Detector-Encoder Autoencoders for Unsupervised Decomposition into Visual Parts

Mohammad Sadil Khan
M.Sc in Machine Learning and Data Mining
Supervisor: Remi Emonet, Theirry Fournel, Amaury Habrard

Hubert Curein Laboratory
University Jean Monnet
July 2,2021

Contents

Acknowledgement

It's a genuine pleasure to express my deep sense of thanks and gratitude to my mentor and guide Professor Rémi Emonet and Professor Thierry Fournel. Their dedication, constant supervision , keen interest and above all an overwhelming attitude towards me have made it possible to complete my work. Their timely advice, meticulous scutiny, helpful suggestions and scientific approach have helped me to a great extent to accomplish my task.

I owe a deep sense of gratitude to my parents. Without their steady support and encouragement, none of it would have been possible.

1 Abstract

In this project, we present a novel idea of creating autoencoders for anomaly detection, which will help decide whether some documents (thus the books) can be attributed to Marc-Michel Rey, a publisher of Thinkers. DAE, detecter-encoder autoencoders is end-to-end trainable and uses a detector as a replacement of encoder. We use the feature maps in the detector to extract Region of Interest and use it as an encoded part of the image. We also introduce AutoLabelme, a graphical user interface and an extension of LabelMe, for efficiently and automatically annotating Images. We also present Ransac, a gui which can align some given images based on a reference image. The github code is available at https://github.com/SadilKhan/internship-hubert-curien-2021.

2 Introduction

2.1 Lab Overview

The Hubert Curien laboratory is a joint research unit (UMR 5516) of the Jean Monnet University, Saint-Etienne, the National Research Centre "CNRS" and the Institut d'Optique Graduate School. It is composed of about 90 researchers, professors and assistant professors, 20 engineers and administrative staff and 130 PhD and post-PhD students. This makes the Hubert Curien laboratory with a total of about 240 staff the most important research structure of Saint-Etienne. The research activities are organized according to two scientific departments: Optics, photonics and surface and Computer Science, Security, Image.

2.2 Internship Task

The project is a part of ANR Roii (Rey's Ornament Image Investigation). The ROLi project brings together researchers in the fields of the history of ideas, literature and the history of books (Ihrim) on the one hand, and computer vision and machine learning (Hubert Curien Laboratory) on the other. The aim of this collaborative and interdisciplinary project is to design a tool to help authenticate books published under fictitious or counterfeit names or addresses in the 18th century, through the analysis of ornaments.It is based on the design of a database of ornaments used by the bookseller Marc Michel Rey (1720-1780), the largest and most important publisher in the French language in the United Provinces. The editorial and commercial practices of this French-speaking bookseller based in Amsterdam are indeed particularly representative of both the typographical uses of ornaments and the strategies for circumventing censorship.

The study of these editorial strategies is enlightening in order to understand the book-trade system and the way in which the sharing of knowledge and ideas on a European scale was practised at a time when it was beginning to be defended philosophically. These strategies make it particularly difficult to attribute a work to a publisher and to distinguish between genuine and counterfeit works. The investigation of ornaments then constitutes an additional clue to identify the works, within a cluster of concordant clues. The database is thus associated with an anomaly detection task, combining computer vision and automatic learning. A decision support tool was designed likely to reveal differences in shape at the level of wood ornaments and the publisher's mark, and differences in typographic style at the level of compound ornaments, based on the content of the database's data.
The rest of this report is organized as follows. Deep Learning requires huge set of training images. This report discusses the methods the dataset is created in section 3. In section 4, the algorithms we used to create the network like SSD and Autoencoder are explained. In section 4.2 we explained how we modified SSD to run faster and accurate than usual SSD and how IOU losses can potentially be a loss that object detetctors can use. In section 5 we discuss the DAE and the resutls. Section 6 is for future improvement of DAE.

3 Graphical User Interface

We introduce two graphical interfaces developed in Python for our project.

- **AutoLabelMe**: Automatic Image Annotation
- **Ransac**: Image Alignment

3.1 AutoLabelme

3.1.1 Introduction

Data Collection is a major step in every Machine Learning projects. The quality and quantity of data alone can greatly impact the results of the model. Since we are at the beginning of the project, we had to devise a way to create a dataset for object detection purposes. We have 18 images and all the images are filled with vignettes/Components. Our first job was to create a bounding box around each and every object. So we developed AutoLabelme.
AutoLabelme is an automatic image annotator created using the Tkinter library in Python. It's an extension of LabelMe, the open-source Image Annotator available in https://github.com/wkentaro/labelme. It matches the template provided by the user

in an image and find same objects associating a bounding box and label for every object. Autolabelme uses Normalized Cross-Correlation to check whether two templates are similar. It is designed keeping in mind with the necessity of creating dataset for object detection purposes and currently can only find objects in the cropped image i.e the search space to find same object is the space around the current template.

