Master of Science in Informatics at Grenoble

Master Mathématiques Informatique - spécialité Informatique

option Graphics, Vision and Robotics (GVR)

Deep learning for Object detection in Videos

Peter Bardawil

An internship report submitted to Ensimag Grenoble INP

for the degree of Master of Science

Under the supervison of:

Nicolas Franco Gonzalez

Xavier Alameda-Pineda

EsoftThings

France - 2020

Abstract

Video Object Detection (VOD) is a surfacing field of Computer Vision that has applications in many areas such as video surveillance, autonomous cars and Human-Computer interaction. However, VOD had not been deeply explored in the past due to the complexity of video data. The relation between sequential images in videos makes it add a new dimension to our Object detection problem and therefore is an important research field to exploit. The purpose of this master's thesis was to find a modern approach to leverage the temporal data present in videos to optimize object detection. We developed a new architecture incorporating special types of neural networks to leverage temporal data present in videos. Moreover, we create a new post-processing technique called Kalman Detection Rescoring that optimizes detections through rescoring (updating the confidence score of detections). In this algorithm, we use accurate detections in previous frames to predict their potential localization in current frame, assign those predictions to current detections and rescore them using averaging. We also propose a variant framework called KDR-Skip that permit to decrease the runtime of the whole framework by limiting the use of the object detector and compensating the accuracy with optimization techniques. The work evaluated with ImageNet Vid and Visdrone Dataset show some promising results when compared to state-of-the-art VOD methods.

Résumé

La détection d'objets vidéo (VOD) est un domaine récent de la vision par ordinateur qui a des applications dans de nombreux domaines tels que la vidéosurveillance, les voitures autonomes et l'interaction homme-ordinateur. Cependant, la VOD n'avait pas été profondément explorée dans le passé en raison de la complexité des données vidéos. La relation entre les images séquentielles dans les vidéos ajoute une nouvelle dimension à notre problème de détection d'objets et constitue donc un domaine de recherche important à exploiter. Le but de ce mémoire de master était de trouver une approche moderne pour exploiter les données temporelles présentes dans les vidéos pour optimiser la détection d'objets. Nous avons développé une nouvelle architecture incorporant des types spéciaux de réseaux de neurones pour exploiter les données temporelles présentes dans les vidéos. De plus, nous créons une nouvelle technique de post-traitement appelée Kalman Detection Rescoring qui optimise les détections grâce à du rescoring (modification de la confidence des détections). Dans cet algorithme, grace aux bonnes détections des images précédentes, on predit leur localisation potentielle dans l'image actuelle, attribue chacune des ces prédictions aux détections actuelles et les rescore en calculant leurs moyenne. Nous proposons également une variante appelée KDR-Skip qui permet de diminuer le temps d'exécution de l'ensemble du framework en limitant l'utilisation du détecteur d'objet et en compensant la précision avec des techniques d'optimisation. Les travaux évalués avec ImageNet Vid et Visdrone Dataset montrent des résultats prometteurs par rapport aux méthodes VOD de l'etat de l'art.

Acknowledgements

I would like to express my deep and sincere gratitude to my research supervisor Nicolas Franco Gonzalez for giving me the opportunity to do research and provide invaluable guidance during this internship. His dynamism, vision, sincerity and motivation have deeply inspired me. I would like to express my very profound gratitude to my parents for all the efforts and support they provided me throughout this journey, without it I wouldn't have achieved any of this.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Statement

People rely on their eyes to behave properly; react when they detect an object approaching towards them, to move towards the right direction or simply to have an understanding of their environment. Vision is therefore crucial to be able to act as human beings and solve their problems. Computers are nowadays trying to reproduce what is easily achievable by humans in a faster way. Therefore, when speaking about computer vision, we replace the living creature for a computational instrument. We can also define computer vision as the process of employing the data present in observed images in order to interpret the surrounding environment.

Object Detection (OD) is a fundamental problem in computer vision and part of a bigger field in this domain called Object recognition. It involves locating the presence of objects with a bounding box and class (type of object) in an image. OD is applied to many areas of computer vision including security, surveillance, autonomous cars, human-computer interaction and many more.

Similarly, Video Object Detection (VOD) aims in resolving exactly the same problems faced in Object Detection, except this field focuses on utilizing the information present in videos and therefore the approach to achieve detection is different. In fact, video processing adds a new dimension to the problem; the temporal dimension. Videos are not only a simple sequence of images, it is rather considered as a sequence of related images that contain some temporal information that can be

leveraged into solving detection problems such as object blurriness and fast motion.

1.2 Overview of the Method

The purpose of this project is to develop a technique to optimize detection in videos by leveraging the temporal data available in the sequence of related images using deep learning techniques. Therefore, different approaches have been undertaken; the first one is to create a deep learning model that integrates special types of Convolutional Neural Networks called Recurrent Neural Networks that would grasp the temporal information present in videos to obtain temporally-aware detections. The second approach called Kalman detection rescoring (KDR) is a post-processing approach that optimizes detections using a prediction framework; We use Kalman filter to create predictions of previous boxes to get an estimate of their current position, then we rescore current detections if their location is close to the predicted ones. The last approach called KDR-Skip takes advantage of the previous rescoring technique to optimize the speed performance of the global detection pipeline while maintaining the accuracy.

1.3 Contributions

- We implement, train and evaluate a deep learning architecture that integrates recurrent neural networks such as Convolutional LSTM and Convolutional GRU to produce temporally-aware feature maps.
- We introduce a novel post-processing technique called Kalman detection rescoring that uses Kalman filter to create predictions of previous accurate detections to optimize current ones.
- We conducted experiments on both ImageNet VID and Visdrone Challenge Dataset and showed that our post-processing approach shows interesting results compared to the state-of-the-art.
- We additionally experiment a variant framework called KDR-Skip that intro-

duces new techniques to optimize the runtime of the object detection pipeline while keeping the detections accurate enough. One of them uses Optical flow to correct bounding boxes localization.

1.4 Thesis Structure

- **Chapter 2:** This chapter provides a general introduction to the concepts used in a Object detection and Video Object detection. The classical Object detection techniques and modern deep learning frameworks are explained. Moreover, some important detection components used throughout the study and state-of-the-art video datasets are presented.

- **Chapter 3:** This chapter guides the reader through all the steps taken to develop the different systems. The implementation of the network and the different components of the post-processing frameworks are thoroughly described.

- **Chapter 4:** This chapter describes the experiments conducted in order to evaluate our model and post-processing frameworks. Moreover, the results are provided and analyzed.

- **Chapter 5:** In this chapter we argue about the contributions of this work, its scope and conclusions. We recommend changes that might lead to accuracy improvements.

Chapter 2

State of the Art

2.1 Object Detection In Images

Object detection is an essential component in many computer vision applications, such as video surveillance (pedestrians, traffic, surgeries), human-computer interaction, robotics or autonomous driving. The goal is to detect instances of semantic objects in static images by accurately estimating their position(object localization) and identity(object classification).

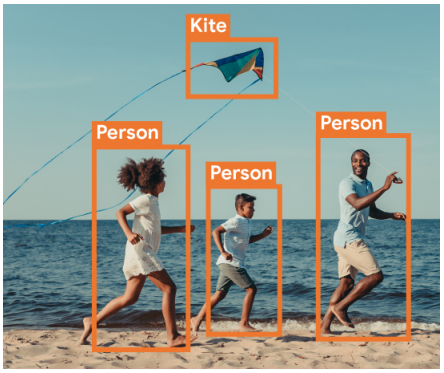

Figure 2.1: Object Detection

In object detection, classical machine learning algorithms can be used to define some features that will be later classified using classification techniques such as SVM or to do regression with regression trees. In the other hand, Deep learning techniques, the main focus in this research, typically based on Convolutional Neural Networks are able to perform end-to-end detection and are usually performing the

best. However two different approaches are widely used, either one-stage methods or two-stage methods, both of them can offer different benefits and drawbacks in term of accuracy and performance.

2.1.1 Two-Stage Methods

Those methods achieves detections on a two-stage manner: First, a region proposal network or a selective search algorithm [32] proposes a set of interesting regions that could possibly surround an object. Following that, a classifier will process those regions to output a set of bounding boxes corresponding to detected objects in the image. R-CNN [11] is the typical two-stage detector where some proposed regions are fed into a deep CNN that outputs features extracted from the image and then classifies and refines the corresponding bounding boxes (Figure 2.5). The problem here is that training this network requires a long time and running it is highly inefficient.

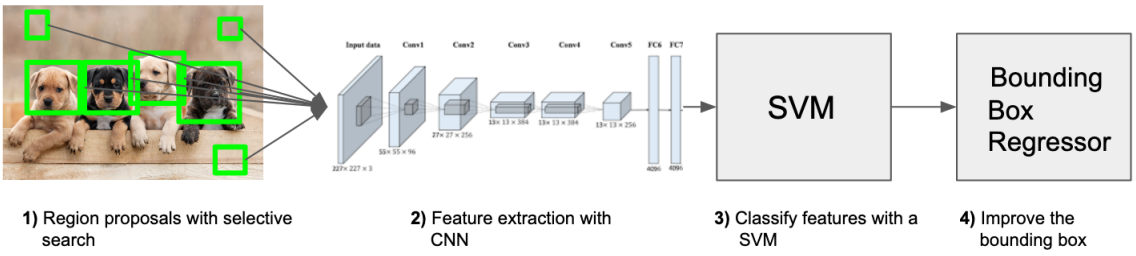

Figure 2.2: R-CNN framework

An improved version Fast-RCNN [10] proposed ROIPooling to speed up the process by allowing to input a full image to the CNN and reshape the output region proposals to a fixed size so that it can be fed to the fully connected layer for classification. Selective search algorithm being a slow and time consuming process affecting the performance of both networks, Faster R-CNN [28] formulated the Region Proposal Network, a separate deep CNN network used to predict region proposals. Another improvement was introduced by R-FCN [6] with the position-sensitive ROI-pooling which improves the detection efficiency by sharing the computation of features in the same region.

2.1.2 One-Stage Methods

A different approach is to skip the region proposal phase and run detection directly over a dense sampling of possible locations. One-stage detectors are usually faster than two-stage detectors while dragging down the precision a bit. A typical detector such as Yolo [27] apply a single neural network to the full image, divide it into regions and predicts bounding boxes and probabilities for each region. This way, predictions are informed by the global environment in the image.
Moreover, during this research, other One-Stage detectors such as SSD [24] and CenterNet [33] were chosen for deeper study due to their highly interesting performance in terms of speed/accuracy trade-off; SSD scores 73.3% mAP running at 46 FPS while CenterNet achieves 75.7% mAP for 100 FPS both on PascalVoc Dataset.

Single Shot Detector

Designed for real time object detection, The SSD uses the pyramidal feature hierarchy of CNN's [18] to efficiently detect objects of different size. First it extracts feature maps at different levels since each level has a different receptive field size thus is only responsible for objects of one particular scale , then it applies convolution filters for each location to make multiple predictions where each is composed of a score for each class and one boundary box. In addition to the convolution layers of the backbone feature extractor (commonly VGG16 [31]) 6 auxiliary convolution layers are added to cover smaller objects (Figure 2.5).

