold.



Figure 6. Bounding boxes obtained from NMS (left) and after removing parts of objects (right)

f) Remove false positive clusters

As mentioned previously, clusters can be formed by grouping detections whose IoU with a particular more highly scored detection is beyond a threshold. Because of the first NMS before propagation, boxes present at the combining stage usually come from different frames (with a possible exception being two disjoint small detection boxes from the same frame being added to a cluster formed by a more confident larger box from a different frame). Therefore, the number of boxes in a cluster is roughly the same as the number of frames which gives a positive detection result; hence the higher the number of boxes a cluster has, the more confident it is for this cluster to cover an object. Conversely, if a cluster consists of too few boxes, then it is very likely for this cluster to be a false positive, in which case it should be eliminated.

In the experiments, the threshold is set to be 2 for a window size (i.e. number of frames from which detections get propagated) of 5 and to be 3 for a window size of 7. Boxes belonging to clusters with fewer detections are removed.

g) Keep only frequently appearing classes

It is worth noting that the process described above only takes into consideration the confidence scores, but not the class to which the detection belongs. This is understandable since at this point the class label is yet to be refined. To refine the class using contextual information, we would like to assign more confidence to detections from more frequently appearing classes, which has a similar rationale to that of MCS which adds a bonus point to classes with the mostly highly scored detections.

When forming clusters for each frame, the program keeps track of the number of detections or the sum of their confidence scores corresponding to each class. Classes with the highest count or sum of confidence scores are considered as “true class”. Other classes whose count is below  $\frac{1}{2}$  of the true class’s count or whose sum of confidence scores is below  $\frac{2}{3}$  of the true class’s sum are removed in an attempt to further reduce false positives.

(f) and (g) both attempt to utilize information which can reflect temporal consistency at a longer term. Though there was no explicit matching procedure between boxes from different frames, statistics such as number of proposals and the range of confidence scores are expected to help relate the detection results and exploit temporal consistency.

In addition, some of the above methods can be used together. For instance, false positive clusters can be removed before using weighted average, and detections that are likely to be parts of a larger object can be removed during NMS. Using Caffe implementation of Faster RCNN model trained on a dataset of 30K images, we experimented with several combinations with varying window sizes, and some of the results can be found in the appendix.

A summary of the results is as follows:

For NMS:

- a) Window size 5 or 7 has negligible effect;
- b) False positive clusters should be kept;
- c) Keeping only “true classes” had no effect;
- d) Weighted NMS can help improve the results, but different kinds of weights had similar effects.

For averaging:

- e) Smaller window size is preferred;
- f) False positive clusters should be removed;
- g) Removing parts of a larger object is helpful.

Overall, NMS methods produced much better results than averaging methods. Removing positive clusters or “false classes” seemed to be too aggressive in eliminating results for NMS methods, however they appeared to be desirable for averaging methods. Together with the observation that averaging methods produced much poorer results, this could be explained that the averaging process is prone to noisy results, therefore aggressively removing less reliable detections may effectively decrease the disturbance. This also agrees with the observation that smaller window size tends to give better results.

Some other methods may only be useful to specific classes. For example, removing parts of a larger detection resulted in a 3.1 increase in average precision for “train”, but also a 3.6 decrease for “sheep”.

To conclude, selected MGP can help to improve the results, while NMS variants did not provide significant improvement over all classes.

## 6.2 Temporal information in Faster RCNN: volumetric convolution and propagated proposals

As explained in the methodology section, it may be beneficial to make use of the temporal information early in the feature level. Instead of detecting on single images, we propose to feed the network with 3 input images (the current frame, and the frame before and after), hoping the combined feature maps to yield a better result.

### 6.2.1 Training data

The training dataset consists of 64216 unique images containing 30 classes. For the list of videos corresponding to each class, at most 50 video snippets are sampled for each class to make sure the training data will not be dominated by only a few classes. For each video snippet, frames are sampled at an interval of 3, for example, the 1<sup>st</sup>, 4<sup>th</sup>, and 7<sup>th</sup> frames. The reason for not using adjacent frames is that adjacent frames may be too similar to provide notable variances which are expected to contain complementary information. The choice of interval value 3 resembles the T-CNN’s MGP window size of 7. All training data comes from VID training set and are fully labeled. When used in training, each image is normalized by subtracting the image mean and divided by standard deviation. Horizontal flip is used for data augmentation.

### 6.2.2 Changing from Caffe to Torch

Previous results are obtained based on the official Caffe implementation [3] of Faster RCNN. Due to the complexity of Caffe, we decided to change the framework to Torch [16], which is generally considered to be more flexible and easier to experiment with. The baseline torch implementation is <https://github.com/andreaskoepf/faster-rcnn.torch> which credits to [Andreas Kopf](#).

For this torch implementation, we changed the backbone structure from VGG net [13] to ZF net [12], since the latter is easier to train by having fewer layers and parameters as well as more efficient both computationally and in terms of memory consumption.

### 6.2.3 Joint training to alternating training

The baseline implementation jointly trains the network, where proposals are produced by RPN based on features obtained from the 5 convolution layers, then get fed to Fast RCNN to perform ROI pooling on the same set of feature maps for the classification results.

Figure 7 shows the curve for joint training using RMSprop as the optimization algorithm, with a batch size of 128 and a learning rate of e-5.