3.1.2 Properties

- AutoLabelMe is simple to use. After LabelMe Manual Annotation just run AutoLabelMe.py.
- It is fast. We reduced the search space to the neighbourhood of the template which made it efficient for our given project.
- AutoLabelMe can identify rotated, scaled, horizontally and vertically flipped templates. There is no need for manually annotate rotated version of any template. It assigns a new label for the rotated templates with the rotaion information added in the label.
- AutoLabelMe also saces any meta information user stores during the LabelMe Manual Annotation step. For example, if for a bouding box, user assigns its label as "0 This is a Sun". AutoLabelMe will create a separate json file storing the new label for the bounding box and the meta information. This is helpful to save any additional information user wants for any object.

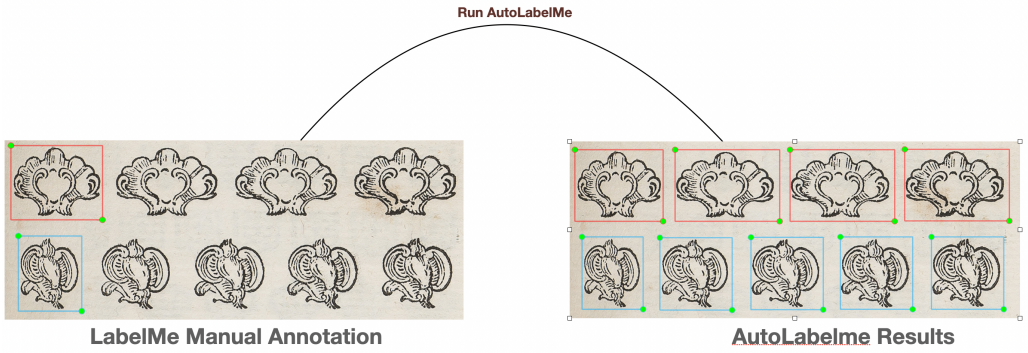


Figure 1: Demonstration of autolabelme results

Figure 2: Autolabelme Interface

3.1.3 Tutorial

Types of Windows:

1. *Left Pane*: Shows the image and matching for the current label. Red for the usual boxes.Blue for horizontally or vertically flipped boxes. Green for rotated. Although these are not final colors of the boxes. These are present only for checking.
2. *Top-Right Pane*: Buttons.
3. *Bottom-Right Pane*: Shows the image with all the matched templates. These image is helpful to recognise if every templates are matched or not. If any is missing, it can be added using LabelMe.

Functions for every Button:-

1. *Next Line* >>: Template matching for next label.
2. << *Previous Line*: Template Matching for the previous label.
3. −: Increases the threshold which results in less number of boxes.
4. +: Decreases the threshold which results in more number of boxes.
5. *Save JSON*: Saves a JSON file which can be read by LabelMe for further edits.
6. *Save Images*: Save the cropped vignettes from the image.

7. *min*: The minimum value for Rotation.
8. *max*: The maximum value for Rotation.
9. *Rematch*: Match again for current label.
10. *Correct Label*: Transform all the boxes to red boxes(no flip).

How to Run:-

1. Open Terminal.
2. Write *labelme* or *python3 /path/to/labelme.py*. Check https://github.com/wkentaro/labelme for more details.
3. Create one bounding box per label.
4. Run AutoLabelme.py *python3 /path/to/AutoLabelme.py*.
5. Open JSON and press *Next Line* >> to start matching.
6. The Left Image will show the boxes for current Label. The smaller right image is for showing all the templates.
7. Press << *Previous Line* for viewing the matched boxes for the previous label.
8. Press + for more boxes and − for less boxes.
9. If you have rotated image, fill the rotation range or just enter *min* value. For example *min*=45 and *max*=90 will give *list(range(45,91,5))* values or just enter *min*=45 which will only rotate the image once (45 degree).
10. The *search space value* is by default 2 which means the algorithm will check for the templates from two heights up to two heights down in the original image. Choose any value from 1 to 15. For example if your template starts from coordinate (200,200) and height and width is 100 and 100 respectively and search space=2, the algorithm will search for the template from (0,0) to (400,400) in the image. If you select more than 15 than it's just *heights-the value*. For example, if you choose 100 in the previous example, the it will be (0,100) to (300,300) where the templates will be searched in the image. In any case, value from 1 to 15 is sufficient.
11. Press *Rematch* button or Press *Enter* or *Return* in your keyboard.
12.(Optional) Sometimes if the template is symmetric, the algorithm picks up some templates as flipped, to fix this, press *Correct Label*.
13. Press *Save JSON* to save a json file.
14. Open the saved json file in Labelme. Labelme will show the matched templates. Edit it if necessary.
15. Press *Save Images* in AutoLabelme if all the boxes are okay. This will save the matched templates in JPEG.

