Practical Scheduling of Computer Vision Functions

Adrien Chan-Hon-Tong and Stéphane Herbin
ONERA, Palaiseau, France

Keywords: Time Constraint Classification of Images, Dynamic Scheduling, Robotic Application.

Abstract: Plug and play scheduler adapted to computer vision context could boost the development of robotic platform embedding large variety of computer vision functions. In this paper, we make a step toward such scheduler by offering a framework, particularly adapted to time constraint image classification. The relevancy of our framework is established by experimentations on real life computer vision datasets and scenarios.

1 INTRODUCTION

Computer vision focuses on developing functions which tackle precise tasks like deblurring (Shan et al., 2008), registration (Lucas et al., 1981), segmentation (Zhang, 1996), classification (Lazebnik et al., 2006), pixelwise classification (Shotton et al., 2006), object detection (Dalal and Triggs, 2005).

However, in computer vision, there are multiple non dominated functions designed for a same task: for image classification, there is a large set of classifiers with different quality and speed such that no classifier is both the faster and the better.

Let assume that, from a review of the state of art, one found K image classifiers. To tackle a given time constraint images classification problem, the simplest way is to select the classifier which best fit the requirement from the K preselected ones. But from optimisation/scheduling perspective, this approach is clearly not the one which maximizes the output quality So, how to design a system to classify batches of N images in T seconds using K black box classifiers.

A small step toward such maximization is done with the following model. Let assume that the system has a vector encoding the available resources $C \in \mathbb{N}^R$ and can rely on K classifiers each associated to an overall quality $q_1,...,q_K$, a speed $s_1,...,s_K$, and a resource consumption $c_1,...,c_K \in \mathbb{N}^R$. Let x_k be the number of images processed by the classifier k, as batch should be processed in T seconds, each classifier should process this own sub-batch in less than T seconds (leading to $\forall k, s_k x_k \leq T$) ; in addition, there is no direct interest to classify multiple times a same image ($\sum_k x_k \leq N$) ; finally, if a classifier is used ($x_k \neq 0$) then it uses some resources of C. Now let do the approximation that if a classifier is used, it is used until the end of the allowed time (so resource allocation is constant over the time - this is a one stage allocation assumption). Under these assumptions, the values x_k are the solution of the following integer linear program:

$$\max_{x_1,...,x_K \in \mathbb{N}} \left(\sum_k q_k x_k \right)$$
$$sc: \begin{array}{l} \forall k,\ s_k x_k \leq T \\ \sum_k x_k \leq N \\ \sum_k c_k \times (x_k \neq 0) \leq C \end{array}$$

Clearly, even if this set of inequalities may be NP-hard, it is in the format of well studied family of discrete optimisation problems and can be - at least - easily approximated.

But, the reality is much harder because this set of inequalities does not take into account cascaded decisions: one can apply a classifier on an image and react to the output of this classifier for example by using an other classifier only under some circumstances. More precisely, let suppose we are talking about binaries classification (images are either positive (e.g. target) or negative (e.g. background)) and assume that all classifiers have almost the same behaviour on positive but have very different behaviour on negative. It could still be interesting to use a fast classifier which will produce a lot of false alarms as soon as we could filter these false alarms using a slower classifier - and this is completely not taken into account by the previous set of inequalities. By the way, such cascaded strategy is the heart of (Girshick et al., 2014) which contributes to trigger the deep learning revolution by offering an algorithmic solution to the deep learning slowness using the box proposal paradigm - against

Chan-Hon-Tong A. and Herbin S.
Practical Scheduling of Computer Vision Functions.
DOI: 10.5220/0006086303470352
In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, pages 347-352
ISBN: 978-989-758-227-1

the sliding windows one - in object detection.

To return to our scheduling problem, modelling the possibility to form cascaded decisions would lead to a very challenging scheduling problem of stochastic programming with free number of recourse steps assuming probabilistic models exist for classifiers. Thus, computing the optimal allocation would probably take more time than the allowed time T. But worse, classifiers are not predictable: the number of false alarms produced by a classifier can significantly differ from validation set to test set. Thus, the behaviour of a given cascaded allocation is even not calculable (independently from the data) whereas cascaded strategies are known to belong to good strategies.

So to summarize, we take one of the simplest computer vision problem and the derived scheduling problem is yet intractable.

