Colour Patch Detection using State-of-the-Art Deep Learning Algorithms

Allwyn Joseph
University of Jean Monnet
allwyn.joseph@etu.univ-st-etienne.fr

Evhenii Zotkin
University of Jean Monnet
yevhenii.zotkin@etu.univ-st-etienne.fr

Abstract

In recent years, Deep Learning has come to play a pivotal role within the Computer Vision (CV) domain. From image classification to object detection, deep learning is supplanting traditional CV techniques and becoming the state-of-the-art (SOTA). In this work, we focus on using some of the SOTA algorithms to detect 24 colour patches in images. The work primarily revolves around understanding, implementation and testing of the YOLO-v3 and the RetinaNet algorithms for the object detection task at hand. The code and related literature for our work can be found in this link.

1. Introduction

This project was tasked upon us by our 'Deep Learning and Applications to Computer Vision' coursework, with the aim of getting us to better understand and work with the SOTA object detection algorithms. As of today, there exists several object detection algorithms, off which some of the most commonly used ones are:

- **R-CNN**[4] - The Region-based Convolutional Neural Network (R-CNN) combines selective search (for region proposal detection) and deep learning to detect objects. The region proposed after running selective search is re-sized to match the input of a CNN from which a vector of features is extracted. This is then fed into multiple classifiers which output probabilities to belong to each class. Each one of these classes has a trained SVM classifier to infer detection probability of an object given the vector of features. This vector also feeds into a linear regressor to adapt the shapes of the bounding box for a region proposal and thus reduce localization errors.
  In the Fast R-CNN derivative of the algorithm, instead of using a CNN for each region proposals, a single CNN with multiple convolution layers is used to generate feature maps (onto which selective search is applied to detect the region of interests). This reduced the computation time required for a large number of models to analyse region proposals from selective search.
  In the Faster R-CNN derivative of the algorithm, to further speed up computation time selective search was replaced by the Region Proposal Network (RPN) directly generate region proposals, predict bounding boxes and detect objects (a combination between the RPN and the Fast R-CNN model).
- **R-FCN** [2] - The Region-based Fully Convolutional Network (R-FCN) is a model with only convolutional layers allowing complete back-propagation during training. The network uses a combination of a ResNet and an RPN to classify and detect objects. The ResNet produces feature maps which are specialized in the detection of a particular object, while the RPN generates the region of interests essential to localised detected objects.
- **YOLO** [11]- The You Only Look Once (YOLO) algorithm is a completely convolutional one-shot learning algorithm which effectuates both, image classification and object detection on the feature map outputs from the network (allowing for real-time predictions). YOLO-V2 and YOLO-V3 are derivative of the YOLO algorithm which led to faster and more accurate detections (discussed in more detail in section 4).
- **SSD** [10] - Similar to the YOLO model, the Single-Shot Detector (SDD) algorithm predicts both bounding boxes and the class probabilities in a single shot. The model takes as input an image, and outputs feature maps at different positions of the network which are then used to predict the bounding boxes and classify objects within them.
- **NASNet** [15] - The Neural Architecture Search (NASNet) consists of learning the architecture of a model to optimize the number of layers while improving the accuracy over a given dataset.

Off the aforementioned algorithms, YOLO-V3 and RetinaNet where to picked to realise the object detection task (expounded on in section 2).

2. Dataset, Objective and Performance Metric

Most of the job related with practical training was in constructing the reliable pipelines of data preprocessing, training, model selection and validation. We first begin by addressing the proplem of preprocessing. The original images comes in 12-bit untreated PNG format. The proposed in course technique led to increasingly greenish images on the first camera. When tested on the other cameras datasets it resulted in dark images. Finally, we failed to find the optimal prepossessing routine and used suboptimal, but acceptable pipeline. We normalize images (scale pixel values) using the following formula:

$$image_{normalized} = (\frac{image - min}{max - min})^{1/2.2}$$

where max is per channel max pixel value and min is per channel min pixel values. The subtraction is performed channel-vise. Lastly, as suggested in the course description we perform the inverse $\gamma - correction$ with $\gamma = 2.2$. The images is then resized to 640x640 in case of RetinaNet and 604x604 in case of yolo using nearest neighbors interpolation. As we will discuss in the improvements section, this approach is suboptimal and driven only by computational expenses logic.