3.2 Ransac

3.2.1 Introduction

The images that are used are taken from books resulting in the projective transformation of the vignettes on the left side. To fix this we use Ransac-Flow[1] (from https://github.com/XiSHEN0220/RANSAC-Flow). It's two-stage image-alignment process, first, a feature-based parametric coarse alignment using one or more homographies, followed by non-parametric fine pixel-wise alignment.Coarse alignment is performed using RANSAC on off-the-shelf deep features. Fine alignment is learned in an unsuper- vised way by a deep network which optimizes a standard structural similarity metric (SSIM) between the two images, plus cycle-consistency.

3.2.2 Tutorial

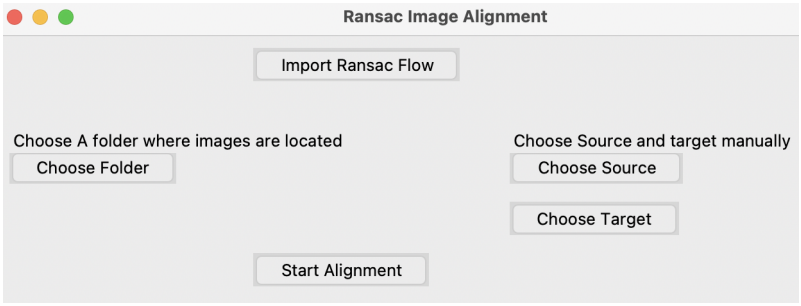


Figure 3: Ransac Interface

Functions of Buttons:

1. *Import Ransac Flow*: Opens Directory to import ransac flow python library. 2. Ransac Flow needs a Reference Image(Target) and a Source image for alignment. There are two ways to import images in Ransac.

- *Choose Folder:* Choose the folder where the images are saved. The last image will be chosen as Reference and rest of them will be chosen as targets.
- *Choose Target*: Choose the target image.
  Choose Source: Choose manually the source image/images.

3. *Start Alignment:* Choose which folder to save. The aligned images are saved.

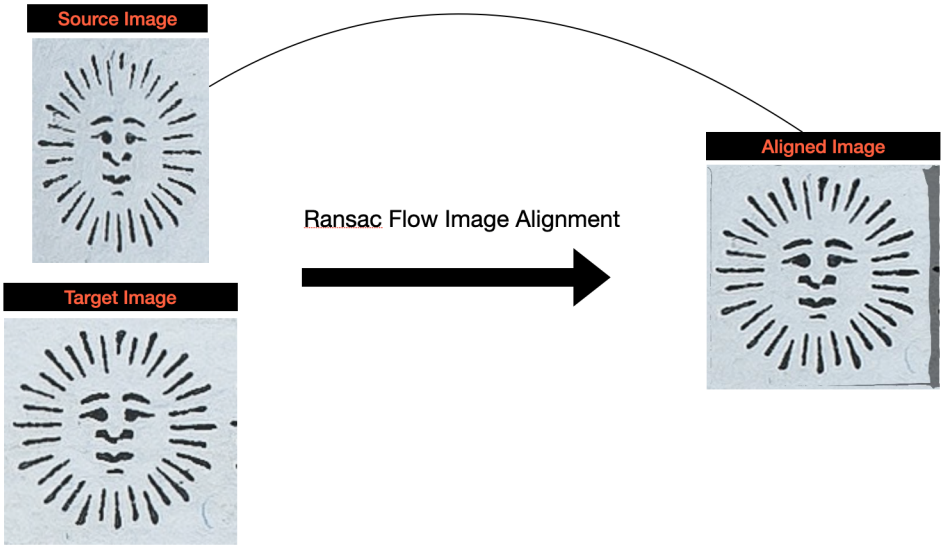


Figure 4: Example of Image alignment in Ransac

4 Models

Our proposed method uses an object detector as an encoder and adds a decoder to reconstruct an image. The objective is to recreate the image with the objects that are not in the catalogue as blurry or not at all. With this process we can detect the anomalies that will lead to any ornaments labelled as fraudulent. To achieve this task, our network should have the following properties:

- The network should be end-to-end trainable which will be possible if the detector is end-to-end trainable.
- The detector needs to be fast and the predicted boxes should be precise. (High mean average precision).