However, we argue that scheduler adapted to computer vision functions (like classifiers) may be useful not for the sake of the performance but to make robotic development easier. Such scheduler - even if producing output with quality slightly under state of the art - could be useful: it could allow easy prototyping, boost the apparition of robotic platform embedding a large variety of computer vision functions, and, separate computer vision from integration implementation.

We argue that robotic community heavily relies on middleware like ROS (Quigley et al., 2009) or YARP (Metta et al., 2006) very useful for easy prototyping and large project even if such middleware obviously introduces computation overhead - this is not that far from using scheduler of black box functions. Thus, we believe that this work about plug and play scheduling of computer vision functions is relevant for robotic community (and computer vision one) in a way ROS and YARP are.

In the following, we first describe related works in section 2. Then, in section 3, we describe a simple but effective scheduler for the time constraint batch classification problem under real setting both in term of hardware target (hybrid CPU GPU) and in term of classifiers (e.g. with Alexnet). The relevancy of this scheduler is validated on experiment on real computer vision dataset (in section 4), before conclusion in section 5.

2 RELATED WORKS

The task of selecting combinations of actions to optimize a process is a generic task that overlaps at least three well studied fields: discrete optimisation (Ouelhadj and Petrovic, 2009; Graham et al., 1979), path planing (Lamiraux and Laumond, 2000; Marti and Qu, 1998), and Markov decision process (Bellman, 1957; Song et al., 2000).

In this paper, we want to apply scheduling to computer vision functions. However, computer vision is an exigent field in resources consumption: for example, in automatic car driving, computer vision system should typically process an HD image each 33ms whereas 1 hour of uncompressed HD video is around 1To. In addition, computer vision is a data driven field where effect of a computer vision function dramatically varies depending on data making impossible to precompute computer vision function effects.

Thus, there is a crucial difference between classical scheduling context and our context: here, time taken to compute the scheduling is required to be insignificant regarding action duration (which is yet very low) and can not be precomputed.

In discrete optimization, the goal is often to precompute an allocation of the resources. The ability to react to the effect of an action that could have been different from what was expected is handled in particular by stochastic programming (SP) e.g. (Ferrero et al., 1998). In SP, the goal is to select the first order variables before seeing the realisation of the unknown parameters taking into account that second order variables will be selected after seeing the realisation of the unknown parameters. So, it is a clear way to handle dynamic problem with partially know parameters. However, we found no SP literature relevant to handle problems with very high dynamic (like having to schedule the processing of 250000 jobs in less than 200ms).

There are also difference with path planning. In path planning, optimisation is usually considered at trajectory level while, in this paper, optimisation is considered at a level whose dynamic has the same order of magnitude than engine controller one. There exist examples (like (Van Den Berg et al., 2011)) of path planning optimisation at engine controller level. However, this literature is relatively restricted. And, even if scheduling was relevant on such time scale: engine controllers are much more regular than computer vision functions.

These considerations also stand for Markov decision process. In Markov decision process, applying an action a from a state s is often simply getting the value $T[a][s]$ of the matrix T. Thus, the action duration is insignificant regarding the time of optimisation which is the opposite of our context.

For all these reasons, there are only very few papers applying scheduling methods to computer vision context. From these papers, the closers to our work are (in our opinion) (Trapeznikov and Saligrama, 2013) and (Russakovsky et al., 2015). In (Trapeznikov and Saligrama, 2013), the considered system is a predefined cascade of K classifiers. Each incoming data can either be classified by the classifier k or be considered as suspicious and given to the classifier $k+1$. As classifier K cost much more than classifier 1, the trade off is to tune the system to achieve high quality at low cost. In addition to provide framework to deal with this trade off problem, (Trapeznikov and Saligrama, 2013) use Markov decision process theory to prove that a simultaneous training of all classifiers taking into account the cascaded structure of the system is possible. This work is thus close to ours, however, our goal is to dynamically schedule classifier jobs - considering classifiers as black box. So we are not scheduling at the same level.

Finally the closest work, in our opinion is (Russakovsky et al., 2015). In (Russakovsky et al., 2015), dense application of deep learning classifiers is precomputed on the batch of images. Then the goal is to interact with an human operator to extract from this dense precomputed values, the set of object present in the batch of images. The system takes as input the cost of several human actions (drawing a bounding box, tell if a bounding box is correctly located, tell is there is an object in the image, tell is there is a certain kind of object in the image ...) and a trade off criterion between the cost of human annotation and the quality of the the set of extracted boxes. Clearly, if we add human interaction to the set of computer vision functions, then scheduling the interaction with a human is part of what our scheduler would do. The main difference is that we do not precompute computer vision values instead we are doing it online (i.e. on the fly) while reacting to the previous computation. For this reason (Russakovsky et al., 2015) is not directly applicable. However, some Markov decision process tools are shared between (Russakovsky et al., 2015) and our work. In particular, we also use look ahead planning (LAP).