Predicted results are evaluated using mean average precision metric.

3. RetinaNet

In this section we will discuss the second approach we tested on the dataset, the RetinaNet [9] architecture. Being the result of evolutionary improvement, we find it important to discuss all the building blocks of the model to fully appreciate the performance benefits, understand the drawbacks and discuss the potential improvements of the model. In further sections we will discuss the Resnet architecture [5], Feature Pyramid Networks [8] and finally Focal Loss, which is distinctive part of Retinanet architecture. In the last subsections we discuss the details of technical implementation and possible heuristics to improve our models.

3.1. Resnet

Resnet is a CNN architecture which is most commonly in modern neural networks. Numerous extensions to the original network architecture were proposed to the day [14][6], but we find that they all follow the same main idea behind ResNet - skip connections.

The problem which motivated the invention of residual networks is the so-called Vanishing/Exploding gradients. Intuitively, observing the impressive performance of deep neural networks one might think we can improve any models by simply adding more layer to the architecture. Indeed, the argument that even if increasing capacity of the network won't increase the performance at least it is going to remain the same. Theoretically, this argument is perfectly valid, since we can claim that if network which has n layers has performance P we can always have $n + 1$ layers with performance at least P while simply learning the identity projection in the last layer. Unfortunately, in the reality, we often find that by increasing capacity of the network we actually decrease the performance (we assume that the network is properly regularized and this drop is not due to overfit). This phenomena is often due to Vanishing of Exploding gradients. Since during the backpropagation phase we often perform multiplication, we can observe that numbers are increase/decrease with each additional layer(multiplication). This will result in Vanishing/Exploding gradients for the nonliniarity (or in dead neurons in case of ReLU activations).

To mitigate this effect, numerous solutions were proposed. For instance GoogleNet with their Inception architecture [13] propose to use additional classifiers to improve the grafient flow. Resnet architecture introduce the idea of skip-connections which are to propagate the original featuremaps deeper into the network during the forward pass and to propagate the gradients during the backward pass. Formally, assuning the x denote the input of the n^{th} layer of the deep neural network, $F(x)$ if the output of $(n+k)^{th}$ later authors inroduce the skip connection as $F(x) := F(x) + x$ where x skips k layers. Practically, x is often represented as $C(x)$ where C is 1x1 convolution to match the spacial dimentions of $F(X)$. The latter is not strictly necessary (authors underline that the model achieve remarkable performance on ImageNet even with simple identity projection) but often used in practice.

This architecture is the backbone of Feature Pyramid Network used in the RetinaNet architecture.

3.2. Feature Pyramid Networks

The Feature Pyramid Network (FPN) as a feature generator for object detection models. FPN comes as a fast and efficient replacement of sliding window approaches used in earlier models like R-CNN and FastR-CNN. FPN allows for fast end-to-end training of object detection network. The idea of FPN comes from the properties of the general convolutional networks. One can describe typical convolutional network as feature pyramid where each next layer is a condensed version of the previous layers. On the other hand, single shot detectors such as SSD use the same idea by selecting regions of interest in the top level of feature pyramid. While being fast, this approach lead to sub-optimal performance and practical impossibility to detect small features. To tackle this, FPN adopt the novel approach of re-using lover level featuremaps to formulate different-size proposals. They use 2 ResNet52 (this is variable parameter, implementations with deeper variants can be found in the lit-

erature) architectures, one normal and one in with all convolutional layer are replaced by upsampling layer. This architecture, one the one hand gives the possibility to perform end-to-end learning and, on the other, gives the possibility to leverage lower level features, and thus, improve the peformance of the models both with large and small objects.

3.3. Focal Loss and Retinanet

The most accurate object detection models are based on the 2 stage approach proposed by R-CNN framework. In this approach, a classifier is applied to the sparse set of features obtained from the output of first stage. While this approach shown to be wery efficient in terms of accuracy of the predictions, it is also slow in terms of time of prediction. For instance, the time for prediciton makes it prohitive to use in on-line settings.