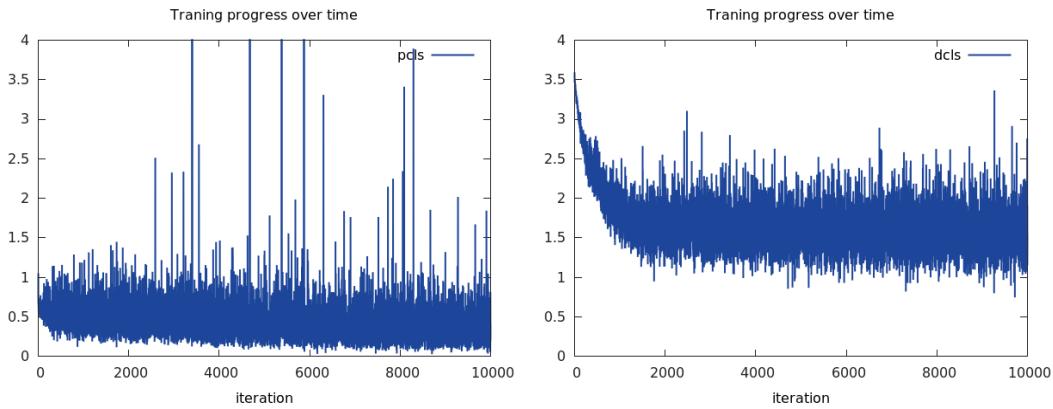


Figure 7: joint training: classification error for fg/bg proposals (left) and classification for 31 classes (right)

The graph on the right show that the classification error for the 31 classes plateaus at around 1.5-1.6, which corresponds to a likelihood of 0.213. Different batch sizes, starting learning rates and learning rate update strategies were also tried out but yielded similar results. As a comparison, the classification loss for the official Caffe implementation of Faster RCNN is 0.246 at the end of training, which corresponds to a likelihood of 0.77. The Caffe model's loss is 3.548 at the beginning, and drops to 0.510 after around 20 iterations, and drop from 0.367 to 0.246 during iteration 2000 to iteration 60000. For torch models, the loss curves were similar when using VGG and ZF.

This training loss did not seem to update very effectively based on our training results and some comments found online, which we assumed might be related to the joint training strategy. Since Faster RCNN shares convolution layers between RPN and Fast RCNN, in the implementation the two differ by an anchor network or two fully-connected layers after the convolutional layers. A possible explanation to the poor training results is that RPN and Fast RCNN may update weights for the shared convolutional layers differently which makes convergence difficult. Therefore, we decided to change the strategy to alternating training.

For alternating training, the proposal network RPN and the classification network Fast RCNN are trained separately, where Fast RCNN uses the proposals produced by RPN.

### 6.2.3.1 Proposals

#### a) RPN

We started with generating proposals using RPN, which is the same torch implementation as mentioned above. Figure 8 shows some examples of training batches.

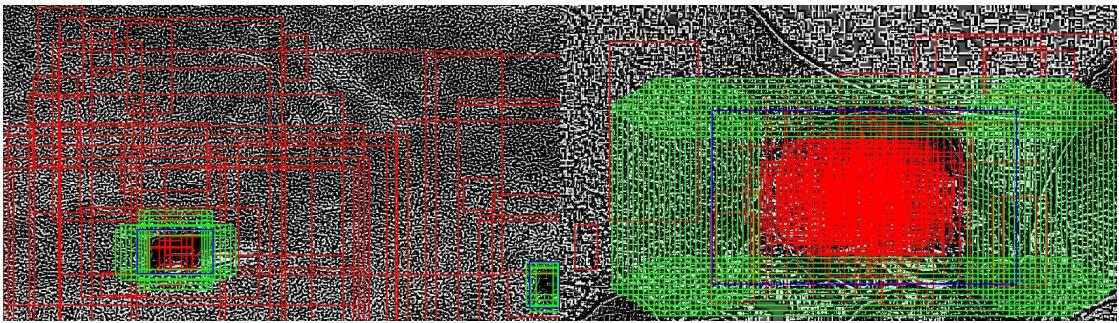


Figure 8: Examples of training batches: blue, green, and red boxes denote the ground truth, positive samples, and negative samples, respectively.

To obtain positive samples, first scan the region containing ground truth in a sliding-window fashion, and prepare 12 candidate boxes for each location within this region, corresponding to three aspect ratios (1:2, 1:1, 2:1) and four scales (48, 96, 192, 384). Candidate boxes with a ground truth IoU higher than a threshold are marked as positive samples.

Negative samples are obtained from two approaches. The first approach samples boxes at random locations with an IoU below the threshold; the second approach generates boxes which share the same anchors with the positive samples, however are at a much larger or smaller scale such that the IoU is low. The combination of these two types of boxes guarantees the balance of positive and negative samples in each training batch.