3 OFFERED SCHEDULING FRAMEWORK

3.1 Basic Assumption

We describe in this section our framework for scheduling computer vision classifiers to tackle the toy problem of time constraint image classification. The system should classifies a batch of N images in T seconds using K black box binary classifiers whose expected quality $q_1,...,q_K \in \mathbb{N}$ (assumed to be scalar), speed $s_1,...,s_K \in \mathbb{N}$ (assumed to be scalar) and resource consumption $c_1,...,c_K \in \mathbb{N}^R$ are known. And, all of this takes place on a hardware platform with $C \in \mathbb{N}^R$ available resources. In the following, some of the images can be left unprocessed, by default, its are then considered as positive.

Classifiers are basically evaluated by 2 numbers precision and recall (or better by the precision/recall curve). In order to reduce this 2d quality, we tune all classifiers such that its have common and relatively high recall: it means that classifiers have almost the same behaviour on positive images. Thus, under this tuning, quality is just the precision (i.e. a scalar as assumed).

However, we allow large variability in speed, resource consumption and precision (which is linked to false alarms rate: the better is a classifier, the lower there are false alarms - as recall is fixed).

Then, at each moment, the state of the system is map between image and it current label and confidence on this label.

The processing of the batch of images is done by round of action. At each round, the scheduler selects an action from the predefined pool of allowed actions based on the current state.

Each classifier is encapsulated into an elementary action. Applying an elementary action on a state consists in: sorting images per confidence (if not already done), extracting images with lowest confidence, classifying each extracted images and re-inserting them on the state.

The number of images extracted by an elementary action depends on the speed of the underlying classifier. More precisely, we select a step duration and each elementary action should have this duration.

A set of elementary actions which can be run simultaneously (considering the set of available resources) is an action. A crucial point is that when we apply an action in a state, all elementary action select its images sequentially but images processing is done in parallel and no reinsertion is allowed until all elementary actions have finished (then re-inserting is done sequentially). So into a same action, elementary actions can not process a common image.

3.2 Look Ahead Planning

Like (Russakovsky et al., 2015), we rely on look ahead planning (LAP) to schedule actions. Precisely, we perform $\frac{T}{\delta}$ steps of LAP from the initial state.

LAP is a classic method to deal with dynamic system consisting in two main steps : to explore paths

```
LAP(A, state, n)
   for t from 1 to n
      paths = [state]
      explore(paths,A, n-t)
      a = argmax(paths);
      state = apply a on state;
   return state;
```

Figure 1: the pseudo code of the Look Ahead Planning.

```
explore(p, A, n)
   for a in A
      for t from 1 to n
         for b in A
            best=a
            if p[a]::b > p[a]::best
               best=b
            p[a] = p[a]::best
```

Figure 2: Greedy routine exploration in LAP.

from a starting point to compute the reachable score and to apply the first action from the best path. Applying only the first action and looping allows to react to the effect of the action as it is different than the expected effect. This way the scheduling is not precomputed and then executed but instead computed on the fly while being executed. The pseudo code is described in figure 1.

Notice that we have introduced the concept of score of a state without describing it. For real states, we use the expected precision as state score. This computation is done using the classifier known quality: the expected precision is simply the average on the images classified as positive of the precision of the classifier used on. The use of the precision is relevant as all classifier have a common recall.

Now, there is two problems with this version of LAP:

- we can not explore all paths as there is α^{β} paths where α is the number of actions and β the temporal horizon of the optimisation
- we could - at this point - not explore at all because applying an action onto a state to know the resulting state is exactly what we do not want to do: the computational time should overwhelmingly be used to apply action to state and not to select the next action.

For the first problem, we use, like in (Russakovsky et al., 2015), a greedy pruning of the exploration to maintain a low computational volume while taking advantage of long term information. The pseudo code of the exploration is described in figure 2.

For the second problem of LAP, we do not need exact state computation: we only need a coarse estimation of what would be the state after the application of the action. For this purpose, we introduce virtual states which is one of the main contribution of this work.

3.3 Virtual States

We offer to simulate the application of computer vision functions on virtual representation of the state.