On the other hand, many single shot architectures were proposed lately, such as SSD and YOLO. This type of the architectures allow for the real time prediction but often falls short in the accuracy of the prediction. As the authors of the RetinaNet argue, this difference is (at least partially) due to the dramatic class imbalance which network suffers during the training. In case of R-CNN like detectors this class imbalance is handled by the two-stage cascade and sampling heuristics. The proposal stage reduce the number of predictions to 1000 and eliminates most of the background class. In the case of single shot detectors the network needs to process on average 100000 proposals which are dominated by easily classifiable background class.

In this setting one can observe that allowing for the same amount of loss induced by background and target classes is sub-optimal. Several approaches exist to handle this problem, such as bootstrapping and hard example mining, but they all result in slow training and sub-optimal in their nature.

The novel approach presented in RetinaNet paper consist in tackling this issue on the loss function level. They propose new loss function, named Focal Loss, which implicitly handles class imbalance issue. Formally, this loss function is formulated as an extension of Cross Entropy:

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma} log(p_t)$$

where p_t is the probability p output of the classifier if $y = 1$ and $1-p$ otherwise. γ and α are tunable hyperparameters. γ defines how little loss will be incurred from background class. α acts as the balancing factor. When $\gamma = 0$ and $\alpha = 1$ this loss function is equivalent to Cross Entropy. Experimentally, authors find that using $\gamma = 2$ leads to the best result. We follow this recommendations in our experiments.

3.4. Notes on practical implementation and training

To train our network we decided to use Tensorflow[1] and built-in Object Detection API[7]. Object detection API provides a ResNetXt implementation of RentinaNet.

Since available dataset is small as for object detection task (2000 images in total) we decided to adapt transfer learning approach. It is now a common practice to use transfer learning when faced with computer vision task. We used a network which was first pretrained on the ImageNet dataset and then trained on COCO dataset. There exist several common approaches to transfer learning. For our task, we define two possible options:

1. Freeze the network backbone and only train classification and regression heads. This approach is commonly used when there are little data available.

2. Don't freeze anything and use pretrained network as good initialization.

For our task we decided, perhaps counter intuitively, to continue with second approach. The motivation behind this decision is that in our task most of the classes from COCO dataset will be considered as background (cars, humans).

During training we first reached 85 % map@0.5 after 10 epochs. We found that using cosine annealing learning rate scheduling let to train for the longer times and improves the accuracy. We were able to reach 94% map@0.5 after 20 epochs of training

3.5. Possible improvements

Since we tested our models on 2 datasets, we observed 2 distinct error types. In this section we discuss the error made by RetinaNet.

First type of error occurs in the first dataset. As can be observed at 5 in the first dataset we often can observe the situation where patches of the color which does not belong to the color board are detected as color patch. Our hypothesis is that this occurs due to suboptimal preprocessing of the images. Also, we see that in areas of the image where the structural pattern is similar to the checker board (brick walls, etc.) we can observe false positive detection. We think that by correctly preprocessing images we can mitigate this effect.

Second kind of error occurs in the second test dataset. As can be observed in 10, RetinaNet model often fails when colorboard rotated in space. As we saw during examination the dataset, there are no images with rotation ¿ 45 in the training dataset. Naturally, model fails when it see data which was not present in the training dataset. We tried to mitigate this effect with data augmentation, but it does not help with heavy rotations. Possible way to improve is to use test-time augmentation. We can cut the image into tiles,

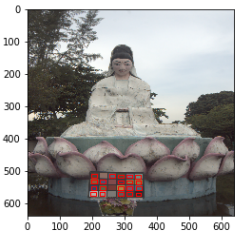

Figure 1: False Negative (1)

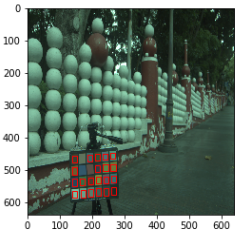

Figure 2: False Negative (2)

Figure 3: False Positive (1)

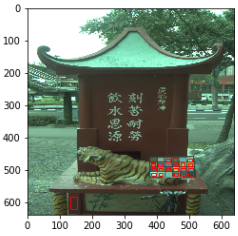

Figure 4: False Positive (2)

Figure 5: Example of detection by RetinaNet where type 1 and type 2 errors occurs on the First dataset

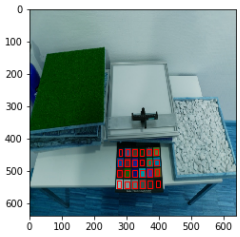

Figure 6: Perfect detection (1)