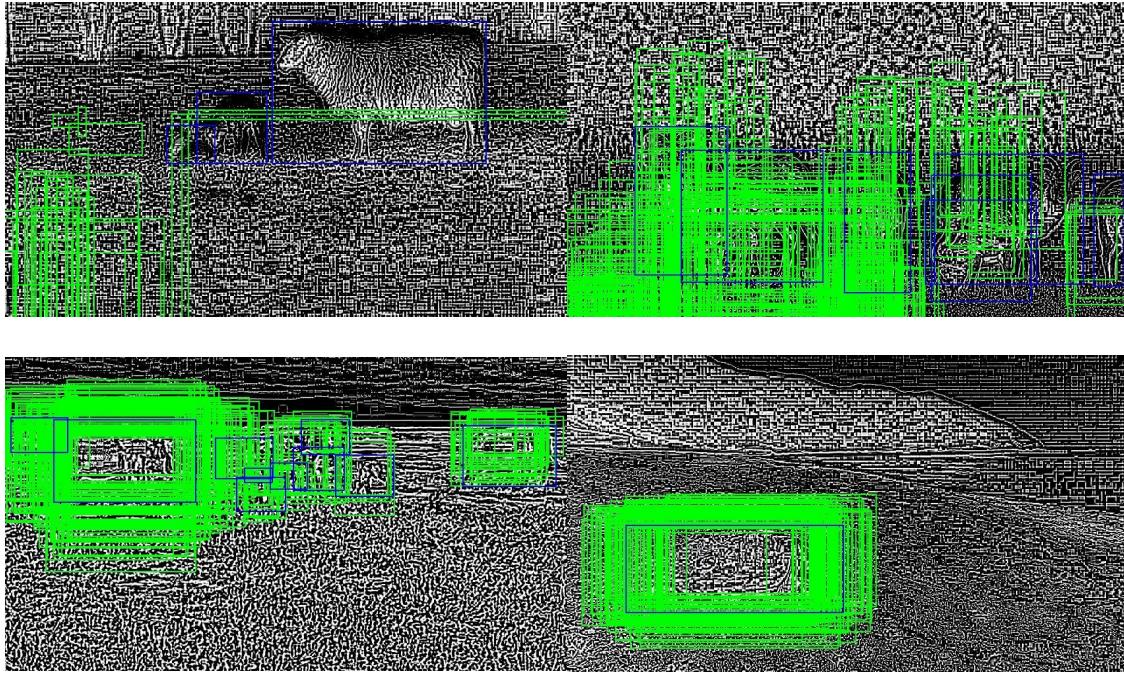


Figure 9: First row: examples of proposals generated in early iterations; Second row: examples of proposals generated in later iterations. Proposals are in green and ground truths are in blue.

RPN is used by the official Caffe implementation of Faster RCNN and is reported to give decent results. However, our torch implementation failed to achieve similar performance. As shown below,

Proposals generated in forward passes during the training were used as a temporal solution.

#### b) EdgeBox

As the proposals generated by our implementation of RPN were not satisfying, we tried to use EdgeBox [10] to obtain proposals. Figure 10 shows some examples of EdgeBox's proposals.



Figure 10: proposals successful generated by EdgeBox.

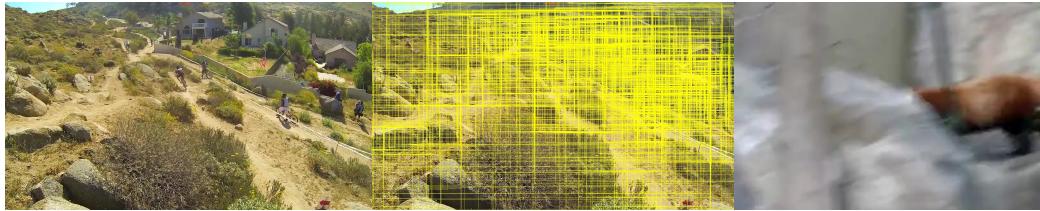


Figure 11: images where EdgeBox fails to produce reasonable proposals:  
(left) too many boxes for the bicycles; (right) zero box proposed for the bear.

On the other hand, Figure 11 shows that the quality of proposals can vary significantly. It can be seen that EdgeBox excels at objects that are well-posed and contrast strongly with the background; however, there are also cases where EdgeBox may fail. As in the bicycle example, small objects may fail to be detected, and the large number of proposals may be due to the rich features in the background. For the bear example, the image's being blurry and smooth changes may have caused EdgeBox's failing to produce any proposals.

#### 6.2.3.2 Classification

Fast RCNN used in this project is forked from <https://github.com/mahyarnajibi/fast-rcnn-torch> and credits to [Mahyar Najibi](#). The classification loss converged for a test run on a small dataset consisting 300 images. The loss curve for training is presented in Figure 13 together with the curves for the modified versions.

#### 6.2.4 Spatial convolution to volumetric convolution

As mentioned earlier, it seems that all the existing methods for video object recognition integrate temporal information after single-frame detection results are obtained, which may not be able to maximally take advantage of the information since the single-frame results usually pose limitations on the performance. This project therefore attempts to introduce these temporal information early in the detection framework at the feature level.

##### **h) Sequences of three frames as inputs**

To achieve this, three input images are fed into the detection framework rather than a single one, changing the input tensor from dimension  $3 * W * H$  to  $3 * 3 * W * H$  (the first 3 is number of channels per image, the second 3 is the number of input frames).

Following the architecture suggested in [17], the three images are first forwarded through a common convolutional layer, where the weight sharing is ensured by setting the filter size along the temporal dimension to be 1. The resulting feature map is now  $96 * 3 * W' * H'$ , which is then reshaped (i.e. concatenated along the time dimension) to be  $288 * W' * H'$ . A  $1*1$  spatial convolution is then performed to reduce the feature map to a dimension compatible in later layers, followed by a spatial max pooling layer and a cross-channel local response normalization layer. The remaining part of the network is the same as the original ZF net. Figure 12 shows the network structures of the original ZF net and our volumetric version.