Pre-existing domain-specific image object detectors usually can be divided into two categories, the one is two-stage detector, the most representative one, Faster R-CNN [2]. The other is one-stage detector, such as YOLO [4], SSD[3].
Two-stage detectors have high localization and object recognition accuracy, whereas the one-stage detectors achieve high inference speed. The two stages of two-stage detectors can be divided by RoI (Region of Interest) pooling layer. For instance, in Faster R-CNN, the first stage, called RPN, a Region Proposal Network, proposes candidate object bounding boxes. The second stage, features are extracted by RoIPool (RoI Pooling) operation from each candidate box for the following classification and

bounding-box regression tasks. But two-stage detectors are not fully trainable. Although in FasterRCNN, RPN and FastRCNN can be jointly trained but it results in appoximate Furthermore, the one-stage detectors propose predicted boxes from input images directly without region proposal step, thus they are time efficient and can be used for real-time devices. So we proceed with SSD300 because of its speed and accuracy.[11]

4.1 Single-Shot Multibox Detector(SSD)

SSD is a single-shot object detector for multiple categories that is faster than the previous state-of-the-art for single shot detectors (YOLO), and significantly more accurate, in fact as accurate as slower techniques that perform explicit region proposals and pooling (including Faster R-CNN). SSD is end-to-end trainable and even on low resolution input images, it improves the speed vs accuracy trade-off.
The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), which is called the base network[Fig 5]. Auxiliary structures are added to the network to produce detections with the following features:-

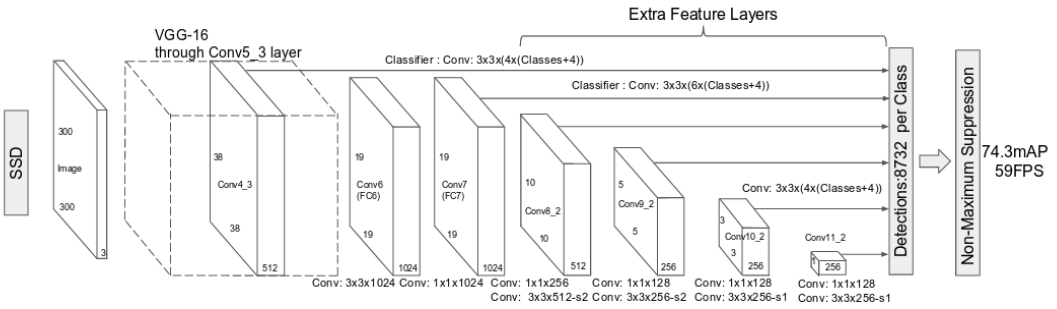


Figure 5: SSD architecture. Base network(vgg16) is truncated at conv5_3 layer and auxiliary network is then added.

1. Multi-Scale Feature Maps

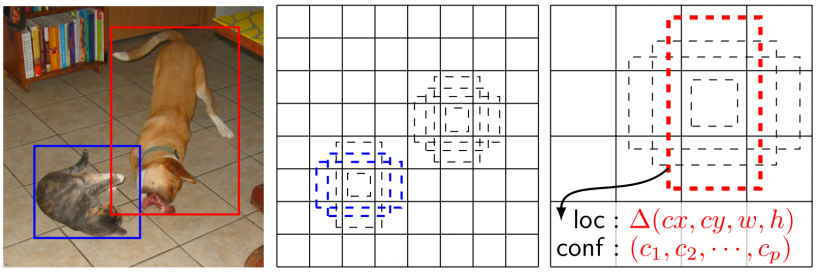
Convolutional Layers are added to the base network which progressively decrease in size and allow predictions of different scales. In Figure 6, it can be seen that in lower

layers bigger objects are captured more precisely and in the upper layers when the feature maps are large, the smaller objects are detected. For example, the following feature maps are selected for detection purposes.

- **Conv4_3:** Size $38 \times 38 \times 512$
- **Conv7:** Size $19 \times 19 \times 1024$
- **Conv8_2:** Size $10 \times 10 \times 512$
- **Conv9_2:** Size $5 \times 5 \times 256$
- **Conv10_2:** Size $3 \times 3 \times 256$
- **Conv11_2:** Size $1 \times 1 \times 256$

2. Default Boxes

For each of the feature maps, for each cell, SSD associates 4 or 6 boxes of different scale and aspect ratios[Figure 6] with objectness score. These are called default boxes which tile the feature map in a convolutional manner so that the position of each box relative to its corresponding cell is fixed.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 6: SSD. Default Boxes

Default boundary boxes are chosen manually. SSD defines a scale value for each feature map layer. Starting from the left, Conv4_3 detects objects at the smallest scale 0.1 (or 0.2 sometimes), and then increases linearly to the rightmost layer at a scale of 0.9.
The scales for every layers are takes as follows:

- **Conv4_3:** 0.1

- **Conv7:** 0.2
- **Conv8_2:** 0.375
- **Conv9_2:** 0.55
- **Conv10_2:** 0.725
- **Conv11_2:** 0.9

Combining the scale value with the target aspect ratios, SSD compute the width and the height of the default boxes. For layers making 6 predictions, SSD starts with 5 target aspect ratios: 1, 2, 3, 1/2, and 1/3 and the layers making 4 predictions, 3 target aspect ratios 1,2,0.5. Then the width and the height of the default boxes are calculated as:

$$w = scale \times \sqrt{aspect\ \ ratio}$$
$$h = \frac{scale}{\sqrt{aspect\ \ ratio}}$$

Then SSD adds an extra default box with scale

$$scale = \begin{cases} \sqrt{scale \times scale\ at\ the\ next\ level} \\ 1\ for\ the\ last\ level \end{cases}$$

Total number of default boxes $38 \times 38 \times 4 + 19 \times 19 \times 6 + 10 \times 10 \times 6 + 5 \times 5 \times 6 + 3 \times 3 \times 4 + 1 \times 1 \times 4$=8732. For each one of the boxes b_{ij}^k (the box associated with $(ij)th$ coordinate of the kth feature map), SSD predicts offsets relative to the default box shapes in the cell and confidence score $P(c_n/b_{ij}^k) \quad \forall n = 1, 2..., num\_classes$ i.e the probability of object class c_n in the box b_{ij}^k. Finally non-maximum suppression is performed to produce the final detection box.

Matching Strategy

SSD predictions are classified as positive matches or negative matches. SSD only uses positive matches in calculating the localization cost (the mismatch of the boundary box). If the corresponding default boundary box (not the predicted boundary box) has an IoU greater than 0.5 with the ground truth, the match is positive. Otherwise, it

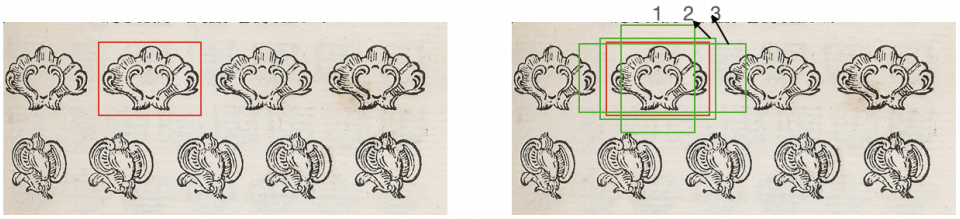


Figure 7: The ground truth box(red) and three default boxes(green)

is negative. (IoU, the intersection over the union, is the ratio between the intersected area over the joined area for two regions.)

For example let's take 3 default boxes. Only default box 1 and 2 (but not 3) have an IoU greater than 0.75 with the ground truth box above (blue box). So only box 1 and 2 are positive matches. Once identify the positive matches are identified, corresponding predicted boundary boxes are used to calculate the cost. This matching strategy nicely partitions what shape of the ground truth that a prediction is responsible for. This matching strategy encourages each prediction to predict shapes closer to the corresponding default box. Therefore the predictions are more diverse and more stable in the training.

3. Loss Function

SSD has two loss functions:- Localization Loss and Classification Loss

Locatization Loss

The localization loss is the mismatch between the ground truth box and the predicted boundary box. SSD only penalizes predictions from positive matches. Since the predictions from the positive matches are desired to get closer to the ground truth. Negative matches can be ignored.
The locatization 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 default bounding box d of width w and height h.

$$\text{Let } O = \{cx, cy, w, h\}$$

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

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w, \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h.$$

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

$$x_{ij}^{p} = \begin{cases} 1 & \text{if } IOU > 0.5 \text{ between default box i and ground truth box j on class p} \\ 0 & Otherwise \end{cases}$$

Confidence Loss

The confidence loss is the loss of making a class prediction. For every positive match prediction, we penalize the loss according to the confidence score of the corresponding class. For negative match predictions, we penalize the loss according to the confidence score of the class "0": class "0" classifies no object is detected.
It's calculated as the softmax loss over multiple classes confidences c(class score)

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{k} log(\hat{c_i}^{p}) - \sum_{i \in Neg}^{N} x_{ij}^{k} log(\hat{c_i}^{0})$$

$$where \ \hat{c_i}^{p} = \frac{exp(c_i^{p})}{\sum_p exp(c_i^{p})}$$

N is the Number of matched default boxes.