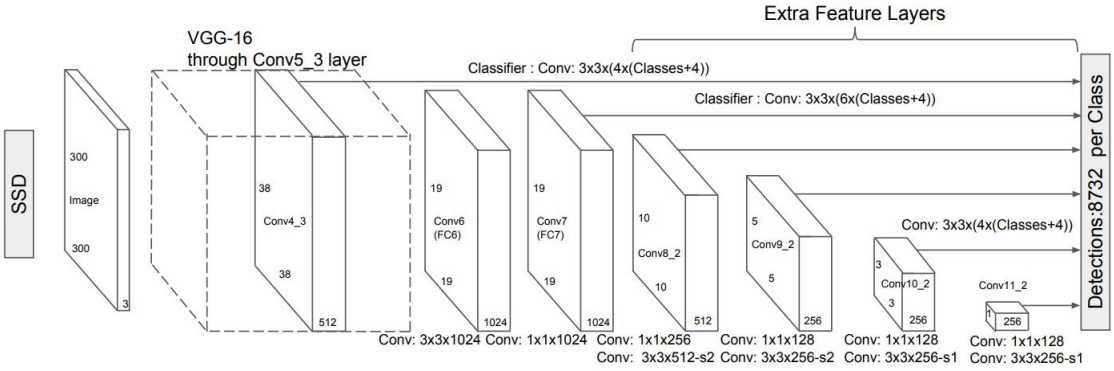

Figure 2.3: SSD architecture

Every level of the network uses Anchor boxes (default boxes) re-scaled to fit the size of objects searched at this scale, e.g for a feature map of size 8x8, anchor boxes of different aspect ratios correspond to small area of the initial input while when considering a feature map of 4x4, the anchor boxes cover larger area. A scale value is defined at each feature map layer starting at 0.2 and increasing to 0.9 for the last layer. Also, aspect ratios are chosen manually leading to a possible computation of the size of those anchor boxes:

$$w = scale \cdot \sqrt{aspect\ ratio} \quad h = \frac{scale}{\sqrt{aspect\ ratio}} \tag{2.1.1}$$

The SSD classifies each prediction as either a positive match or a negative match (positive matches are used to calculate the localization cost). To evaluate that, It calculates the IOU (Intersection over Union) between corresponding anchor boxes and ground truth boxes, if greater than 0.5 than the match is positive and hence the corresponding predicted bounding box is used to calculate the cost. This matching technique incite each prediction to predict shapes closer to the corresponding anchor box. Consequently, the predictions will be more diverse and stable during the training phase.

The objective function used for training is comprised of the localization loss (mismatch between the ground truth box and predicted bounding box) and the confidence loss (loss in making a class prediction)

$$L_{loc}(x, l, g) = \sum_{i \in \{Pos\}}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} smooth_{L1}(l_i^m - \hat{g}_j^m) \tag{2.1.2}$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h \tag{2.1.3}$$

$$\hat{g}_j^w = \log(\frac{g_j^w}{d_i^w}) \quad \hat{g}_j^h = \log(\frac{g_j^h}{d_i^h}) \tag{2.1.4}$$

$$x_{ij}^p = \left\{ \begin{array}{c} \text{1 if IoU} > 0.5 \text{ between anchor box i and ground truth j on class p} \\ \text{0 Otherwise} \end{array} \right\} \tag{2.1.5}$$

Where the localization loss between the predicted box l and the ground truth box g is defined as the smooth L1 loss with cx,cy as the offset to the anchor box d of size (w,h).

For every positive match prediction, the loss is penalized according to the confidence score of the corresponding class. For negative matches, we penalize the loss according to the confidence score of the class '0' which classifies that no object is detected.

$$L_{conf}(x,c) = \sum_{i\in\{Pos\}}^{N} x_{ij}^{p} \log(\hat{c}_i^p) - \sum_{i\in\{Neg\}} \log(\hat{c}_i^0) \quad where \quad \log(\hat{c}_i^p) = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \tag{2.1.6}$$

The confidence loss is calculated as the Softmax loss over multiple classes confidences c (class score) where N is the number of matched Anchor boxes.

The Final Loss function can be computed as :

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \tag{2.1.7}$$

With N the number of positive match and α the weight for the localization loss.

CenterNet

This detector uses a complete different approach where the object is modeled as a point at its bounding box center. Each predicted center is used to find the coordinates and offset of the bounding box. This center prediction problem is considered as a standard key-point estimation problem. After feeding the image to a Fully Convolutional Network [25], a heatmap is generated for different key-points where peaks correspond to predicted object centers.In addition to that, the network predicts a unique bounding box Width and Height for each of these predicted center.

In total three heads are predicted after a forward pass from the network architecture (see figure 2.4); The Heatmap Head for predicting keypoints, the Dimension Head to predict W and H of the bounding box, and the Offset Head used to recover discretization error caused by the donwnsampling of the input.

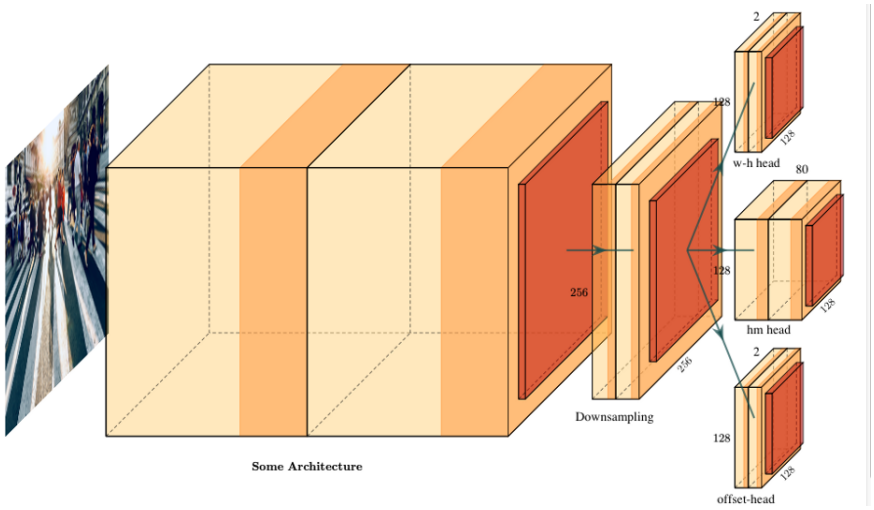

Figure 2.4: CenterNet Detection Head Structure

Consider an Input image I having a Width W and Height H composed of three channels. R being the output Stride and C the number of classes. The aim is to create a Heatmap $\hat{Y} \in [0,1]^{\frac{H}{R} \times \frac{W}{R}}$ where a prediction $\hat{Y}_{x,y,c} = 1$ corresponds to a detected keypoint and $\hat{Y}_{x,y,c} = 0$ to background. They use different networks to predict $\hat{Y}$. For the training of this network, we create a heatmap of the ground truth keypoints $Y \in [0,1]^{\frac{H}{R} \times \frac{W}{R}}$ using a Gaussian Kernel. The training objective (see figure 2.1.8) is a Heatmap Variant Focal Loss [21].

$$L_k = \frac{-1}{N} \sum_{xyc} \begin{cases} (1 - \hat{Y}_{x,y,c})^{\alpha} \log(\hat{Y}_{x,y,c}) & \text{if } Y_{x,y,c} = 1 \\ (1 - Y_{x,y,c})^{\beta} (\hat{Y}_{x,y,c})^{\alpha} \log(1 - \hat{Y}_{x,y,c}) & \text{Otherwise} \end{cases} \tag{2.1.8}$$

Where α and β are hyper parameters of the focal loss and N the number of Keypoints in Image I.

Then a local Offset $\hat{O} \in R^{\frac{W}{R} \times \frac{H}{R} \times 2}$ is predicted for each keypoint. The training objective (see figure [21]) of the Offset Head is an L1 loss .

$$L_{off} = \frac{1}{N} \sum_{p} |\hat{O}_{\tilde{p}} - (\frac{P}{R} - \tilde{p})| \tag{2.1.9}$$

Where p is the Ground Truth keypoint and $\tilde{p}$ a low resolution equivalent ($[\frac{p}{R}]$).

Next, knowing the center point p_k of object k, we regress to the object size $s_k = (x_2^{(k)} - (x_1^{(k)}, (y_2^{(k)} (y_1^{(k)})$ with a prediction $\hat{S} \in R^{\frac{W}{R} \times \frac{H}{R} \times 2}$ (see figure 2.1.10)

$$L_{size} = \frac{1}{N}\sum_{k=1}^{N} |\hat{S}_{p_k} - (s_k)| \tag{2.1.10}$$

The overall training objective of CenterNet is

$$L_{det} = L_k + \lambda_{size} L_{size} + \lambda_{off} L_{off} \tag{2.1.11}$$

With $\lambda_{size} = 0.1$ and $\lambda_{off} = 1$

2.2 Object Detection in Videos

Videos can suffer from blurriness due to the fast motion of objects or the camera. Therefore, sometimes those objects are indistinguishable for normal detectors. Moreover, since the main challenge lies in making use of the rich information present in videos (e.g. temporal consistency) to improve the accuracy and speed of still image detectors, the task of detection of objects in videos can be considered as a different task than object detection in still images Existing work and methods treating this topic can be mainly categorized into three classes : Optical Flow methods that exploits change of intensity of pixels between consecutive images to optimize both accuracy and speed, Recurrent Neural Networks which are special types of CNN that were created to handle temporal data, and Post-processing methods which are extensions to regular object detection pipelines.

2.2.1 Optical Flow in deep learning

Optical Flow estimation is the task of estimating the apparent motion of objects between two frames due to movement of either the camera or the objects in the image. The output is usually a 2D vector field where each vector maps the displacement vector of a pixel from one frame to the next one. Furthermore, there a multiple number of modern deep learning architectures that can leverage this technology to establish correspondences between frames, we will be looking at some of them for having a better understanding of its usability.

Sparse Feature Propagation

The concept proposed in Deep Feature Flow [34] introduces the concept of Sparse Key-frame. The motivation behind that lies in the fact that consecutive frames share similar features and therefore is it unnecessary to compute the features of every frame.

To this end, only Key-frames will enter an expensive feature network N_{feat} while the feature maps of any intermediate frames (Non-key frames) i are propagated from its preceding Key-frame k by per-pixel feature value warping and bi-linear interpolation. The propagation from k to i is represented as $F_{k\to i}$. The pixel-wise motion between frames is estimated by a lightweight Flow Network [8] $N_{flow}(I_k, I_i) = M_{i\to k}$. Finally, the detection network N_{det} works with $F_{k\to i}$ instead of F_i computed from N_{feat}. Since, the computation of N_{flow} is much cheaper than feature extraction with N_{feat}, this method is performs really well in terms of speed/accuracy trade-off.

Dense Feature Aggregation

Another concept introduced in Flow-Guided Feature Aggregation for Video Object Detection [35] is the concept of temporal feature aggregation, the problem that is tackled here is that deep features that could suffer from appearance defects (e.g., motion blur, occlusion) on some frames could be improved by aggregation with nearby frames.

During inference of the network, the features of all frames are extracted with N_{feat}. Also, for any frame i, an aggregated feature map $\hat{F}_i$ is obtained by aggregating the features of its neighborhood:

$$\hat{F}_i = \sum_{k\in[i-r,i+r]} W_{k\to i} \cdot F_{k\to i} \tag{2.2.12}$$

Where $W_{k\to i}$ is the similarity between the feature map propagated from k to i $F_{k\to i}$ and the real feature map F_i, r is the neighborhood window.

This end-to-end network improves the detection by enhancing the features, especially for fast moving objects. Important to note that it slows down the performance due to the overuse of the flow network and feature aggregation.