In particular, if the real state is the vector of label/confidence, a virtual representation is just the distribution of label/confidence forgetting individual information and mapping to the images.

Now, the critical point is to be able to estimate the virtual state resulting of an elementary action using only it known quality-speed.

The main observation allowing such computation is the following. Let assume that the state is composed of α negative images and β positives ones associated with a precision of λ. Then applying a classifier of precision $\mu > \lambda$ on $\gamma < \beta$ positive images leads on average to $\alpha + \gamma - \frac{\gamma\lambda}{\mu}$ negatives and $\beta - \gamma + \frac{\gamma\lambda}{\mu}$ positives ($\beta - \gamma$ with precision λ and $\frac{\gamma\lambda}{\mu}$ with precision μ). These equations comes from the assumption of conservation of positive (which can be adapted to take into account a proportion of loss) and from the equation $positives = estimate\_positive \times precision$.

These equations are directly extended on a distribution of label/precision.

Using this trick of virtual states, the selection of action is pretty close to Markov decision process framework. Now, we can really perform the simple path exploration like in (Russakovsky et al., 2015) to estimate the reachable virtual states (and score) from a given (real) state. And thus, we can really use the LAP framework (described in figures1)

Now, we have completely described our scheduler. To summarize, our framework consists to use virtual state to allow exploration of reachable states from a given real state - allowing LAP framework to deal with the actions scheduling.

4 EXPERIMENTS

Our scheduler is designed to tackle time constraint image batch classification. The simple use case of such problem is object detection using sliding windows framework: the batch is composed of all the positions of the sliding window and the time allowed is allowed time to process the image.

Let accept the following setting of the experiment: the scheduler should process a car detection on an area of a standard city $75km^2$ (on 5cm-resolution images) in half an hour on standard hardware : 6 intel7 CPU, 6Go of RAM and 1 GeForce GTX 750.

4.1 Data

We choose to use the 2015 IEEE GRSS Data Fusion Contest to perform our experiments. This dataset (called grss_dfc_2015 in this article) is composed of 6 large size (10000x10000 pixels) remote sensing ortho-images with resolution of around 5cm (only RGB image are used in this experiment).

A manual semantic labelling has been made by (Lagrange et al., 2015).

We adopt a 128x128p mono scale sliding window framework with spatial displacement of 20 pixels of the window. So a 10000x10000p image leads to 250000 128x128p images to processes in 12s: this is almost 2 order of magnitude the number of 128x128 images that the AlexNet convolutional neural (upscaled in 227x227p) network can processes in 12s with a GeForce GTX 750 making relevant the possibility to combine different classifiers to process more 128x128p images (even if these other classifier are less accurate than Alexnet).

We split the dataset in 3 sets (train/val/test) of 2 images. Each computer vision classifier is calibrated on the validation set (after a training on train set) in order to obtain statical characteristics of each classifier. The scheduler considers each computer vision classifier like a black box only coming with a calibration report. The complete system (scheduler + classifier) is tested on the test set (composed of the 27032011_315140_56865 and 27032011_315135_56865 images).

4.2 Computer Vision Library

The computer vision classifiers considered for the experiment are summarize in table 1 (we downscale the original 128x128p image for 64x64p or 32x32p classifier and upscale the original 128x128p image to 227x227p for alexnet) The table 1 presents performances of all classifier and statistic extracted from the auxiliary data. This table is the only classifier descriptions available to the scheduler.

4.3 Results

We run both our scheduler and several baselines with this experiment setting. In our opinion, all elements of this experiment are realistic: data are real life remote sensing images, hardware target (C) is a standard hybrid CPU GPU environment, the allowed time (T) seems representative and considered classifiers contains in particular both outdated HoG and state of the art Alexnet. In all experiment, one CPU is allocated to the image server (so the targeted system behaves like having 5 CPU instead of 6).

Table 1: Performance and speed of the classifiers on the validation test.

classifier	precision	speed
alexnet	10%	280
lenet 128x128p	3%	1088
lenet 64x64p	2%	5739
hog 128x128p	1%	1580
hog 64x64p	0.7%	6185
hog 32x32p	0.1%	14229

All classifiers have a common recall of 90% and speed is measured in 128x128p images per second. We use caffe (Jia et al., 2014) implementation with cudnn3 to reproduce expected speed for deep learning classifier.