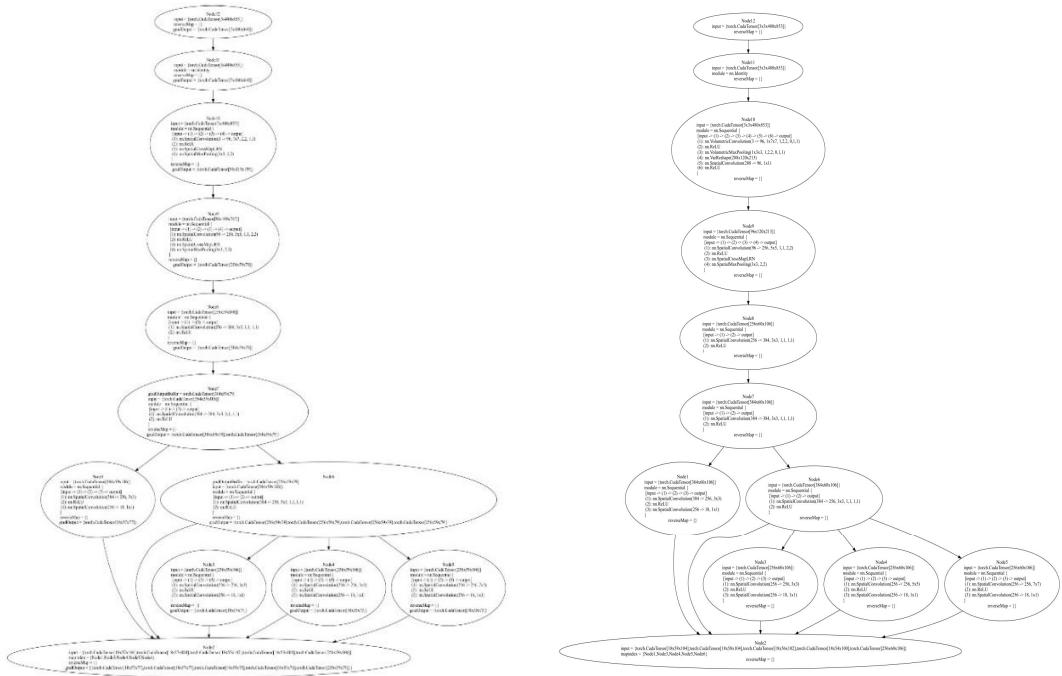


Figure 12: (left) original ZF net; (right) modified ZF net with the first layer changed to volumetric convolution

i) Spatial filter size of  $1 * 1$

The spatial convolutional layer following the volumetric convolution and reshaping is expected to select the best features out of the three feature maps obtained from the three frames. Note that the filter size of is  $1 \times 1$ , which means no further spatial variances is considered in this layer. We consider this to be acceptable, since the shared volumetric convolutional layer has a large filter size of  $7 \times 7$  (omitting the temporal dimension which

is 1), which should have taken into consideration a region that is large enough to cover the variances among neighboring frames.

### j) Combining feature maps after two shared convolutional layers

In addition to the architecture described above, another network with the first two convolutional layers being volumetric was tried out, which combines feature maps at a deeper depth than as considered in [17]. [17] only experiments with architectures that fuse feature maps after the zeroth (i.e. directly concatenating the input images) or first convolutional layer, both of which are rather shallow. Our explanation for their choosing such shallow layers is that their tasks are all concerning lips, whose features are much more constraint than general images as used in this project. Using feature maps after two convolutional layers is expected to encode higher level information, and may be more robust to spatial variations.

The above architectures were used for both RPN and Fast RCNN. Figure x shows the loss curves for training Fast RCNN, where the unit for the x-axis is 200 (i.e. there are 90k iterations in total).

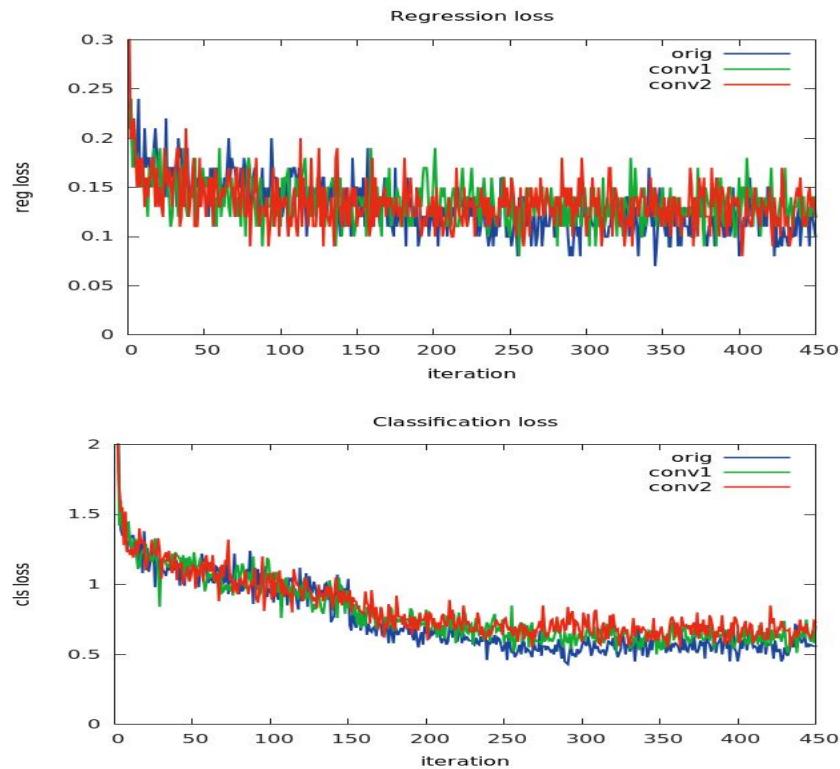


Figure 13: (upper) regression and (lower) classification loss for Fast RCNN

It can be seen from the graphs that adding volumetric convolutions seem not to help increasing the performance. A possible explanation could be, though expected to be tolerant to moderate spatial translations, these networks may be harder to train, since the parameters of the combining spatial convolution layer need to be learned such that it can select the best features among the three frames, which would require encoding information regarding relative positions. However, motions of different directions would update the parameters differently. For example, an object moving to the left may update the parameters such that more weights are assigned to the right-hand side for the feature map from the previous frame and more weights to the left-hand side for the feature map from the next frame, while an object moving to the right would prefer exactly the opposite. Since the training examples contain motions in different directions which correspond to optical flows in different directions, the net effect over the entire training set may turn out to be isotropic, causing information from neighboring frames to be neglected.