Those Optical flow methods showed great results in term of accuracy of detections: DFF [34] achieves 73.1% mAP for 22 fps on GPU while the Dense Feature Aggregation [35] achieves 76.3% mAP for 1.3 fps on GPU.However, they are not fast enough to work in an embedded environment. Therefore, those types of networks were not studied further in the scope of my work.

2.2.2 Recurrent Neural Networks

Recurrent neural networks are a type of artificial neural network designed to recognize patterns in sequences of data. In fact, they have a temporal dimension since they take into account time and sequence. In contrast to regular feedforward networks, RNN's have a notion of order in time since as their input they take not just the current example they see, but also what they have previously perceived in time. They have some kind of "memory".

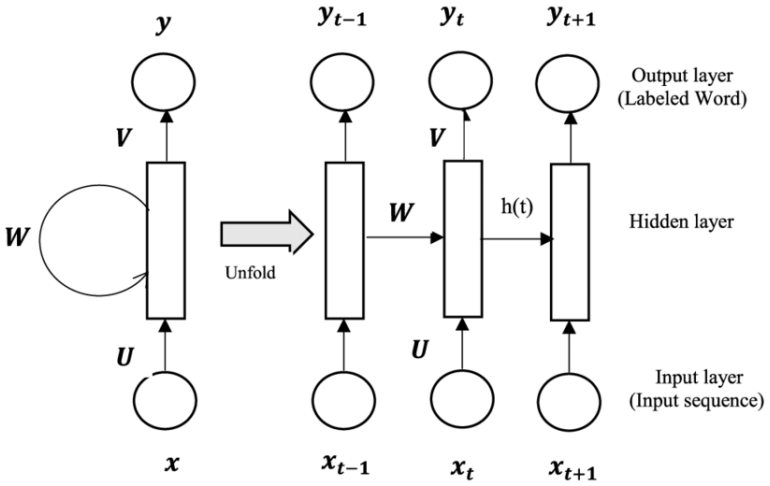

Figure 2.5: Recurrent Neural Network schema

Mathematically, this memory can be described as $h_t = \phi(Wx_t + Uh_{t-1})$ with h_t the hidden state at time step t, x_t the input at time step t, W a weight matrix, h_{t-1} the previous hidden state and U the hidden-state matrix. Those matrices are filters that determine how much importance to give to both the current input and past hidden state. Finally, the function ϕ normalizes the output.

In theory, basic RNN's are able to learn long term dependencies (e.g., the param-

eters that relates information that are temporally distant). However, in practice, they don't seem to be able to achieve that due to various problems introduced in [1]. Fortunately, other types of Recurrent Neural Network such as Long short-term memory networks (LSTM) and Gated recurrent unit networks (GRU) can achieve that according to some current work in this area Looking fast and slow: Memory-guided VOD[23] and Mobile VOD with temporally-aware feature maps [22].

Introduced in 1997 by [14] , LSTM is a special kind of sophisticated RNN that is capable of learning long term dependencies thanks to its complex structure :

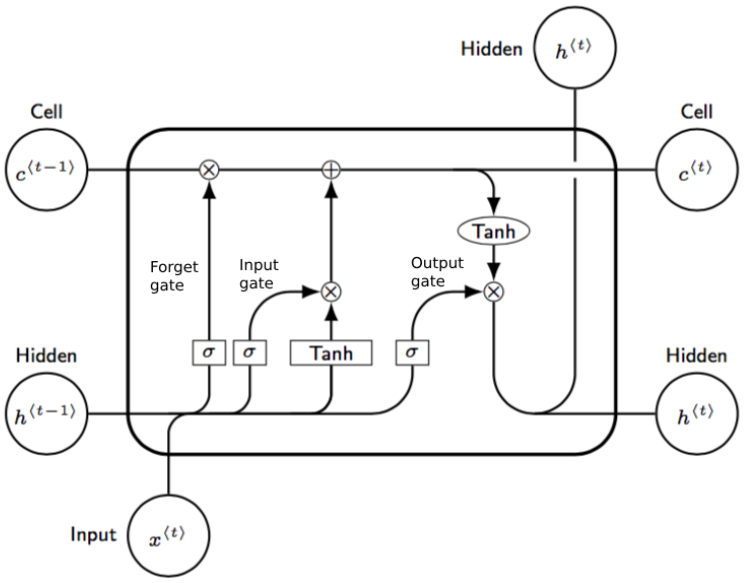

Figure 2.6: LSTM cell structure

The key feature to LSTMs is the Cell state; the upper horizontal line in figure 2.6 which can be represented as the memory of the network. The LSTM has the ability to add of remove information from the cell state regulated by structures called gates. Those gates composed of sigmoid layer and pointwise multiplication operation can decide what information to get through.

An LSTM has three gates, the first one called "Forget gate" decides what is relevant to keep from previous steps. It uses a sigmoid function that outputs a number between 0 or 1 that tells us how much relevant and important is the previous information. Next, the "input gate" decides what new information to store in the cell state. To do so, the previous hidden state and current input goes into a sigmoid function that will decide which values to be updated. In parallel, the previous

hidden state and current input is passed to a tanh function to squeeze the values between -1 and 1. By multiplying both output, the Sigmoid output will decide which information to keep from the tanh output. After updating the Cell state with the first two gates output, the "Output gate" will filter it to decide what information to give to the hidden state (which is also the output of the cell). Moreover, a second type of RNN similar to LSTMs are Gated Recurrent Units(GRU), it has in contrast, two gates instead of three; the reset gate and update gate. It is usually used because it has less parameters(perhaps at the cost of accuracy) and therefore training it is faster.

Additionally, a variant of the LSTM that is relevant to us considering the fact that we are manipulating sequential images is the Convolutional LSTM where matrix multiplication are exchanged with convolution operations to let the data flowing inside the cells keep its 3D input dimension instead of the 1D vector of features.

Temporally-Aware Feature Maps

Introduced in [22] for the first time in the domain of video object detection, the combination of fast single image detector and Convolutional LSTM showed great promises in term of speed and accuracy. The proposed bottleneck Convolutional LSTM variant could propagate features both temporally and across scales by adding multiple convolutional LSTM layers in between the convolutions layers of the SSD detector, achieving a high accuracy improvement and proving the efficiency of LSTMs in this new temporal context.

Memory-Guided Video Object Detection

A second successful example of a work that uses RNNs is proposed by [23] in which they want to exploit the "gist" of a scene (representation of a complex environment in a short period of time) by relying on relevant prior information which is inspired by how humans are able to perceive their environment and recognize objects. To do so, they implemented a visual memory module consisting of an LSTM layer that fuses very different visual features produced by fast and slow feature extractors across frames to generate detections in an online manner. They work promises achieving

state-of-the-art results for mobile video object detection as their system obtains 59 mAP running at 72 fps.

The great potential of RNNs brought to light with those recent work, a deeper study and experimentation was made to get first optimization results.

2.2.3 Post-Processing Methods

A straightforward way of improving detections results of video object detections is to apply methods at the end of an object detection pipeline. This way, a substantial improvement in accuracy can be achieved without sizeable model architecture change or performance loss.

A notable state-of-the-art post-processing method is Sequence Non-Maximal Suppression(Seq-Nms) [12]. This algorithm applies modifications to the detections confidences resulting from the object detector by res-coring objects that had a better confidence in a different step in time. The motivation here is that due to occlusion, motion blur, an object can lose its confidence on certain frames but since in the previous frames, this same object had a good score, it can be boosted.

Seq-NMS takes as input all object bounding boxes and scores for an entire video clip and does a global optimization of the detections. It is applied iteratively. At each iteration it performs three steps:

First, **Sequence Selection** selects the sequence of boxes having the highest sequence score.(see figure 2.7)

Second, **Sequence Re-scoring** applies a function to the scores of previously selected sequence to get a new score for each frame of the sequence.

Third, **Suppression** which suppresses from the previously selected sequence of boxes any box that has a sufficient overlap with another one.

Following the good results of Seq-NMS, with improvements of up to 4% in mAP, the implementation of a post-processing method was also conducted to optimize further detection pipeline results.

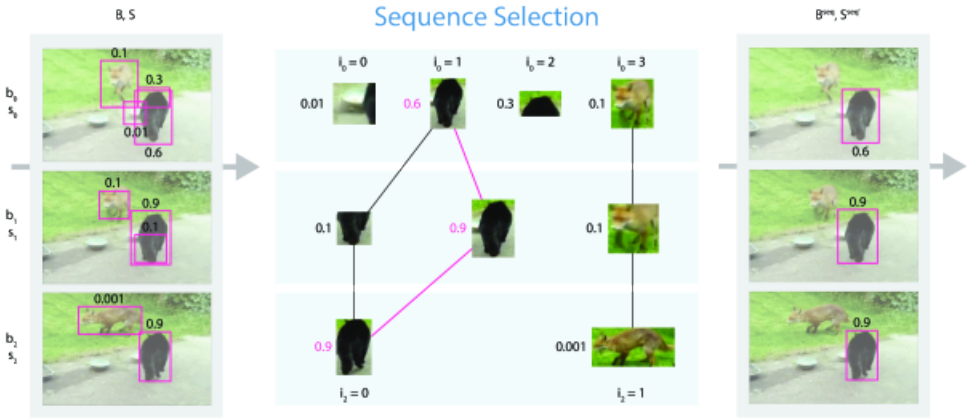

Figure 2.7: Sequence Selection of Seq-Nms

2.3 Detection complements

This section explains some key concepts directly related to video object detection that will be useful to be acquainted with for the understanding of my work.

2.3.1 Optical Flow

Optical flow could be defined as the motion of objects between consecutive frames of sequence, due the relative movement of the object or camera. It is represented as a 2D vector field where each displacement vector represent the movement of a group of pixels a first frame to the next one as seen in Figure 2.8. This information is useful to determine the relationships between consecutive frames. For example, to track the movement of objects in a video.

Figure 2.8: Optical Flow Illustration

The following assumptions are made in order to compute the optical flow :

1. **Brightness Consistency:** Projection of the same point looks the same in every frame.

2. **Spatial Coherence:** Points move like their neighbors.

3. **Small Motion:** Points do not move very far from one frame to another.

The Brightness Constancy Equation may be expressed as:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.3.13}$$

In the Equation 2.3.13 $I(x, y, t)$ is the intensity of a pixel in the frame t. The pixel moves by distance (dx,dy) in the next frame after a time dt. As the intensity does not change and the pixels are the same, we can assumed the affirmation given in Equation 2.3.13. Based on the second assumption about **spatial coherence**, the Taylor series can be developed to get:

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{2.3.14}$$

In a summarized version:

$$f_x u + f_y v + f_t = 0 \tag{2.3.15}$$

where:

$$u = \frac{\partial x}{\partial t}; v = \frac{\partial y}{\partial t} \tag{2.3.16}$$

In Equation 2.3.16, u and v are the x and y components of the velocity also called optical flow of $I(x, y, t)$, and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x, y, t). In Equation 2.3.15, the terms u and v are unknown. There are not enough equations to find u and v. Thus, to find the optical flow, another set of equations is needed. Several methods are provided to solve this problem, and one of them is Lucas-Kanade [17].