The considered baselines are:

- the better available combination: mapping one GPU and one CPU onto alexnet and one HoG 128x128p on each other CPU
- the faster available combination: mapping each CPU to HoG 32x32
- the better combination respecting the constraint of processing almost all 128x128p images: here, this leads to map all CPU on HoG 64x64p

The results are summarized in table 2.

Table 2: Performance of our system (scheduler + classifiers) and baselines on the grss_dfc_2015 dataset.

system	F_1 (precision/recall)
scheduler	**54%** (50%/75%)
baseline *faster*	5% (2%/90%)
baseline *better*	9% (4%/90%)
baseline *better finisher*	8% (3%/90%)

Performance is measured in F_1 measure which is $2\frac{precision \times recall}{precision + recall}$. All system terminates in the allowed duration - time overhead introduced by the scheduler against computer vision computation is less than 3%.

The global output of the scheduler is incomparably better than baseline ones in F_1 measure.

This score comes with a loss of recall but with a high precision gain. In order to ensure a fair evaluation of all methods, we evaluate each baseline with different classifier tuning: for each classifier, one can tune the tradeoff between precision and recall by biasing the classifier output, this tradeoff is tuned to lead to 90% of recall when classifier are used by the scheduler but other tradeoff have be evaluated to ensure that this tradeoff selection does not bias the evaluation. However, even with a grid search on all tradeoff, all baselines are always at least 30% lower in F_1 measure than the scheduler.

Thus, this experiment confirms the relevancy of such scheduler at least for time constraint batch classification.

Currently, the very high F_1 score of the scheduler could not be expected from the calibration (see table 1). As, all classifiers are calibrated on validation set to have a common recall of 90%, precision should be the only mutable value. This is however not the case due to interaction in the cascaded decision: our system has a lower recall but achieves higher precision than expected. We believe that this could be explained by the difference between calibration which is done classifier per classifier and the scheduling were boxes are finally classified by multiples classifiers. In other words, the considered classifiers have some complementarity that are freely exploited by the scheduler. Anyway, this does not invalidate the evaluation. A more careful study of this last result is out of the scope of this paper whose main result (in our opinion) is that the scheduler is able to produce sufficiently good output.

5 CONCLUSION

The goal of this paper is to make a step toward schedulers able to help the integration of large computer vision library into complex robotic system. To make a first step, we chose a scheduling problem derived of one the simplest computer vision: time constraint batch classification. We describe a scheduling framework for this problem. We apply it on car detection on real remote sensing images with realistic settings in term of time constrains, target hardware (hybrid CPU-GPU) and computer vision classifiers (with deep learning ones). The results of this experiment is that our scheduler goes further than expected by achieving state of the art results - while it is just designed to produce a *sufficiently good* results to be relevant for prototyping/integrating requirements.

REFERENCES

Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR*.

Ferrero, R., Rivera, J., and Shahidehpour, S. (1998). A dynamic programming two-stage algorithm for long-term hydrothermal scheduling of multireservoir systems. *Power Systems*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *ICM*.

Lagrange, A., Le Saux, B., Beaupere, A., Boulch, A., Chan-Hon-Tong, A., Herbin, S., Randrianarivo, H., and Ferecatu, M. (2015). Benchmarking classification of earth-observation data: from learning explicit features to convolutional networks. In *IGARSS*.

Lamiraux, F. and Laumond, J. (2000). Smooth path planning for car-like robots. In *ISAS*.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*.

Lucas, B. D., Kanade, T., et al. (1981). An iterative image registration technique with an application to stereo vision. In *IJCAI*.

Marti, K. and Qu, S. (1998). Path planning for robots by stochastic optimization methods. *Journal of Intelligent and robotic Systems*.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: yet another robot platform. *IJARS*.

Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA*.

Russakovsky, O., Li, L.-J., and Fei-Fei, L. (2015). Best of both worlds: human-machine collaboration for object annotation. In *CVPR*.

Shan, Q., Jia, J., and Agarwala, A. (2008). High-quality motion deblurring from a single image. In *ACM Transactions on Graphics*.

Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV*.

Song, H., Liu, C.-C., Lawarrée, J., and Dahlgren, R. W. (2000). Optimal electricity supply bidding by markov decision process. *Power Systems*.

Trapeznikov, K. and Saligrama, V. (2013). Supervised sequential classification under budget constraints. In *ICAIS*.

Van Den Berg, J., Abbeel, P., and Goldberg, K. (2011). Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *IJRR*.

Zhang, Y. J. (1996). A survey on evaluation methods for image segmentation. *Pattern recognition*.