One possible way to solve this problem would be to take motion into consideration, such as to warp the feature maps using optical flows before combining them. This was proposed earlier but not able to be included in this project, therefore we decided to move it to future work.

#### 6.2.5 Propagating proposals

In this modification, proposals from the previous and next frames are combined with the proposals generated in the current frame before forwarded into the classification layers. Proposals are also obtained from RPN, and optical flows are generated using Farneback's algorithm [18] provided in OpenCV. Figure 14 gives an example of warping, where the middle image is warped from frame 0 following the optical flow from frame 0 to frame 1.



Figure 14: (left) frame 0; (middle) warped frame; (right) frame 1

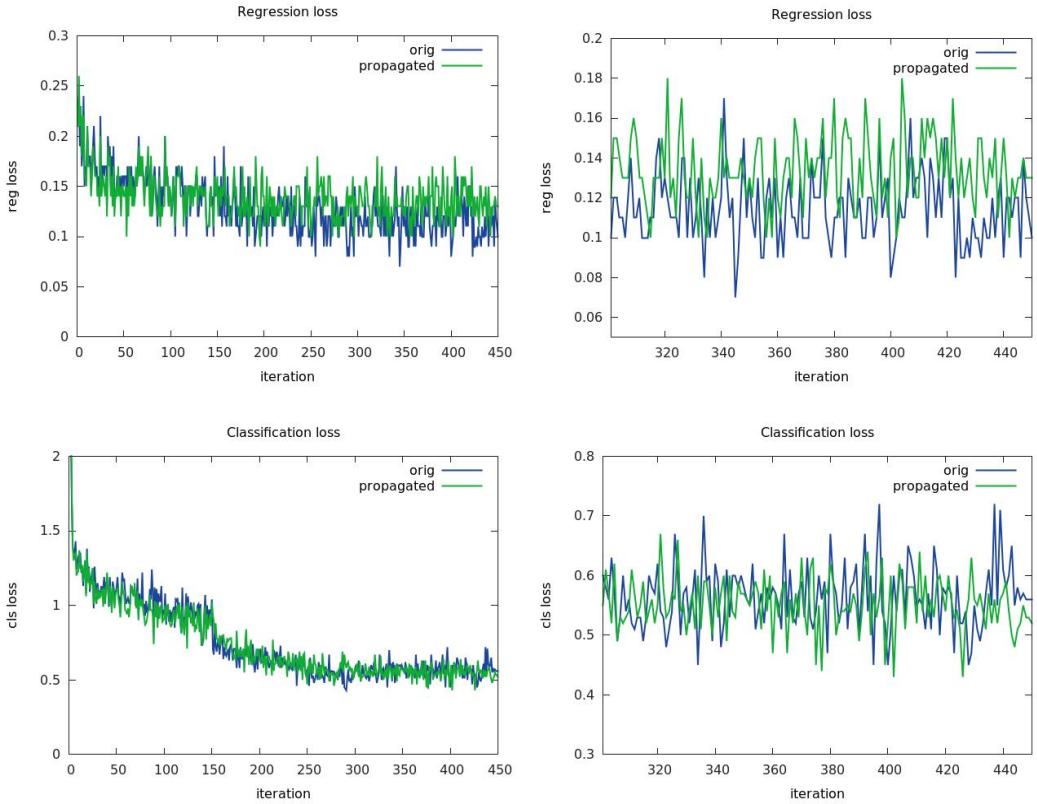


Figure 15: comparison of Fast RCNN training curves between the original settings (blue) and using propagated proposal (green). First row: regression losses; second row: classification losses. First column: statistics for entire 90k iterations; second column: statistics for the last 30k iterations.

Figure 15 shows the training loss for Fast RCNN, comparing the original version and the one using propagated proposals. The two methods are very similar overall, with the original version slightly better at regression and the propagated version having slightly more accurate classification results. A possible explanation could be some inaccurate optical flows may have caused the propagated proposals' locations to be slightly off, however the disposition is relatively minor so that the propagation can still help recover some of the false negatives.

#### 6.2.6 Pending problem: RPN losses not converging

For our torch implementation, after 60k training iterations, the classification loss dropped to around 0.3, which is comparable to 0.216 from the official Caffe implementation. However the regression loss were around 0.1, while the Caffe implementation's was around 0.005.

This problem was first discovered when testing the trained RPN on images. We mistakenly thought the problem lied in the detection program, therefore after several unsuccessfully attempts to debug, we decided to import the detection algorithms used in the Caffe implementation in torch. After failing again to produce good results with the imported detection algorithms, we realized that the problem may lie in model training.