Lukas Kanade

Based on the second assumption of the optical flow (the **spatial coherence**), the neighbouring pixels are supposed to have a similar motion. So Lukas-Kanade takes a piece of the image with $n * n$ pixels and establishes that all the $n * n$ points have the same movement. The $n * n$ piece is called the window size. Suppose that we take a window size of $9 * 9$, then the problem will have 9 equations for just two unknowns. The least-square method can then solve this system of equation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{2.3.17}$$

As we assume small motion(Third assumption of Optical Flow), the method fails when there is large motion. A solution for this is to use Lukas-Kanade with pyramids. In the pyramids, small motions are removed and large motions become small motions. So we get optical flow along with the scale. In summary, the optical flow tracks a patch of pixels during a sequence of frames.

Interest Point Detectors

In order to compute the optical flow, it is required to select the points to track. These points are called interest points (IPs), and should have some properties that make their re-identification easy during subsequent frames. Usually, IPs are points in images that are invariant to rotation, translation, intensity and scale (robust and reliable). There are different interest points, such as corners, edges, blobs etc. In addition, different ways computing them. For example, a way to identify corners is that a slight shift in the location will cause a large change in intensity in both horizontal (x) and vertical (y) axes. There are many methods to compute IPs.

Shi Tomasi Corner Detection

Shi Tomasi [16] could be seen as the evolution of Harris Corner Detector [4] due to its better performance and its similar structure. The first step in calculating Shi Tomasi is to determine which windows (small image areas) produce very large

variations in intensity when moved in both (x) and (y) directions (i.e. gradients) of the image. When these windows are determined, a score R for each window is calculated. Finally, if R is greater than a threshold, the window is classified as a corner. Different from Harris Corner Detector, Shi Tomasi computes the score R based on the eigenvalues of a matrix M which represents the weights of the pixels on a window. An in-depth explanation can be found in [16] and [4]. Figure 2.9 shows an example of the results given by different Interest point detectors.

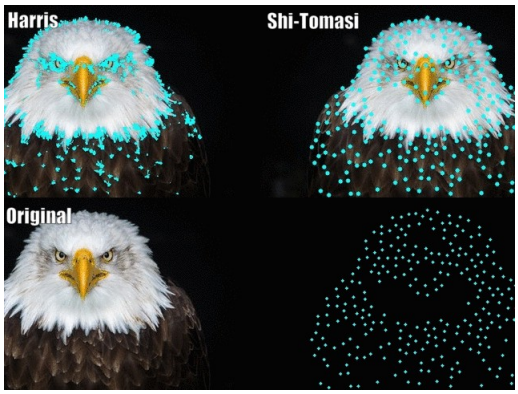

Figure 2.9: Interest point detectors

2.3.2 Intersection Over Union IoU

Intersection over Union is simply an affinity metric. It allows us to compare the position of two bounding boxes. Computing IoU consists of dividing the area of overlap between the bounding boxes by their area of union as shown in Figure 2.10. This gives as result a percentage of overlap between two bounding boxes.

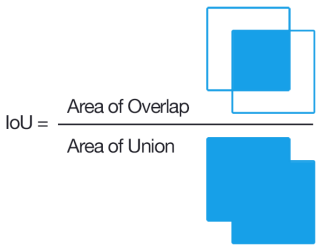

Figure 2.10: Computing IoU

When using IoU as an affinity metric, a high IoU means that both detections are highly likely to represent the same object. An example of different performance

of IoU is found in Figure 2.11.

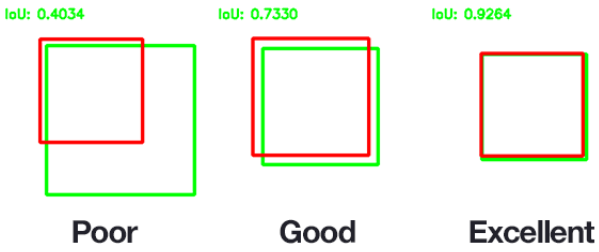

Figure 2.11: IoU examples

2.3.3 Linear assignment with Hungarian's Algorithm

Linear Assignment problems are fundamental combinatorial optimization problems that we will be facing when associating predicted bounding boxes to detected bounding boxes later in the post-processing stage.

The Hungarian method solves the assignment problem. The assignment problem can be stated as follows: imagine that there are n men and n jobs, every man has a cost for performing each job. Assign exactly one worker to each job and exactly one job to each worker in such a way that the total cost of the assignment is minimized.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Figure 2.12: Assignment problem example

Figure 2.12 shows the cost matrix. The green values in this matrix show the optimal jobs assignment. The total cost obtained is the minimum that we could obtain combining the different possibilities. The Hungarian algorithm solve the assignment problem in $O(n^3)$ time. It was proposed by H.W Kuhn in 1995 [19] and improved by J. Munkres in 1997 [26].

2.3.4 Kalman Filter

Kalman filtering is an algorithm that provides estimations of some unknown variables given measurements observed over time. This algorithm have been demonstrating its usefulness in various applications such as robotics, signal processing or econometrics. Kalman filters have relatively simple form and require small computational power, this is why we decided to use it as a key part of our post-processing method. Kalma filter algorithm consists of two stages: prediction and update. In the first stage a mathematical state model is used to make a prediction about the system state. The following equation called **State Prediction** predicts the new state:

$$x_{predicted} = Ax_{n-1} + BU_n \tag{2.3.18}$$

Where A is a State Transition matrix, B the control matrix(used to define linear equations for any control factor) and U_n a Control Vector(indicates the magnitude of any control system's on the situation, it is given as an input)
Then, we predict how much error in our prediction with the **Covariance Prediction**:

$$P_{predicted} = AP_{n-1}A^T + Q \tag{2.3.19}$$

Where Q is the estimated process error Covariance
In the next stage this state prediction is compared to measured state values, we call it the **Innovation**:

$$\hat{y} = Z_n - Hx_{predicted} \tag{2.3.20}$$

Where Z_n is the Measurement Vector which contains the real-word measurement received at this time step and H is an Observation matrix(matrix to translate state vector to measurement vector)
In parallel, we calculate the **Innovation Covariance** which compares the real error against prediction error:

$$S = HP_{predicted}H^T + R \tag{2.3.21}$$

Where R is a constant representing the estimated measurement error covariance
Next, the difference between the predicted and measured state is moderated based on estimated noise and error in the system and measurements, we call it the **Kalman**

Gain:

$$K = \frac{P_{predicted} H^T}{S} \tag{2.3.22}$$

Accordingly, a new state estimation is returned during the **State Update** phase :

$$x_n = x_{predicted} + K\hat{y} \tag{2.3.23}$$

Finally, the **Covariance Update** give us a new estimate of the error:

$$P_n = (I - KH) + P_{predicted} \tag{2.3.24}$$

Where I is the identity matrix

Those output estimations are then used to predict the future state during the next time update, and the cycle begins again. A summary of the flow of Kalman filter's algorithm can be seen in 2.13

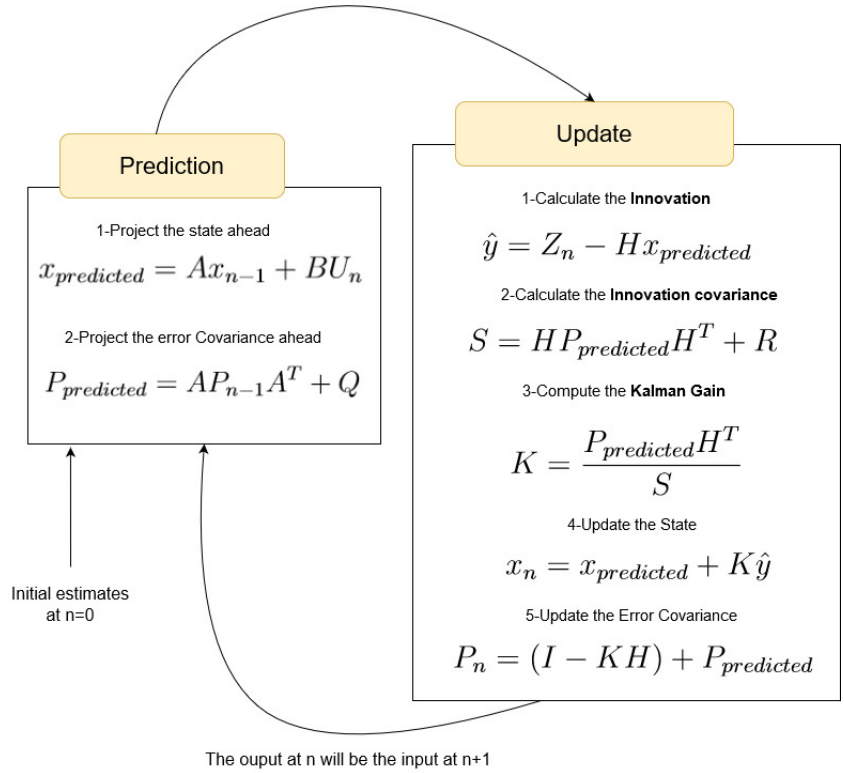

Figure 2.13: Kalman filter's algorithm flowchart

2.4 Object Detection Evaluation

The metric to test the performance of an object detection approach is the **Mean Average Precision (mAP)**. It is well used in benchmarks as [**pascalvoc**] and [**kitti**]. In order to understand the AP, it is necessary to understand the false positives, false negatives, the precision and recall of an object detector:

- **False Positive :** the total number of wrong detections.
- **False Negative :** the total number of missed detections.
- **Recall:** the number of correctly matched detections divide by the total number of detection in the ground truth (see Figure 2.4):

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.4.25}$$

- **Precision:** the number of correctly matched detections divide by total detections in the object detection result (see Figure 2.4):

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.4.26}$$

As a classifier may have multiple classes (cars, pedestrians, cyclist and so on ..) the AP will be computed for each class. The IoU will be used to determine if a predicted bounding box (BB) is True Positives TP , False Positve FP or False Negative FN. For a detection be considered as TP the IoU must be greater than 0.5 and the class must be correct.