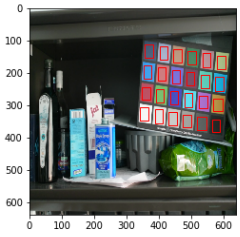

Figure 7: Perfect detection (2)

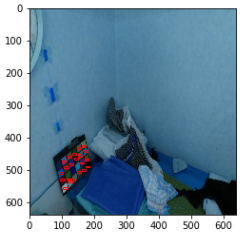

Figure 8: False Negative due to rotation (1)

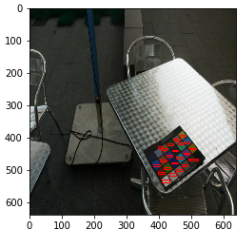

Figure 9: False Negative due to rotation (2)

Figure 10: Example of detection by RetinaNet where type 1 and type 2 errors occurs on the second dataset

randomly rotate them, and make predictions on the rotated images.

4. YOLO-V3 [12]

The YOLO-V3 algorithm is a derivative of the vanilla YOLO algorithm based on a one-shot learning. In comparison to versions one and two of the algorithm, version three is fast and more accurate with it predictions. To better understand YOLO-V3, this section will breakdown and dwell deeper into each facet of what make this a formidable object detection algorithm.

4.1. Darknet-53

Also know as the feature extractor, the Darknet-53 network acts as the backbone of the YOLO-V3 network. In reality the YOLO-V3 network is composed by stacking two Darknet-53 networks together. Ideally the first network is trained on the ImageNet dataset [3], while second network servers to carry out detection tasks on image inputs. The stacked network constitutes five differnet layers:

- Convolutional layer : 75 such layers with filter sizes of either 1 or 3 and varying number of filters are used to perform the convolution operation.
- Shortcut layer : 3 such layers which acts as a skip connection, similar to what is seen in ResNets in section 3.1.
- Up-sample layer : 2 such layers occurring over the course of the first two detections (at the Yolo layer) wherein the feature map of the previous layer is augmented or up-sampled.
- Route layer : 4 such layers that simply outputs feature maps of previous layer(s).
- Yolo layer : 3 such layers that corresponds to detection layers. As will be seen in section 4.4, YOLO-V3 makes predictions across three different scales allowing for more accurate detections.

All in all, the network has 106 layers and is fully convolutional without pooling of any kind, thereby helping with conserving the low-level features.

4.2. Output Feature Maps

As mentioned earlier, the YOLO-V3 network makes detections at three different scales. During each of which the network outputs feature maps whose dimensions are (D x D x (B*(5+C))). The value of D is decided by the height and width of the image input to the network and the stride of the detection layer (strde $\in$ [32, 16, 8]). B stands for the number of bounding boxes the network is capable of detecting in each grid (number of grids = D*D). And C is the class confidence of the object detected within the bounding box. The number 5 stands for the bounding box dimensions (t_x, t_y, t_w, t_h) and the confidence score implying whether an object has been detected or not.

4.3. Bounding Box and Class Predictions

As seen in the above section, the network predicts bounding box coordinates as t_x, t_y, t_w, t_h, where t_x and t_y represent the center of the box and, t_w and t_h represent the width and height of the box. These values are however not representative of the actual scale of the image input and are rather offsets. In order to calculate the true bounding box center and dimensions the following equations are used:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w \exp^{t_w}$$
$$b_h = p_h \exp^{t_h}$$

where

b_x, b_y : actual bounding box center coordinates with respect to input image dimension
b_w, b_h : actual bounding box width and height with respect to input image dimension
c_x, c_y : upper-left corner coordinates of a given grid where the detection was effectuated
p_w, p_h : width and height of pre-defined anchor box

During training the sum of squared error loss is used to learn weights to output offset values close to ground truth (computed by inverting the above equations).

Each bounding box predicts the class of the object it encompasses - only a single object is detected by per box. And instead of using a softmax for multiclass classification, independent logistic classifiers are used. Using a logistic classifier helps overcome the innate problem of softmax classifiers which deem that object classes are mutually exclusive, i.e. an object can only belong to one class. And during training, the weights associated are updated using the binary cross-entropy loss. Also, as mentioned earlier, for each bounding box an objectiveness score is output using a logistic classifier. A value of 1 would mean that a pre-defined anchor box overlaps a ground truth object more other pre-defined anchor boxes.