Here are some aspects which were examined during debugging:

- k) Varied learning rates and batch sizes: e-3, e-4 and e-5 were used as the starting learning rate. e-3 seemed to be too large since it resulted in significant fluctuation in the error, and e-5 seemed too small. Starting from e-4, the learning rate was updated in different ways, including decreasing by a constant factor for every constant number of iterations (e.g. decreasing by 10 for every 10k iterations, or by 5 for every 5k iterations), or decreasing by 10 when there was no significant change in loss for the last 5k iterations compared with the previous 5k iterations.
- l) Optimization algorithm: SGD was used as the optimization algorithm since it is considered to be more stable than RMSprop despite being possibly slower.
- m) Data preparation: the image database used in training was checked to ensure that annotation files and data files match with each other and that annotations are loaded correctly. There was a small dataset consisting of 300 images for test run.
- n) Pretrained models: we wondered whether not converging was due to not using a pretrained model.
- o) Anchors and training batches: scale and ratio jittering were used in generating anchors. Training batches were visualized and seemed correct.
- p) Training proposals: proposals generated by the network forward pass during training were visualized; as shown in Figure 9, proposals generated by later iterations seemed to be more accurate than those generated at the beginning of training.

### 6.3 GUI tools

We have developed and used some GUI tools to help with visualizing the results and debugging.

#### a) Displaying bounding boxes

This tool was used to examine proposals generated by EdgeBox [10] and RPN [3]. For each frame, proposals with a confidence score higher than the threshold are displayed one at a time. Each proposed region is labeled with its coordinates followed by the confidence score. The threshold for confident scores can be specified through a input box, whose value is displayed at the upper left corner along with the proposal ID and the number of proposals after thresholding. “Prev” and “Next” iterate through the list of proposals, and “Jump To” allows selecting proposals by index.

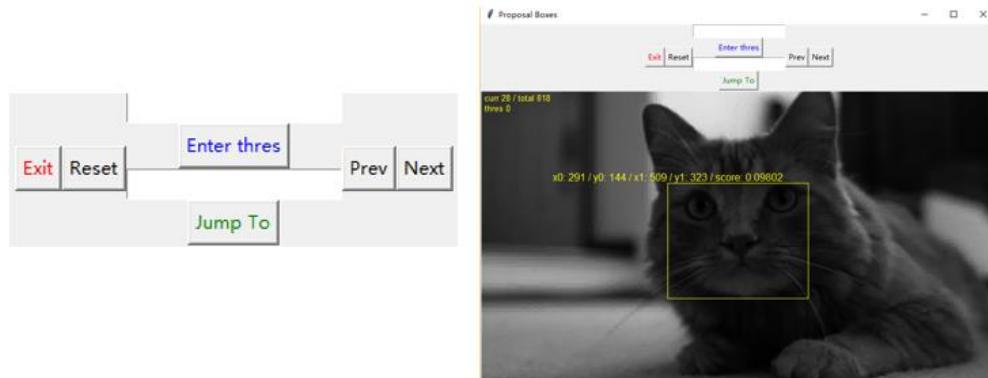


Figure 16: Control tabs and the interface for visualizing proposals

#### b) Visualizing tubelets

This GUI allows to display all objects in the video, or tubelet boxes a specified object of interest. By default non-maximum suppression (NMS) is turned on, which means there is only the tubelet box with the highest score is displayed. NMS can be opted out using the “Switch” button, after which bounding boxes from all tubelets will be displayed for each frame. “Play” displays the selected tubelet(s) in the sequence of frames as a video.

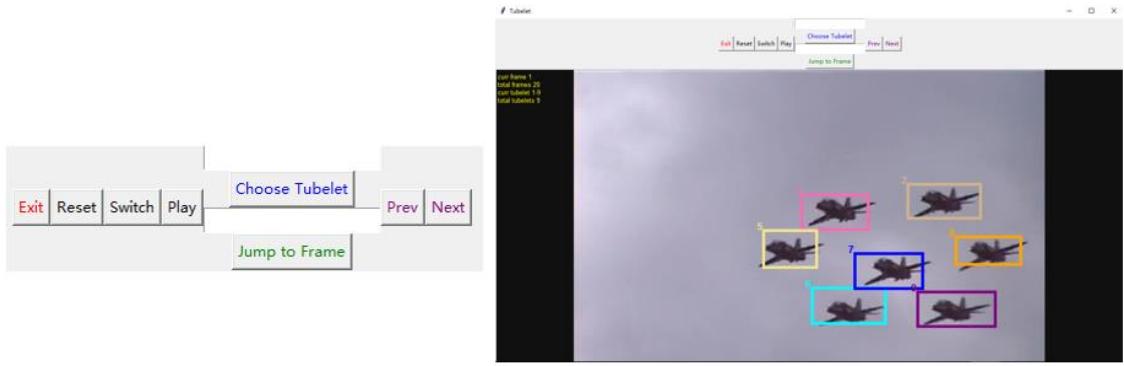


Figure 17: Control tabs and the interface for tubelets.

### c) Visualizing detections

This tool takes as input a txt file containing the detections and an image file (in .pgm format), and displays the gray-scale image with detections labeled with the class ID, confidence score as well as the four coordinates.

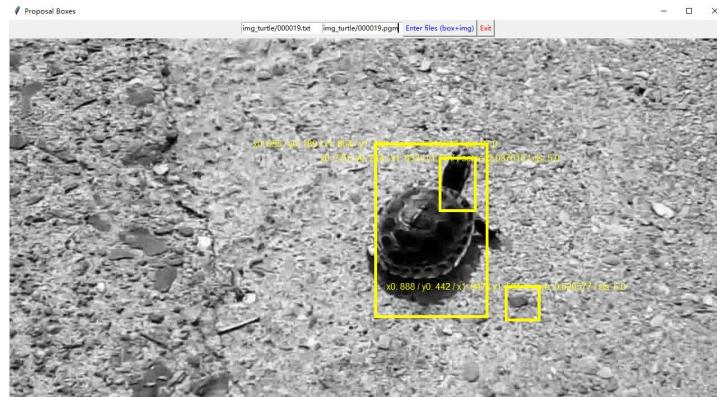


Figure 18

### d) Debugging the network (installed package)

We used package “graphviz” [19] as a debugging tool to visualize the networks (developed using “nngraph” [20]). Below is an illustration of how graphviz helped us locate the error to be in a self-defined module named “VarReshape”.