Once the precision and recall are calculated for a given class across the test set, the interpolated precision Equation 2.4.27 is calculated, where at each recall level, r, the maximum precision value for that r is taken.

$$P_{interp}(r) = max\ p(\tilde{r}) \tag{2.4.27}$$

Then, the AP is computed by taking the area under the Precision Recall curve. This is done by dividing the recalls evenly to 11 parts, $0, 0.1, 0.2, ..., 0.9, 1$ (see Figure 2.15):

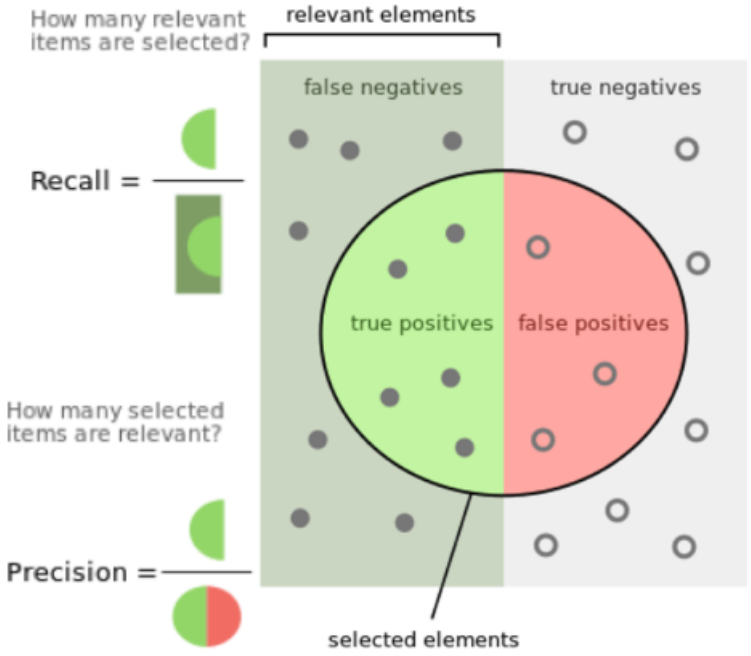

Figure 2.14: Precision and Recall

$$AP = \frac{1}{11} \sum_{r\varepsilon\{0.1,..,1\}} P_{interp}(r) \tag{2.4.28}$$

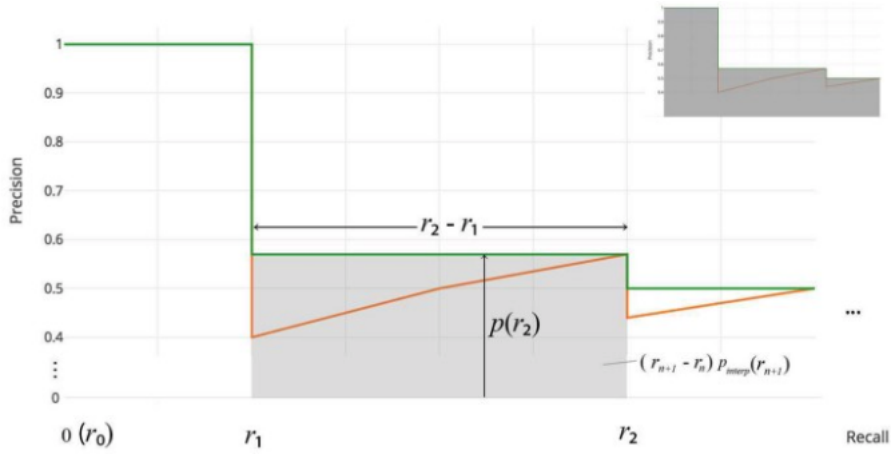

Figure 2.15: PR Curve

Thanks to the use of interpolation precision the impact of "deviations" caused by small variations in the ranking of detections is reduced. In Figure 2.15, the deviation is highlighted in color orange and the precision interpolated in color green. Finally, the map will be the sum of all the AP divided by the number of classes:

$$mAP = \frac{\sum AP_i}{N} \tag{2.4.29}$$

2.5 Benchmark Dataset

Datasets play a crucial role in Objet Detection. They allow training of detectors, evaluating them, while comparing the results with existing methods to estimate state of the art in Object Detection. However, in the domain of Video Object Detection, the data should be sequential and in form of videos. Such datasets are rare but recently more datasets are surfacing while proposing new challenges. Datasets used to evaluate the performance of the work created during this master thesis are ImageNet VID[30] and Visdrone Challenge Dataset[7].

2.5.1 ImageNet VID

ImageNet VID is composed of more than 1 million images divided into 5354 short snippets. There are 30 basic-level categories that were carefully chosen considering different factors such as movement type, level of video clutterness, average number of object instance, and several others.

Figure 2.16: ImageNet VID samples

For each video clip, algorithms will produce a set of annotations (fi,ci,si,bi) of

frame number fi, class labels ci, confidence scores si and bounding boxes bi. This set is expected to contain each instance of each of the 30 object categories at each frame. The evaluation metric is the same as for the object detection task (mAP).

2.5.2 Visdrone Challenge Dataset

Visdrone is a large-scale benchmark with carefully annotated ground-truth for various important computer vision tasks.

It consists of 288 video clips formed by 261,908 frames and 10,209 static images, captured by various drone-mounted cameras, covering a wide range of aspects including location (taken from 14 different cities separated by thousands of kilometers in China), environment (urban and country), objects (pedestrian, vehicles, bicycles, etc.), and density (sparse and crowded scenes).

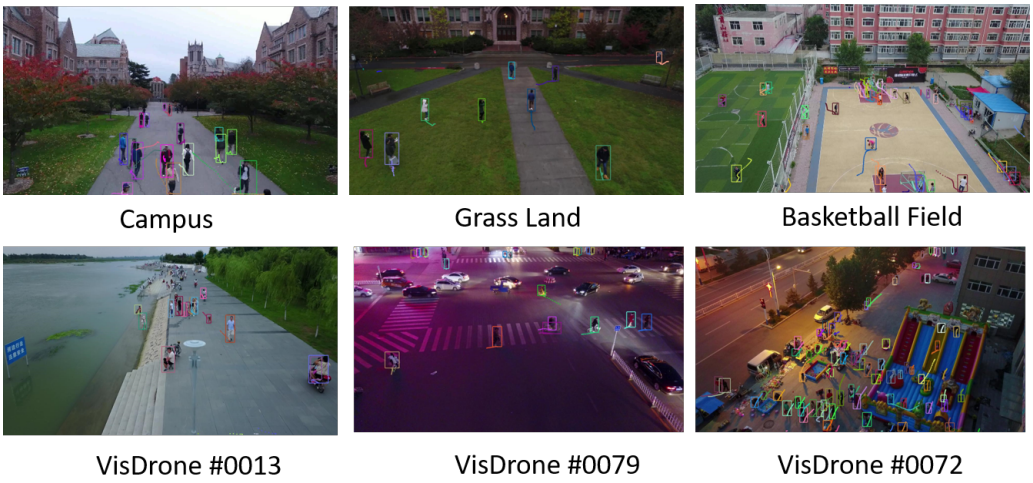

Figure 2.17: Visdrone dataset samples

These frames are manually annotated with more than 2.6 million bounding boxes of targets of frequent interests, such as pedestrians, cars, bicycles, and tricycles. Some important attributes including scene visibility, object class and occlusion, are also provided for better data utilization. The evaluation metric is also the mAP.

Chapter 3

Implementation

3.1 Detection network

The first part of the work is training object detector pipeline that will propagate the features across time thanks to recurrent neural networks. This idea was selected based on the work developed in Mobile Video Object Detection with temporally [22]. This paper as described earlier introduces Convolutional LSTM to the video object detection task. A first task will be to chose a detection pipeline and to train it. Second, An Convolutional LSTM will be implemented and trained. Finally, other variants of RNNs will be trained to have better evaluation material.

3.1.1 Mobile Object Detector

An object detection pipeline can usually be divided into a base network (Backbone network) and a detection network. Backbone network such as MobileNet [15], VGG [31], and ResNet [13] provide high level features for classification or detection. If you use a fully connected layer at the end of this networks, you have a classification. But you can remove the fully connected layer and replace it with detection networks such as SSD [24], Faster R-CNN[28] or CenterNet [33] to generate detections.

I decided to work with a MobileNet SSD since it is known to achieve state-of-the-art results for mobile environment. However, the SSD was slightly modified from its original form by applying separable convolutions (the convolution is divided into depthwise convolution and pointwise convolution to save up some computation

power) instead of normal convolutions. We called it mobileNet-SSD Lite.

The model was implemented in pytorch and the mobilenet was pre-trained on ImageNet Classification.

The mobileNet-SSD Lite was trained with 2 GPUs GEFORCE GTX 2080 on ImageNet VID, with a batch size of 60, a learning rate of 0.003 for 30 epochs. Also, the dataloader was given random images in no sequential manner to have a more effective training since the temporality of the sequences would not get captured by this model.

Due to the large size of the dataset (+1M images), the total training time took approximately 96 hours.

3.1.2 Convolutional LSTM

The training of the Base Network Mobile-SSD-Lite done, a Convolutional LSTM was implemented by taking into account the changes made by [22], in other words, we use a depthwise separable convolution which decreases computation in the gates. As proposed in [22], we remove the conv14 of mobilenet and place the convLSTM just after it and before the first SSD Feature map as seen in fig 3.1.

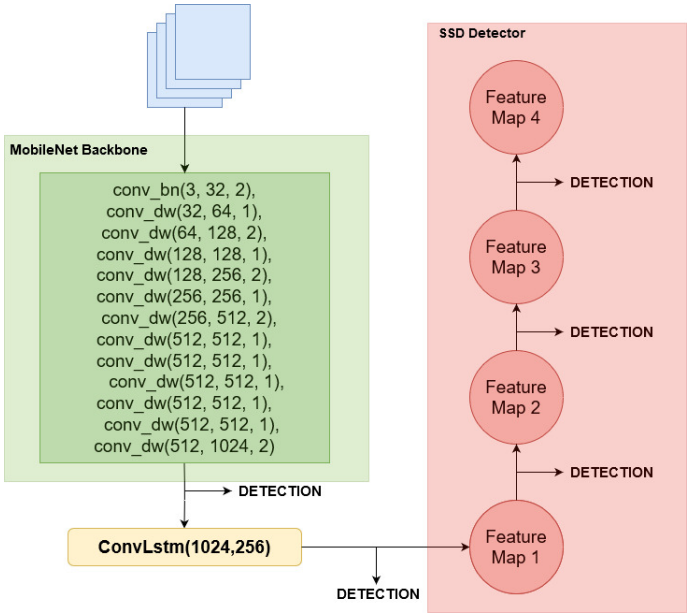

Figure 3.1: Placement of ConvLSTM in model architecture

We setup the training by first freezing all layers up to the newly added convLstm, this way we are choosing the parameters that needs to learn. We assign the learning rate of the parameters of the LSTM to 0.0003 and choose a 10 times smaller learning rate for the SSD since we need the LSTM to learn faster. We run a training on the same 2 GPUs for 50 epochs with a batch size of 10 ,which defines the temporal window and length of each sequence. In contrast to the training of the basenet MobileNet-SSD-Lite [15], the dataloader is given random sequences of consecutive images, for the convLSTM to learn from the features temporality.

3.1.3 Alternative RNNs

Alternatively to the convolutional LSTM, due its smaller sizer, a GRU network [5] was also trained in the same training fashion as the LSTM; We added it between the Backbone network and SSD , trained it for approximately 36 hours with a batch size of 10 , a learning rate of 0.0003 considering 50 epochs.

Moreover, Considering our first results with RNNs that will be reviewed in the next section, and the fact that training those networks was done in a large dataset such as ImageNetVid, we inferred that finetuning it to find the optimal set of weights would take a big amount of time. Consequently, we decided to proceed and work on the post-processing part to get optimized detections.