Finally, the SSD loss =

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

Hard Mining

However, SSD makes far more predictions than the number of objects present. So there are many more negative matches than positive matches. This creates a class imbalance that hurts training. The SSD model is trained to learn background space rather than detecting objects. However, SSD still requires negative sampling so it can learn what constitutes a bad prediction. So, instead of using all the negatives, we sort those negatives by their calculated confidence loss. SSD picks the negatives with the top loss and makes sure the ratio between the picked negatives and positives is at most 3:1. This leads to faster and more stable training.

SSD makes many predictions (8732) for better coverage of location, scale, and aspect ratios, more than many other detection methods. However, many predictions contain no object. Therefore, any predictions with class confidence scores lower than 0.2 will be eliminated.

Non-Max Suppression

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Figure 8: Results of various detections.SSD makes more predictions. Improvements allow SSD to use lower resolution images for similar accuracy.

SSD uses non-maximum suppression to remove duplicate predictions pointing to the same object. SSD sorts the predictions by the confidence scores. Start from the top confidence prediction, SSD evaluates whether any previously predicted boundary boxes have an IoU higher than 0.45 with the current prediction for the same class. If found, the current prediction will be ignored. At most, top 200 predictions per image are kept.

4.2 GIOU optimized SSD

Bounding box regression is one of the most fundamental components in many 2D/3D computer vision tasks. Tasks such as object localization, multiple object detection, object tracking and instance level segmentation rely on accurate bounding box regression. The dominant trend for improving performance of applications utilizing deep neural networks is to propose either a better architecture backbone or a better strategy to extract reliable local features. However, one opportunity for improvement that is widely ignored is the replacement of the surrogate regression losses such as l_1 and l_2-norms, with a metric loss calculated based on Intersection over Union (IoU).IoU , also known as Jaccard index, is the most commonly used metric for comparing the similarity between two arbitrary shapes. IoU encodes the shape properties of the objects under comparison, e.g. the widths, heights and locations of two bounding boxes, into the region property and then calculates a normalized measure that focuses on their areas (or volumes). This property makes IoU invariant to the scale of the problem under consideration. Due to this appealing property, all performance measures used to evaluate for segmentation, object detection, and tracking rely on this metric.[9]

However, it can be shown that there is not a strong correlation between minimizing the commonly used losses, e.g. $l_n - norms$, defined on parametric representation of two bounding boxes in 2D/3D and improving their IoU values. For example, consider

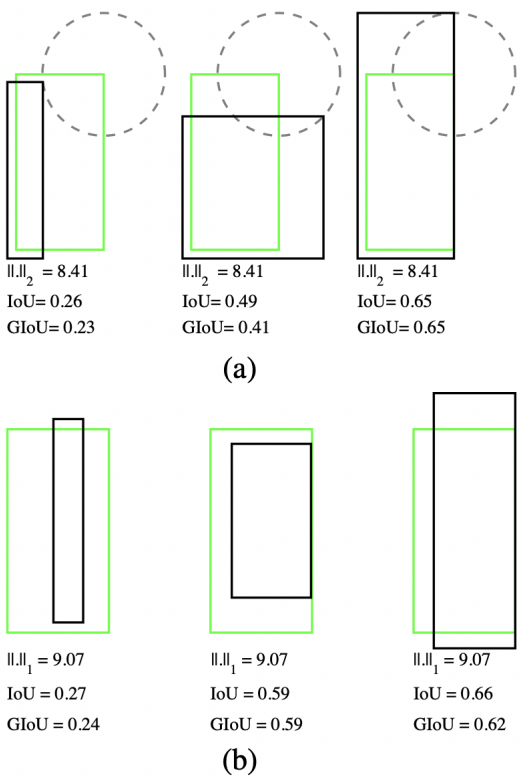


Figure 9: Two sets of examples (a) and (b) with the bounding boxes represented by (a) two corners (x_1, y_1, x_2, y_2) and (b) center and size (x_c, y_c, w, h). For all three cases in each set (a) l_2- norm distance, $||.||_2$, and (b) l_1-norm distance, $||.||_1$, between the representation of two rectangles are exactly same value, but their IoU and GIoU values are very different.