4.4. Prediction Across Scales

As mentioned earlier, the YOLO-V3 network predicts boxes at three different scales, and at each of these scales 3-d tensors of dimension (D x D x (B*(5+C)) is output. To dwell deeper, at each detection, a feature map is first output - detection occurs after a series of downsampling steps. In the layer following the detection, the output feature map is combined (summed) with a feature map output of a layer a little earlier in the network. This new feature map is then convolved and upsampled to twice its size. This allows one to get more meaningful semantic information while preserving finer-grained information from the earlier feature maps. The upsampled feature map is then pushed through a few more convolutional layers (processing this combined feature map), until the next detection layer. And this time the feature map output is twice size of the previous detection layer feature map output. The feature map undergoes a similar process as with the previous detection layer output

(a) Good Detection

(b) Modest detection

Figure 11: Colour-Patch detection on images in dataset 1

feature map and the process is repeated yet another time to realize predictions at the final scale.In theory, the prediction at the third scale benefits from all the prior computation and preserved fine-grained features.

As for the pre-defined bounding box dimensions, nine different boxes - three at each detection layer - were used of the following dimensions: (10x13), (16x30), (33x23), (30x61), (62x45), (59x 119), (116x90), (156x198), (373x326).

4.5. Implementation and Training

For the purposed of the object detection task, a YOLO-V3 network implemented in Pytorch was pulled and developed upon to better suite the colour patch object detection task.

Given that we had less 2000 images to train on, and this being an insufficient amount of data to train a robust model from scratch, we resorted to transfer learning. To this end, we initialized the backbone - Darknet-53 - of the network with weights obtained from training the network on the ImageNet database. Further on, the initialized model was trained on colour-patch dataset for 90 epochs attaining a **mAP@0.5 score of 0.743**

(a) No Detection

(b) Multiple detections and unusually framed ROI

(c) Multiple detections and unusually framed ROI

Figure 12: Colour-Patch detection on images in dataset 2

4.6. Results and Discussions

Figures 11 and 12 exhibit some of the results obtained on feeding the trained YOLO-V3 network with images from dataset 1 and 2 (more images in github repository). The network in general, is able to discern the ROI (colour-patches) from the other elements present in the input image. This being said, the network has three main pitfalls:

- It falters when the colour-patch rectangle is positioned at an angle. As can be seen in image (b) and (c) in figure 12 the ROIs detected (pink box image (b) and green box image (c)) are unusually framed.
- It detects multiple instances of the unique colour-patches as can be seen in image (b) and (c) in figure 12.
- It fails to detect objects when the colour-patch rectangle area is very small in comparison the dimensions of the image, as can be observed in image (a) figure 12.

The following are our hypothesis for the pitfalls of the network:

- The dataset on which the network was trained was not adequately pre-processed due to the lack of accurate information concerning the same.
- The train dataset had very few images in which the colour-patches were titled at an angle, which explains the difficulty the network faces while detecting ROIs and patches on images of this genre.
- The original YOLO-V3 implementation was put together to be able to detect fairly sizable objects, and With this in mind, the anchor-box sizes were defined. The training dataset unfortunately had numerous images wherein the colour-patch rectangles were small. This in turn could have bemused the network into learning non-optimal weights resulting in multiple detection of a colour-patch on an input image with unique colour-patches or no detections at all.

4.7. Possible Improvements

In context with the aforementioned drawbacks of network, following are the suggested course of action for plausible amelioration in detections:

- Better pre-processing of the images.
- Custom generate pre-defined anchor boxes to better suite the dimensions of the colour-patches.
- Decrease the stride of the network (over the detection scales) to avoid multiple detections of unique patches within images.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 3

[2] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016. 1

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 5

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:1–1, 12 2015. 1

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017. 2

[7] J. Huang, V. Rathod, D. Chow, C. Sun, M. Zhu, A. Fathi, and Z. Lu. Tensorflow object detection api. *Code: github. com/tensorflow/models/tree/master/object detection*, 2017. 3

[8] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017. 2

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 2

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector, 2015. cite arxiv:1512.02325Comment: ECCV 2016. 1

[11] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016. 1

[12] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 4

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[14] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017. 2

[15] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. 2017. 1