3.2 Kalman Detection Rescoring

As explained earlier, a post processing algorithm is added at the end of a detector network to boost the detections. But to be able get the best out of original detections, we need to understand why they are not optimal and how we could improve them. In fact, my motivation came after visualizing the detections on ImagenetVID. In some frames, the detector is not able to give a satisfactory detection for objects that were previously detected with an acceptable confidence; it outputs multiple bounding boxes with really low score. However, we can see that some of them could be accepted since they bound the object accurately (see fig 3.2)

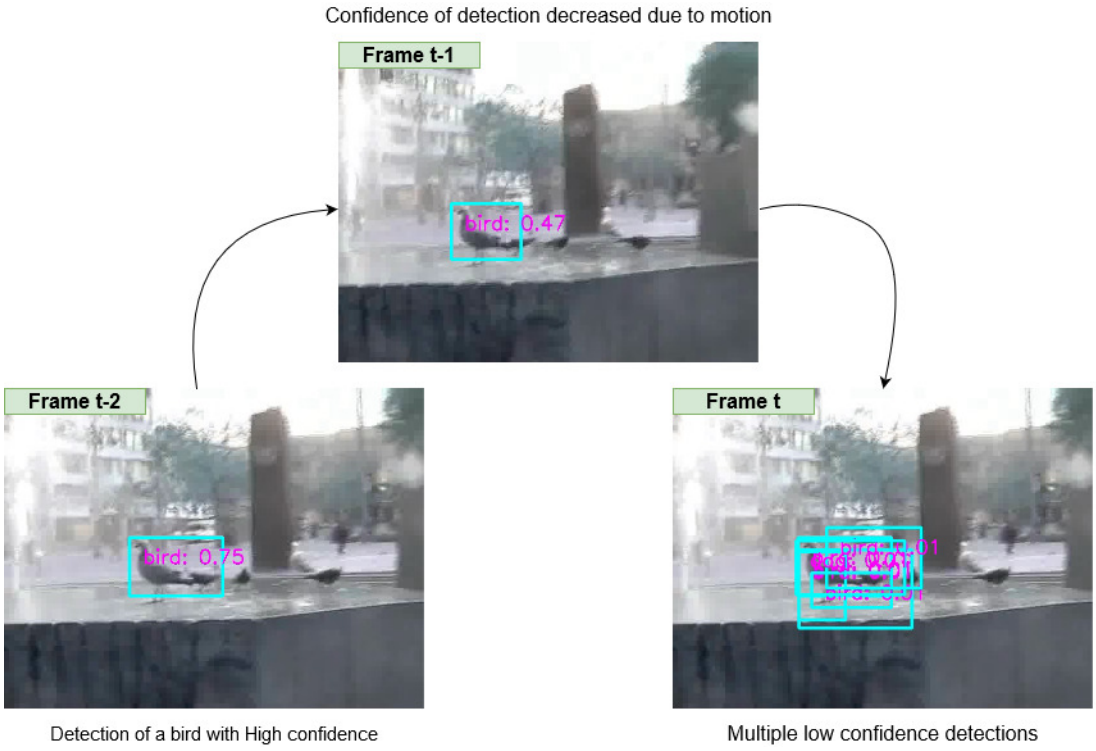

Figure 3.2: Object detection confidence decrease

Hence, it was evident that we needed to take advantage of the good previous detections to update one of those detections with low score. This is where Kalman filter comes to play, we use previous good detections as measurement for the Kalman filter, and predict an estimation of their position in the following frames. Furthermore, we will need to assign this prediction to the closest detection with low confidence and rescore it.

3.2.1 Kalman Filter

As introduced in the previous sections, Kalman takes advantage of accurate previous measurements (in our case detections) to create a new prediction for each object. Based on the kalman filter implemented by Simple Online and Realtime Tracking [2], we create a kalman filter for each detection in the form of tracker, where we update the kalman filter every frame with each new detection.

The kalman filter object is initialized by specifying the size of the state vector with dimension 4 and the size of the measurement vector that we will be using with

dimension 7. We initialize the filter state estimate with the coordinates of the detection. We also have to assign a reasonable initial values to the covariance matrix, the process noise, measurement noise, measurement function, state transition matrix and control transition matrix. For more details about the implementation of Kalman in python, a interesting reference is [20] .

For each frame, we first advance the state vector and return the predicted bounding box estimate for each kalman tracker. This step represent the predict stage(see section 2.3.4). After getting a list of all predictions, we can move on to the assignment part where we need to assign one prediction to one detection for all predictions, this will be explained in the next section.

3.2.2 Data association and rescoring

The prediction obtained thanks to Kalman filtering, we move to the next problem where we need each prediction to be assigned to one detection. A first step is to create an IOU matrix which is a matrix of size *predictions* X *detections*. Each cell of this matrix is filled with the intersection over union of a prediction and detection. Moreover, having now an affinity between the kalman predictions from frame t-1 and the detections t, the problem we have now can be expressed as the linear assignment problem which can be solved with the Hungarian algorithm as explained in section 2.3.4.

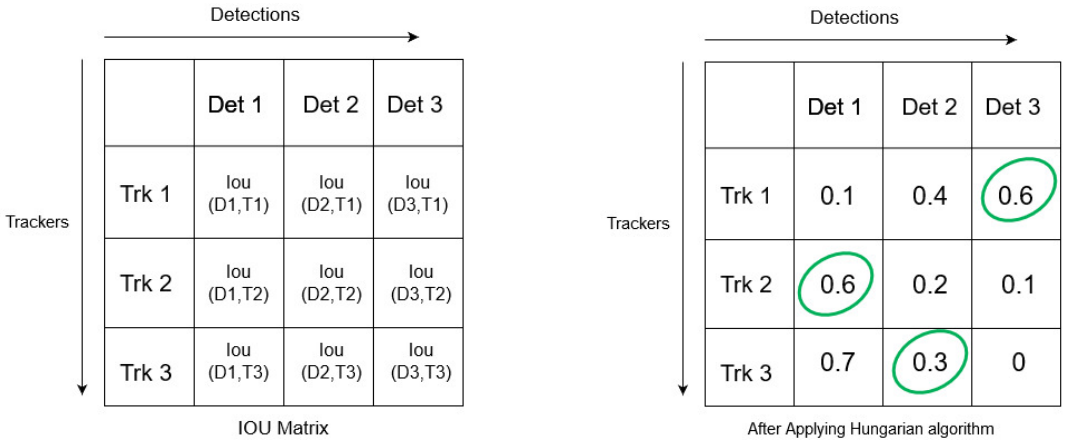

Figure 3.3: Data association

In the figure 3.3, an example of the role of the Hungarian algorithm is shown; in the first row, the green circle infer that prediction Trk1 is best to be assigned to Det3.

The next step is to update all kalman filters with their newly matched detection, this step represent the update stage (see section 2.3.4). More importantly, we can now rescore the detections by averaging current score and previous scores of the same object in precedent frames. For example, if an object at t had a confidence of 0.7, at $t+1$ a confidence of 0.45 and $t+2$ a confidence of 0.15, rescoring will change the last value to $\frac{0.7+0.45+0.15}{3} = 0.43$.

Another important step in the algorithm is to create a new kalman filter tracker to every detection higher than a threshold for future predictions.

Below is an overview of the Kalman Detection Rescoring algorithm :

Algorithm 1: Kalman Detection Rescoring

Input: Detections (BBox ,Score,Class)

Output: Updated Detections(BBox,Updated Scores,Class)

1 Predict new locations with Kalman filter
2 Compute IOU matrix between predicted locations and detections
3 Compute Hungarian algorithm of IOU matrix
4 Update Kalman Filters with new matched detections
5 Rescore detections by averaging with previous scores
6 **for** *detection with confidence > threshold (default=0.5)* **do**
7 Create new Kalman filter tracker for detection
8 Return New updated detections

Some parameters can play an important role in getting an optimal rescoring; min_hits tell us the minimum number of times a kalman tracker need to be matched with a detection. Also, the IOU_threshold gives the minimum IOU value allowed when filling the IOU matrix. Finally, the max_age describes the lifespan of a Kalman tracker.

3.3 KDR skip Variant

Further work looked into improving the speed performance of the whole detection pipeline while possibly obtaining same quality of detections or even better.
Therefore, we imagined a pipeline where only two frames over three would pass through the detector, while the last one could be either copied (with or without improvement) or predicted.
The figure 3.4 illustrates a first skip approach where the first two frames pass through the detector then improved with Kalman Detection Rescoring while the last frame is skipped. Its output is just a copy of the previous improved detections.

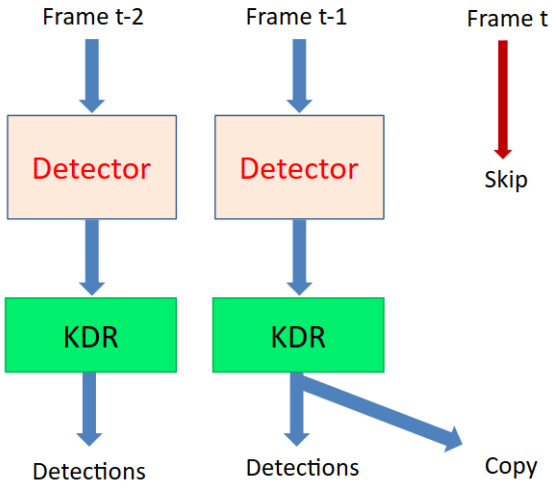

Figure 3.4: KDR Copy Skip approach

3.3.1 Kalman predictions

In attempt to improve even better the detection accuracy since a simple copy is obviously not the most optimal, a next approach was to use Kalman filter to create new predictions by advancing the State vector. However, for the update stage, we use previous rescored detections since detections for the current time step are not available. The figure 3.5 depicts the above.

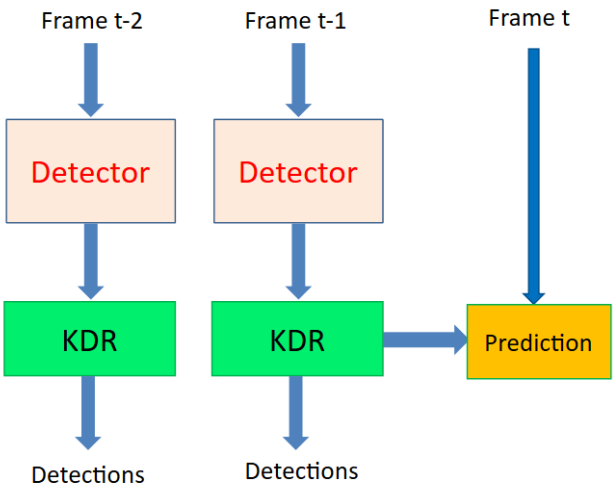

Figure 3.5: KDR Prediction Skip approach

3.3.2 Optical Flow optimizer

An alternative to Kalman predictions when skipping a frame was to use optical flow to compute the motion flow for each object between frame $t-1$ and t. This motion could be applied to detections of the frame $t-1$ to get updated detections for the next frame t.

The first step is to find Interests points (IP), the points that we need to track around each object.

Shi Tomasi Corner Detection:

For computing IPs we decided to use Shi Tomasi [16]: Using OpenCV we need to choose the parameters: **Max Corner** - Maximum number of corners to output. **Quality level** - minimal accepted quality of image corners. **Minimun distance** - minimum accepted Euclidean distance between the returned corners. **Mask** - region of interest. **Block size** - Size of an average block to compute a derivative covariance matrix over each pixel neighborhood. The mask was created based on the detections. The parameters selected were:

$$\left\{\begin{array}{c} maxCorners = 100 \\ qualityLevel = 0.3 \\ minDistance = 5 \\ blockSize = 5 \end{array}\right\}$$

Lukas Kanade

Once the Interest point are selected, we use Lukas-Kanade optical flow from [3] to start tracking those points. The parameters used to initialize the system are:

- **Win Size:** As stated in the previous section, Lukas-Kanade tracks a $n * n$ patch of pixel called window size. This will be the size of our search window for each pyramid level.
- **Max Level:** In the state of the art, we saw that in the pyramids, small motions are removed and large motions become small motions. This helps the system follow interest points when they present large movements. The max Level parameter define the number of pyramids. If Set to 0, pyramids are not used (single level). If set to 1, two levels are used, etc.
- **Criteria**: This parameter defines when the iterative search algorithm will end. There are two criterias: -when the maximum number of iteration is reached and - when the search window moves by less than epsilon.