the simple 2D scenario in Figure. 9 (a), where the predicted bounding box (black rectangle), and the ground truth box (green rectangle), are represented by their top-left and bottom-right corners, i.e. (x_1, y_1, x_2, y_2). For simplicity, let's assume that the distance, e.g. l2-norm, between one of the corners of two boxes is fixed. Therefore any predicted bounding box where the second corner lies on a circle with a fixed radius centered on the second corner of the green rectangle (shown by a gray dashed line circle) will have exactly the same l_2-norm distance from the ground truth box; however their IoU values can be significantly different (Figure. 9 (a)). The same argument can be ex- tended to any other representation and loss, e.g. Figure. 9 (b). It is intuitive that a good local optimum for these types of objectives may not necessarily be a local optimum for IoU. Moreover, in contrast to IoU , l_n -norm objectives defined based on the aforementioned parametric representations are not invariant to the scale of the

problem. To this end, several pairs of bounding boxes with the same level of overlap, but different scales due to e.g. perspective, will have different objective values. In addition, some representations may suffer from lack of regularization between the different types of parameters used for the representation. For example, in the center and size representation, (x_c, y_c) is defined on the location space while (w, h) belongs to the size space. Complexity increases as more parameters are incorporated, e.g. rotation, or when adding more dimensions to the problem. To alleviate some of the aforementioned problems, state-of- the-art object detectors introduce the concept of an anchor box as a hypothetically good initial guess. They also define a non-linear representation to naively compensate for the scale changes. Even with these handcrafted changes, there is still a gap between optimizing the regression losses and IoU values.

4.2.1 IOU Loss

Generally, for two finite sample sets A and B, their IoU is defined as the intersection $(A \cap B)$ divided by the union $(A \cup B)$ of A and B.

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cup B|}$$

For bounding box-level object detection, the target object is usually represented by a minimum Bbox rectangle in the 2D image. Base on this representation, the IoU computation between the ground bounding box $B_g = (x_1, y_1, x_2, y_2)$ and the predicted bounding box $B_d = (x'_1, y'_1, x'_2, y'_2)$ is defined as:-

$$IoU(A, B) = \frac{\text{Area of overlap between } B_g \text{ and } B_d}{\text{Area of union of } B_g \text{ and } B_d} = \frac{(max(x_1, x'_1) - min(x_2, x'_2)) \times (max(y_1, y'_1) - min(y_2, y'_2))}{(x_2 - x_1)(y_2 - y_1) + (x'_2 - x'_1)(y'_2 - y'_1) - (max(x_1, x'_1) - min(x_2, x'_2))(max(y_1, y'_1) - min(y_2, y'_2))}$$

Usually, objects are labeled with axis-aligned BBoxes in the ROI dataset. By taking this kind of labels as ground truth, the predicted BBoxes are also axis-aligned rectangles. For this case, the IoU computation is very easy.

Loss function:

The IOU loss function[7] for the ground bounding box $B_g = (x_1, y_1, x_2, y_2)$ and the predicted bounding box $B_d = (x'_1, y'_1, x'_2, y'_2)$ is defined as

$$L_{IoU} = 1 - IoU(B_g, B_d)$$

- Since $0 \leq IoU \leq 1, 0 \leq L_{IoU} \leq 1$. So L_{IoU} is non-negative. $L_{IoU} = 0$ when $IoU(A, B) = 1 \implies A = B$.
- $IoU(A, B) = IoU(B, A) \implies L_{IoU}(A, B) = L_{IoU}(B, A)$. So L_{IoU} is symmetric.
- L_{IoU} satisfies triangle inequality.[8]

So L_{IoU} is a metric by mathematical definition.

Differentiability of IOU Loss:

IOU loss is differentiable and can be backpropagated.

Let $B_g = \{x_1, y_1, x_2, y_2\}$ be the ground truth and $B_d = \{x'_1, y'_1, x'_2, y'_2\}$ be the predicted bounding box.

$X =$ Area of $B_g = (x_2 - x_1) \times (y_2 - y_1)$

$X' =$ Area of $B_d = (x'_2 - x'_1) \times (y'_2 - y'_1)$

$I = (max(x_1, x'_1) - min(x_2, x'_2)) \times (max(y_1, y'_1) - min(y_2, y'_2))$

$L_{IoU} = 1 - \frac{I}{X + X' - I} = 1 - \frac{I}{U}$, where $U = X + X' - I$

$\frac{\partial L}{\partial x'} = \frac{I(\Delta_{x'} X - \Delta_{x'} I) - U \Delta_{x'} I}{U^2}$

$\frac{\partial X}{\partial x'_1} = -(y'_2 - y'_1), \frac{\partial X}{\partial x'_2} = (y'_2 - y'_1), \frac{\partial X}{\partial y'_2} = (x'_2 - x'_1), \frac{\partial X}{\partial y'_1} = -(x'_2 - x'_1)$

$$\frac{\partial I}{\partial x'_1} = \begin{cases} (max(y_1, y'_1) - min(y_2, y'_2)) & \text{if } x'_1 > x_1 \\ 0 & Otherwise \end{cases}$$

$$\frac{\partial I}{\partial x'_2} = \begin{cases} -(max(y_1, y'_1) - min(y_2, y'_2)) & \text{if } x'_2 < x_2 \\ 0 & Otherwise \end{cases}$$

$$\frac{\partial I}{\partial y'_1} = \begin{cases} (max(x_1, x'_1) - min(x_2, x'_2)) & \text{if } y'_1 > y_1 \\ 0 & Otherwise \end{cases}$$

$$\frac{\partial I}{\partial y'_2} = \begin{cases} -(max(x_1, x'_1) - min(x_2, x'_2)) & \text{if } y'_2 < y_2 \\ 0 & Otherwise \end{cases}$$