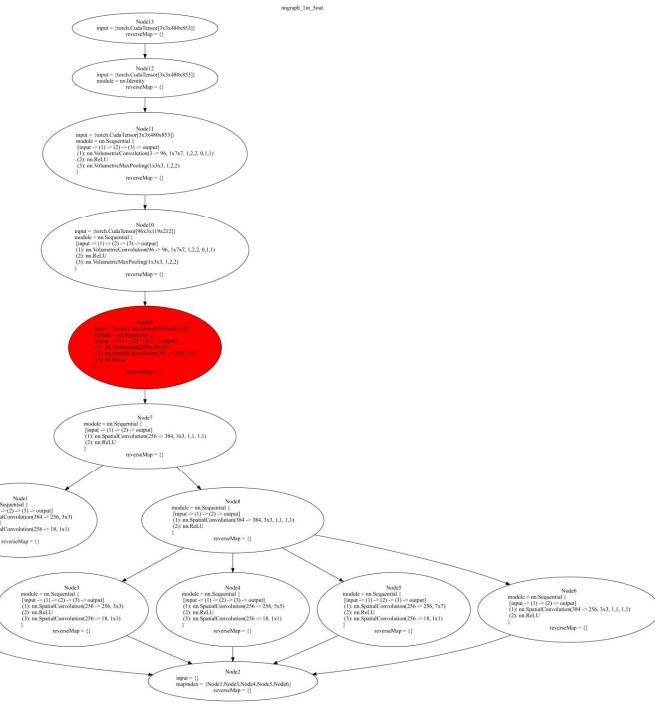


Figure 19: graphviz marks the erroneous module in red

## 7. Future Work

Due to false estimations of the work load, limited programming skills, improper use of time and planning, the following proposals have not been able to be carried out in this project.

## 7.1 Warping using optical flows when combining feature maps

Networks described in section 7 directly use a spatial convolution layer to combine the concatenated feature maps, which may be hard to train due to different motions in the inputs. This motivated us to use optical flows to warp the feature maps from neighboring frame before combining them, such that the effect caused by motions could be counteracted.

In addition to Farneback’s algorithm [18] as used in propagating proposals, we were also suggested to use FlowNet [21], as it can generate more accurate optical flows as well as relatively easy to get integrated into the network. It is worth noting that there is a recent work [22] sharing almost the same idea of flow-guided feature aggregation, which can be considered

as an evidence for the feasibility of introducing temporal information at the feature level. [22] uses ResNet [23] as the backbone structure and FlowNet [21] for generating optical flows, and has shown. [22] has presented some valuable techniques and details about training, which we will learn to adopt in the future.

## 7.2 Temporal Loss

Temporal loss was proposed earlier but not able to be carried out, so we decided to make it part of the future work.

# 8. Conclusion

This project aims at improving the mean Average Precision for object recognition with videos, where we attempted to improve the baseline framework T-CNN from both the single-frame detection framework and the post-processing steps.

For post-processing steps, selected MGP has reduced the running time to around one third of the original, and can also help increasing the results by avoiding highly confident detections at inaccurate locations. Other methods to combine the propagated boxes such as weighted average and removing false positive clusters were tried out, however did not help increasing the performance.

Regarding single-frame results, temporal information was introduced to the detection framework at the feature level and proposal level. These have not resulted in notable improvements, but the ideas seem promising if paired by better implementations and fine-tuning.

There is a pending problem to be solved, and there are some additional modifications that may help overcome some deficiencies in our methods, which we would like to experiment with in the future.

## References

1. Kang K, Li H, Yan J, Zeng X, Yang B, Xiao T, Zhang C, Wang Z, Wang R, Wang X, Ouyang W. *T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos*. arXiv:1604.02532v3, 2016
2. Deng J, Dong W, Socher R, Li LJ, Li K, Li FF. *ImageNet: A large-scale hierarchical image database*. In CVPR, 2009.
3. Ren S, He K, Girshick R, Sun J. *Faster r-cnn: Towards real-time object detection with region proposal networks*. In NIPS, 2015.
4. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T. *Caffe: Convolutional Architecture for Fast Feature Embedding*. arXiv preprint arXiv:1408.5093, 2014.
5. Ludwig T. *Research Trends in High Performance Parallel Input/Output for Cluster Environments*. Proceedings of the 4th International Scientific and Practical Conference on Programming. Kiev, Ukraine: National Academy of Sciences of Ukraine; 2004.
6. myfavouritekk on GitHub. Available from: <https://github.com/myfavouritekk/matutils>
7. Matlab Python Engine. Mathworks. Available from:  
<http://www.mathworks.com/help/matlab/matlab-engine-for-python.html?refresh=true&requestedDomain=www.mathworks.com>
8. Ouyang W, Wang X, Zeng X, Qiu S, Luo P. *DeepID-Net: Deformable Deep Convolutional Neural Network for Object Detection*. arXiv: 1412.5661v2. 2015.
9. Yang B, Yan J, Lei Z, Li Z. *CRAFT Objects from Images*. In CVPR, 2016
10. Zitnick C.L, Dollar P. *EdgeBox: Locating Object Proposals from Edges*. In ECCV, 2014.
11. Dodge S, Karam L. *Understanding How Image Quality Affects Deep Neural Networks*. arXiv: 1604.04004v2. 2016
12. Zeiler M.D, Fergus R. *Visualizing and Understanding Convolutional Networks*. In arXiv:1311.2901 [cs.CV]