Different selections of parameters were considered. After making heuristic evaluation, the parameters that gave us the best results were:

$$\left\{\begin{array}{c} winSize = (15, 15) \\ maxLevel = 2 \\ criteria\_count = 10 \\ criteria\_epsilon = 0.03 \end{array}\right\}$$

Once Lucas-Kanade applied , we get a motion vector for each IP. For each object in the frame t, we compute the average motion vector and simply add this vector to

its previous frame bounding box coordinates to get an updated detection box. The whole process can be visualized in figure 3.7

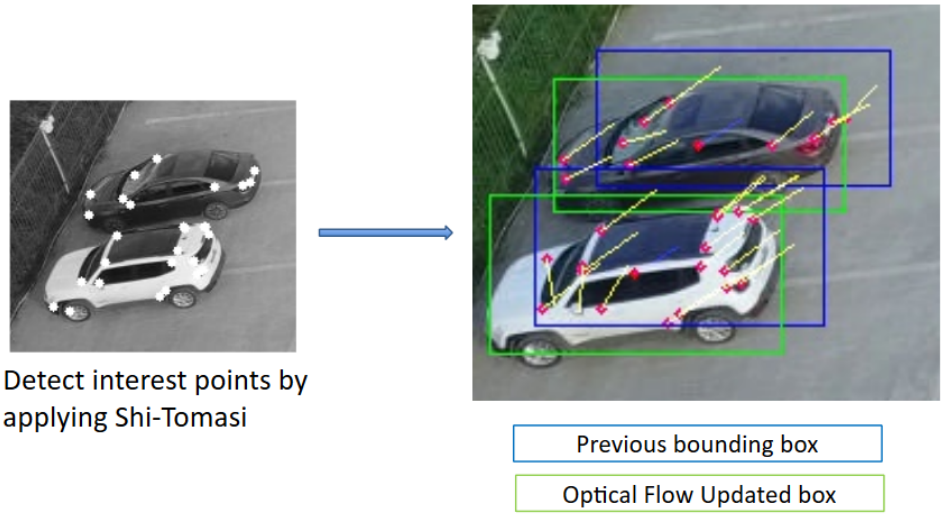

Figure 3.6: Optical Flow update process

Finally, a last approach was to combine both Kalman predictions and Optical flow updated detections to get the best out of our detection pipeline when skipping one frame's detection. The motivation here is that due to fast and sudden motion of an object, the Kalman prediction is much less accurate that an optical flow updated detection.

Therefore, we conditioned the framework to use Optical flow only when we detect a fast motion(average horizontal motion or vertical motion > 6 pixels). This way, we can benefit from both kalman predictions and Optical flow detections.

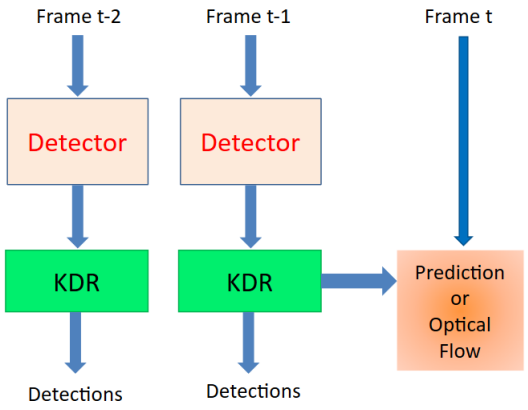

Figure 3.7: KDR Fusion skip approach

Chapter 4

Experiments and Validations

To evaluate my work, I have experimented my implementations using two different Object detectors, MobileNet-SSD Lite and CenterNet, which the architecture was explained in the previous sections. Also, my algorithms were tested on the validation set of both Imagenet Vid and Visdrone Challenge dataset.

4.0.1 RNN evaluation

Training the recurrent neural networks was not straightforward, we had to experiment different parameters (learning rate, epochs, batch size, type of optimizer, choosing what parameters to freeze, etc) to get an acceptable model. Moreover, the difficulty of training a RNN with a database of this size (more than 1 million frames for ImagenetVid) makes it even more complicated for the model to converge for the limited time at our disposal.
The evaluation step consist of comparing the detections obtained with our different models to the ground truth annotations given. More details about the evaluation method was introduced in Section 2.4. Also, it is important to mention that our experiments were made on a CPU machine.

First, the basenet MobileNet-SSD-Lite was evaluated at 24 FPS with a score of 38% mAP. In contrast, the Convolutional LSTM didn't achieve better result globally and was evaluated at 14 FPS with a mAP of 37.5 %. However, we have noticed that the score of some classes have improved in comparison with the basenet (+2% mAP

for Horse class, +1% mAP for train class); this indicated that the Convolutional LSTM learned and successfully made use of the temporal dimension of the input frames. This also shows that it needs more time and finetuning perhaps to be able to achieve that globally for all the classes.

Second, we also conducted the same evaluation process with the trained Convolutional GRU. We noticed the same anomalies when training the network and therefore the global Average precision did not improve compared to the basenet (32& mAP for 22 FPS). Similarly to the Convolutional LSTM, some per-class improvement show the positive impact of Recurrent Neural Nets in terms of temporal data processing.

A last experiment was performed during inference where we aggregate the feature maps from previous frames according to some weights. This method only improved the accuracy by 0.1% as it can be seen in figure 4.1.

Technique	Description	Results
Basenet	MobileNet-SSD-Lite	24 fps with mAP 0.38
ConvLSTM Layer	-Use of Depthwise Separable convolutions and ReLU	14 fps with mAP 0.375
ConvGRU Layer	-Two gates(reset and update) instead of three -Less computationally expensive	22 fps with mAP 0.32
Feature Weight Averaging	-Aggregate Feature maps from previous frame using weight	24fps with mAP 0.381

Figure 4.1: RNN evaluation table (results given on CPU)

Following the unsatisfactory first results of Recurrent Neural Network layers and due the long training time of this type of network considering the length of my internship we decided to move on to the post-processing part of the detection pipeline where most of my contribution and efforts reside.

4.0.2 Kalman Detection Rescoring evaluation

A first evaluation of KDR was conducted on Imagenet Vid; we first train a simple mobileNet SSD-Lite and compare the results with and without our post-processing algorithm. The following table show the result of our KDR implementation with different parameters for the Kalman filter in contrast to the basenet .

Method	mAP	FPS
Mobilenet-SSD lite	0.381	24
Mobilenet-SSD lite + KDR (iou_thresh = 0.5 min_hits=3)	0.405	23
Mobilenet-SSD lite + KDR (iou_thresh = 0.6 min_hits=3)	0.406	23
Mobilenet-SSD lite + KDR (iou_thresh = 0.6 min_hits=1)	0.410	23

Figure 4.2: KDR on ImagenetVid

Adding the Kalman detection rescoring improves by 3% the mean Average precision of our detection with only a 1 FPS cost. A quick look at the per-class scores (see figure 4.3) indicate an improvement for 67% of them. Some classes were improved by as much as 35% (Horse class).

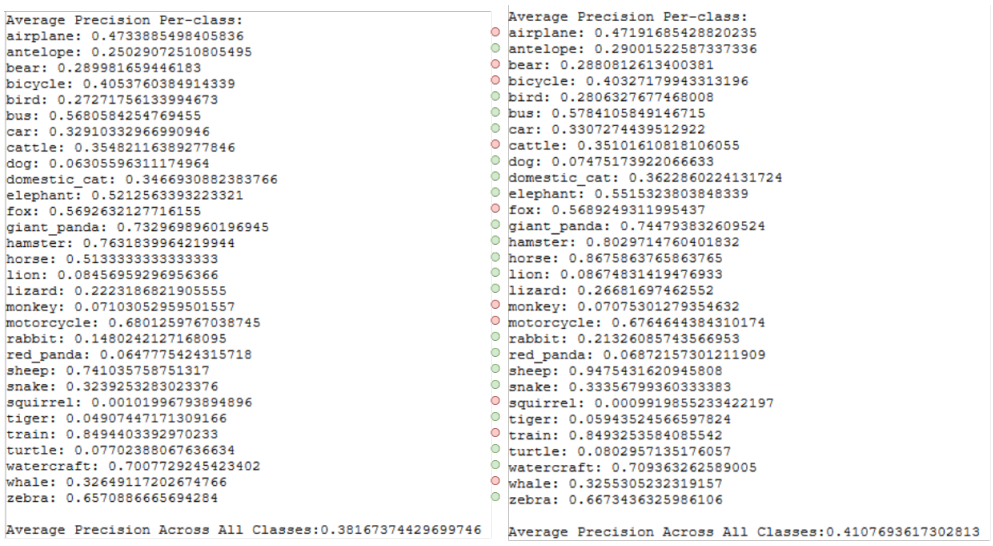

Figure 4.3: Per-class look at the KDR evaluation

Additionally, we evaluated KDR when added to another detection pipeline, CenterNet, which gives much better initial detections. We also use the Visdrone dataset that regroups pictures with an important number of objects in each of them taken from a long distant drone. Having a large number of objects in this dataset compared to ImageNet VID, we can have a better look at the efficiency and accuracy when the environment is much more complex.

Figure 4.4: Rescoring with KDR on Visdrone Dataset

The above image taken from the validation set of Visdrone Dataset is fed into our Rescoring pipeline where dark blue bounding boxes represent improved detections.

Moreover, we manage to achieve a 1.8% improvement in mAP going from 33.8% mAP for the basenet CenterNet to 35.6% mAP after applying KDR. The cost here is heavier, since applying the post-processing tool takes additional 30 ms in average. This is mainly due to the fact that our KDR keeps track of all the detections and since we have much more objects in this dataset, we will have more kalman filters, and the computation of the IOU matrix will then affect the overall runtime. Also, this difference in terms of improvement between the two datasets can be related to the fact that we have more wrong detections in Visdrone and thus they will also be

interpolated and rescored dragging down the accuracy.

Figure 4.5 shows the trade off between number of objects tracked and the runtime of KDR.

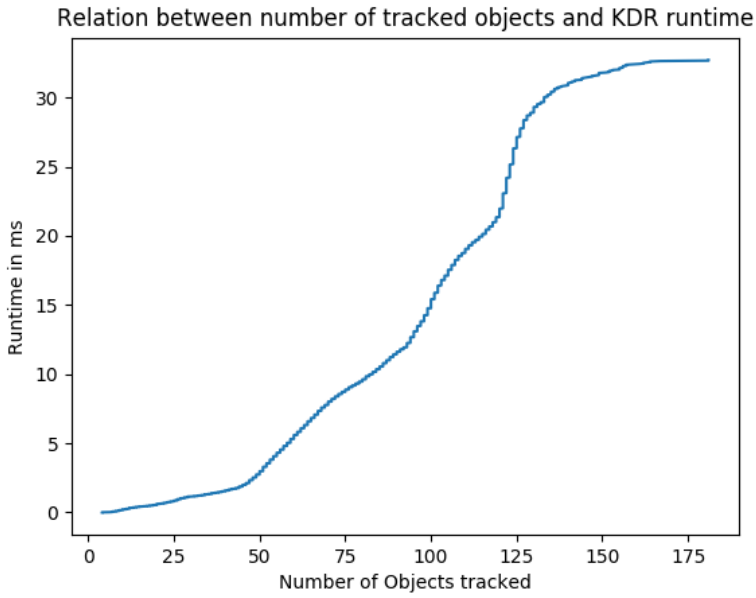

Figure 4.5: Rescoring with KDR on Visdrone Dataset

The following table shows a quick comparison between KDR and State-of-the-art method Seq-NMS for post-processing on ImageNet Vid

BaseNet	Post-Processing	+mAP	+Runtime
VGG Net	SeqNMS	4%	Non-Realtime
MobileNet SSD-Lite	KDR	3%	2 ms

Figure 4.6: Comparison between Seq-NMS and KDR methods

4.0.3 KDR-skip evaluation

The evaluation of KDR-skip was conducted on Visdrone dataset with the goal to understand the contribution of each component of the skip framework.

Initially, the first experiment was to apply the detection pipeline for 2 consecutive frames and then use a copy of the detections of the previous frame. As expected,

the performance gain is of 33% for a decreased precision since we use the detection network only on 66% of the frames, going from 33.8% mAP to 32.4% mAP.

Next, we just add the KDR component after each of the first two frames detections to improve them while still giving a copy of the updated detections to the third frame. We obtain 34.2% mAP already achieving better than the basenet CenterNet(Non-improved detection for every frame) which scored 33.8% mAP. As evaluated earlier, the runtime of KDR per frame on this dataset is on average 30ms which in our whole skip pipeline (3 frames) means 60 ms. Those additional 60 ms costs less than a detection inference with CenterNet which runs at 14 FPS with the hourglass backbone [33] and therefore our goal is already obtained.

Similarly, the skip using Kalman predictions instead of simple copy show a similar score of 34.2% with almost no performance cost since we only advance the state vector. Unexpectedly, using Optical flow for the third frame to update the previous frame boxes, we got an mAP of 32.7% for an average runtime of 16 ms, but the visualization of the updated optical flow boxes (green bounding boxes in figure 4.7) show that the optical flow has a positive impact only when the motion is fast and the predictions/copies are too inaccurate.

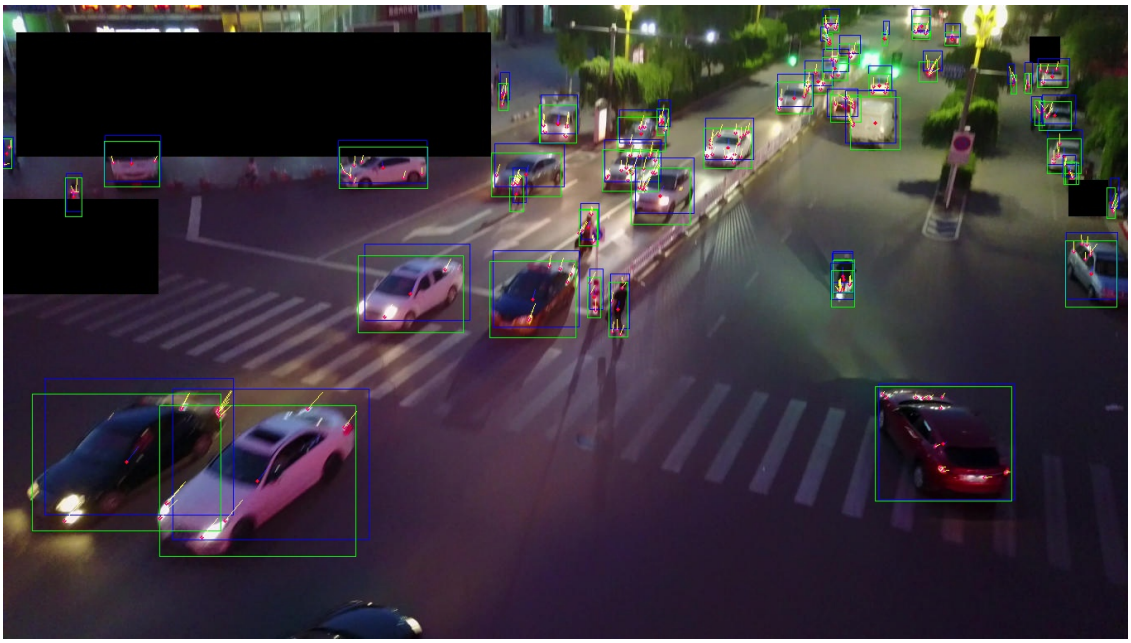

Figure 4.7: KDR-skip with Optical Flow improvement

Consequently, the last approach called KDR Fusion skip which goal was to use Optical flow to update detections only during fast motion was evaluated to 34.5% mAP achieving the best score for this KDR-skip framework for an average cost less than 26 ms/frame (since we replace some KDR prediction by optical flow estimation on certain frames) which is still less than a CenterNet inference.

A brief summary of our KDR-skip framework experiments is presented in figure 4.8.

Method	mAP	Post-Processing runtime per frame
CenterNet	0.338	0
2 CenterNet + KDR	0.351	30 ms
2 CenterNet/1 copy	0.324	0
2 CenterNet+KDR/1 copy	0.342	20 ms
2 CenterNet+KDR/1 KDR	0.342	20 ms
2 CenterNet+KDR/1 OF	0.327	26 ms
2 CenterNet+KDR/1 KDR+OF	0.345	<26 ms

Figure 4.8: KDR-skip evaluation results on Visdrone Dataset

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This research was motivated for the challenge of video object detection. We aimed to create a modern framework that would leverage the temporal data present in videos to optimize its object detection.
The different systems presented are :

- A deep learning model that incorporates recurrent neural networks (ConvLSTM and ConvGRU) to obtain temporally-aware feature maps
- A post-processing algorithm we called Kalman Detection Rescoring (KDR) that allows to rescore detections that are linked to Kalman filter predictions of previous frames detections.
- A Video Object Detection framework we called KDR-Skip that permit to skip object detection every third frame to gain speed while maintaining accuracy.

Our deep learning model was tested on ImageNet VID and showed no global improvement in mAP. However, it showed the potential of adding a Recurrent Neural Network in terms of learning temporal features. This was proven by comparing the average confidence score of its classes to the basenet mobileNet-SSD-Lite where we have noticed some improvements.

With our post-processing algorithm KDR, we prove that we could use the detections made in previous frames to predict their potential location in the next frames using a Kalman Filter. We could then assign those predictions to the current detections and rescore them. When evaluated on ImageNet VID and Visdrone challenge dataset, it showed an improvement of 3% in mAP when added to the basenet mobileNet-SSD-Lite at the cost of an additional 2 ms (ImageNet Vid) and an improvement of 1.8% in mAP when extending CenterNet for a heavier cost of 30 ms. However, we prove that this cost heavily depends on the number of objects tracked with our KDR algorithm. The fewer the number of objects in a dataset, the fewer the cost of our post-processing algorithm.

The proposed KDR-Skip framework allows us to skip the use of an object detection network every third frame to economize performance without loss of accuracy. We compare different variants to determine best speed/accuracy trade off and conclude that using optical flow (for the skipped frame) when the motion is fast instead of Kalman predictions is the best option. Nevertheless, using a simple copy of the updated detections for the third frame could be also considered a good option since we improve the accuracy with no cost for the last frame.

5.2 Future Work

There are several improvements that can be incorporated as:

- Due to the large size of ImageNet Vid and the proven potential of Recurrent Neural Networks , it would be wise to take more time to train the network and fine-tune the parameters. Additionally, choosing a different Object detector could lead to better and faster convergence.
- Concerning the Kalman Detection rescoring, we should test it on even more object detectors and different datasets to find the optimal default parameters (minimum hits, max age, IOU threshold...). This way we could also optimize the speed/accuracy trade-off.

- The prediction phase of the KDR-skip (for the third frame) can be improved by using Optical Flow motion to update the Kalman filter during the Update phase instead of using the simple copy of previous frame detections.

- Also, when computing the optical flow estimation, we could use a faster Interest point detector such as FAST Corner detector [29] instead of shi-tomasi [16]. We could also remove outliers by using RANSAC algorithm [9] that would select only inliers when computing the average motion of each object.

Bibliography

[1] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[2] Alex Bewley et al. "Simple Online and Realtime Tracking". In: *2016 IEEE International Conference on Image Processing (ICIP)* (Sept. 2016), pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003. arXiv: 1602.00763. URL: http://arxiv.org/abs/1602.00763 (visited on 07/29/2019).

[3] G. Bradski and A. Kaehler. "Learning OpenCV Computer vision with the OpenCV library". In: *O Reilly Media Inc* (2008).

[4] Harris C. and Stephens M. "A combined corner and edge detector". In: *4th Alvey Vision Conference* (1988).

[5] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[6] Jifeng Dai et al. "R-FCN: Object Detection via Region-based Fully Convolutional Networks". In: *CoRR* abs/1605.06409 (2016). arXiv: 1605.06409. URL: http://arxiv.org/abs/1605.06409.

[7] D. Du et al. "VisDrone-DET2019: The Vision Meets Drone Object Detection in Image Challenge Results". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. 2019, pp. 213–226.

[8] Philipp Fischer et al. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *CoRR* abs/1504.06852 (2015). arXiv: 1504.06852. URL: http://arxiv.org/abs/1504.06852.

[9] M. Fischler and R. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395. URL: /brokenurl#%20http://publication.wilsonwong.me/load.php?id=233282275.

[10] Ross Girshick. "Fast R-CNN". In: *arXiv:1504.08083 [cs]* (Apr. 30, 2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083 (visited on 08/13/2019).

[11] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[12] Wei Han et al. "Seq-NMS for Video Object Detection". In: *CoRR* abs/1602.08465 (2016). arXiv: 1602.08465. URL: http://arxiv.org/abs/1602.08465.

[13] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 07/22/2019).

[14] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[15] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[16] Shi J. and Tomasi C. "Good features to track". In: *9 th IEEE Conference on Computer Vision and Pattern Recognition, Springer* (1994).

[17] Lucas Kanade. "An iterative image registration technique with an application to stereo vision". In: (1981).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[19] H. KUHN. "The hungarian method for the assignment problem". In: (1955).

[20] Roger Labbe. *Kalman and Bayesian Filters in Python*.

[21] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[22] Mason Liu and Menglong Zhu. "Mobile Video Object Detection with Temporally-Aware Feature Maps". In: *CoRR* abs/1711.06368 (2017). arXiv: 1711.06368. URL: http://arxiv.org/abs/1711.06368.

[23] Mason Liu et al. "Looking Fast and Slow: Memory-Guided Mobile Video Object Detection". In: *CoRR* abs/1903.10172 (2019). arXiv: 1903.10172. URL: http://arxiv.org/abs/1903.10172.

[24] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *arXiv:1512.02325 [cs]* 9905 (2016), pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325 (visited on 07/22/2019).

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[26] J. Munkres. "Algorithms for the assignment and transportation problems". In: (1957).

[27] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[28] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv:1506.01497 [cs]* (June 4, 2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497 (visited on 07/22/2019).

[29] Edward Rosten, Reid Porter, and Tom Drummond. "FASTER and better: A machine learning approach to corner detection". In: *IEEE Trans. Pattern Analysis and Machine Intelligence* 32 (2010), pp. 105–119. URL: http://lanl.arXiv.org/pdf/0810.2434.

[30] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[31] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[32] Jasper Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[33] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. "Objects as Points". In: *CoRR* abs/1904.07850 (2019). arXiv: 1904.07850. URL: http://arxiv.org/abs/1904.07850.

[34] Xizhou Zhu et al. "Deep Feature Flow for Video Recognition". In: *CoRR* abs/1611.07715 (2016). arXiv: 1611.07715. URL: http://arxiv.org/abs/1611.07715.

[35] Xizhou Zhu et al. "Flow-Guided Feature Aggregation for Video Object Detection". In: *CoRR* abs/1703.10025 (2017). arXiv: 1703.10025. URL: http://arxiv.org/abs/1703.10025.