13. Simonyan K, Zisserman A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. In ICLR, 2015.
14. Kalman filter (Wikipedia page): [https://www.wikiwand.com/en/Kalman\\_filter](https://www.wikiwand.com/en/Kalman_filter)
15. Ruder M, Dosovitskiy A, Brox T. *Artistic style transfer for videos*. In arXiv: 1604.08610v2 [cs.CV]
16. Collobert R, Kavukcuoglu K, Farabet C. *Torch7: A Matlab-like Environment for Machine Learning*. BigLearn, NIPS Workshop 2011.
17. Chung J, Zisserman A. *Lip Reading in the Wild*. arXiv:1611.05358 [cs.CV].
18. Farneback, Gunnar. *Fast and accurate motion estimation using orientation tensors and parametric motion models*. Pattern Recognition, 2000. Proceedings. 15th International Conference on. Vol. 1. IEEE, 2000.
19. Gansner E, North S. *An open graph visualization system and its applications to software engineering*. Software – Practice and Experience, 2000, Vol.30, No.11.
20. nngraph: <https://github.com/torch/nngraph>
21. Fischer P, Dosovitskiy A, Ilg E, Hausser P, Hazirbas C, Golkov V. *FlowNet: Learning Optical Flow with Convolutional Networks*. arXiv:1504.06852 [cs.CV]

## Appendix – Results of NMS variants

Weighted Average

	Average & window=7 & remove FP cluster	Average & window size= 7	Average & window size = 5	Average & remove FP clusters (win=5)	Average & remove FP clusters & remove parts (win=5)
Airplane	53.7	48.1	52.6	57.4	58.0
Antelope	13.8	11.0	12.9	16.1	16.3
Bear	16.1	14.6	15.5	17.0	18.2
Bicycle	26.5	22.0	24.8	29.8	31.0
Bird	8.1	6.6	7.5	9.9	10.4
Bus	22.2	19.6	21.2	24.5	27.2
Car	16.8	12.4	15.5	21.6	22.2
Cattle	19.9	17.7	19.4	22.2	23.0
Dog	8.6	7.4	8.3	9.8	10.3
Domestic cat	3.6	3.3	3.6	4.0	4.2
Elephant	28.1	25.3	27.8	30.3	29.5
Fox	25.3	20.9	24.3	29.1	29.4
Giant panda	46.8	43.8	46.3	49.5	49.0
Hamster	7.7	6.5	7.6	9.0	9.4
Horse	28.0	21.4	24.8	31.0	31.7
Lion	0.0	0.1	0.1	0.1	0.1
Lizard	2.5	1.9	2.3	3.4	3.5
Monkey	6.2	4.7	5.6	7.7	7.9
Motorcycle	27.2	20.0	26.1	37.2	39.1
Rabbit	1.5	1.3	1.5	1.7	1.7
Red panda	3.8	3.6	3.9	4.3	4.4
Sheep	12.0	8.7	11.1	15.3	11.7
Snake	1.8	1.7	1.9	2.1	2.4
Squirrel	1.0	1.0	1.0	1.1	1.1
Tiger	20.2	15.2	18.4	24.9	26.1
train	18.0	14.5	17.3	22.0	25.1
turtle	8.5	6.5	8.0	10.3	10.6
watercraft	28.7	24.7	27.1	32.1	33.8
whale	7.6	4.6	6.2	11.6	12.6
zebra	59.8	58.0	59.8	61.1	61.0
Mean	17.5	14.9	16.7	19.9	20.4
	Average & remove FP cluster & window=7	Average & window size= 7	Average & window size = 5	Average & remove FP clusters (win=5)	Average & remove FP clusters & remove parts (win=5)

## NMS

Weights	Plain	1-0.1*offset	Gaussian Sigma = 1	Gaussian Sigma = 2
Airplane	62.6	65.3	65.3	65.5
Antelope	18.8	20.4	20.5	20.5
Bear	19.0	19.9	20.1	20.1
Bicycle	37.9	39.5	39.2	39.3
Bird	16.2	16.7	16.6	16.7
Bus	27.9	29.5	29.8	29.8
Car	31.1	33.5	33.6	33.6
Cattle	26.9	27.5	27.6	27.6
Dog	12.1	13.0	13.1	13.1
Domestic cat	5.3	5.7	5.7	5.7
Elephant	34.6	34.6	34.3	34.4
Fox	33.6	34.1	34.0	34.0
Giant panda	53.5	53.8	54.0	54.1
Hamster	10.6	11.0	11.1	11.1
Horse	40.9	42.4	41.9	42.0
Lion	0.0	0.0	0.0	0.0
Lizard	5.9	6.5	6.6	6.6
Monkey	13.4	13.5	13.3	13.3
Motorcycle	55.4	57.8	57.6	57.7
Rabbit	1.9	2.0	2.0	2.0
Red panda	4.8	5.3	5.5	5.5
Sheep	25.2	26.1	26.0	26.0
Snake	4.0	4.2	4.3	4.3
Squirrel	1.7	1.9	1.9	1.9
Tiger	33.0	34.2	34.1	34.1
train	33.7	34.6	34.5	34.6
turtle	12.2	13.7	13.9	13.9
watercraft	37.4	38.2	37.8	37.9
whale	23.2	25.5	25.5	25.5
zebra	64.6	65.5	65.6	65.6
Mean	24.9	25.9	25.8	25.9