So L_{IoU} can be directly used as the objective function to optimize. It is therefore preferable to use IoU as the objective function for 2D object detection tasks. Given the choice between optimizing a metric itself vs.a surrogate loss function, the optimal choice is the metric itself.

4.2.2 GIOU Loss

However, IOU has two major issues as a metric and loss function.

- If two boxes don't overlap, then their IoU is zero, which doesn't give any indication about the proximity of the two boxes.
- In case of non-overlapping boxes, since their Iou is zero, the gradient is also zero. So L_{IoU} can't be optimized.

Generalized IoU (GIoU) addresses these weaknesses of IoU and (a) follows the properties of IoU i.e the encoding the shape properties of the compared objects into the region property; (b) maintains the scale invariant property of IoU, and (c) ensures a strong correlation with IoU in the case of overlapping objects.

Loss Function

The Loss function for the Generalized IoU is defined as follows:-

Let A and B be two boxes.
Let C be the smallest enclosing box

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

$$GIoU = IoU - \frac{|C \setminus (A \cup B)|}{|C|} L_{GIoU} = 1 - GIoU$$

Some of the properties of GIoU:-

- Similar to L_{IoU}, L_{GIoU} is also non-negative, symmetric and satisfies triangle inequality. So L_{GIoU} is a metric.
- GIoU is a lower bound for IoU. $\forall A, B, GIoU(A, B) \leq IoU(A, B)$ and this lower bound becomes tighter when A and B have a stronger shape similarity $i.e$ $\lim_{A \to B} GIoU(A, B) = IoU(A, B)$.
- $0 \leq IoU(A, B) \leq 1 \implies -1 \leq GIoU(A, B) \leq 1$.
- $GIoU = 1$ when $|A \cap B| = |A \cup B|$ $i.e$ when A and B completely overlaps.
- GIoU tends to -1 as the ratio between occupying regions $|A \cup B|$ and the smallest enclosing box C goes to zero, i.e $\lim_{\frac{|A \cup B|}{C} \to 0} GIoU(A, B) = -1$
- L_{GIoU} is differentiable.

- When IoU=0, i.e boxes don't overlap, $L_{GIoU} = 2 - \frac{|A \cup B|}{|C|}$. By minimizing L_{GIoU}, we are maximizing $\frac{|A \cup B|}{|C|}$ $(0 \leq \frac{|A \cup B|}{|C|} \leq 1)$ which means we are maximizing the region of union $|A \cup B|$ and minimizing the enclosing box area $|C|$, which will be possible if the predicted box goes to the

4.2.3 Optimized SSD

We improved SSD in two ways:-

- **Fully Trainable:** Instead of using l_1 loss, we used GIoU Loss which makes the detector-encoder Autoencoder(DAE) fully trainable.
- **Faster SSD:** SSD uses 4 or 6 default boxes per cell of the feature map which makes total 8732 boxes. Since we don't have any overlapping objects and also number of objects in every image are less than 100, we used 1 box per cell of the feature map, which gives significant reduction in training time(see figure 10) and also maintains the accuracy and box precision. The aspect ratio was kept at 2. The width and the height of the box are,

$$w = scale \times \sqrt{2}$$
$$h = \frac{scale}{\sqrt{2}}$$

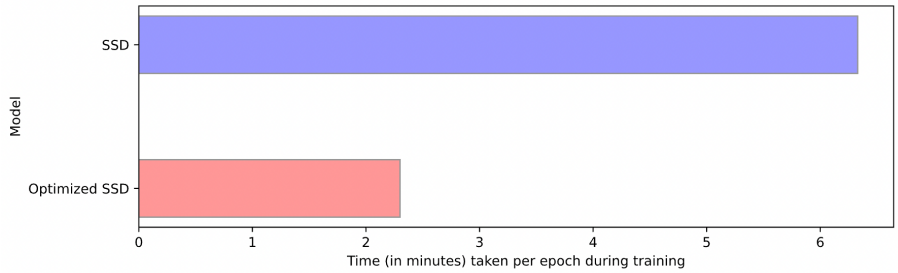


Figure 10: Time taken during training for an epoch

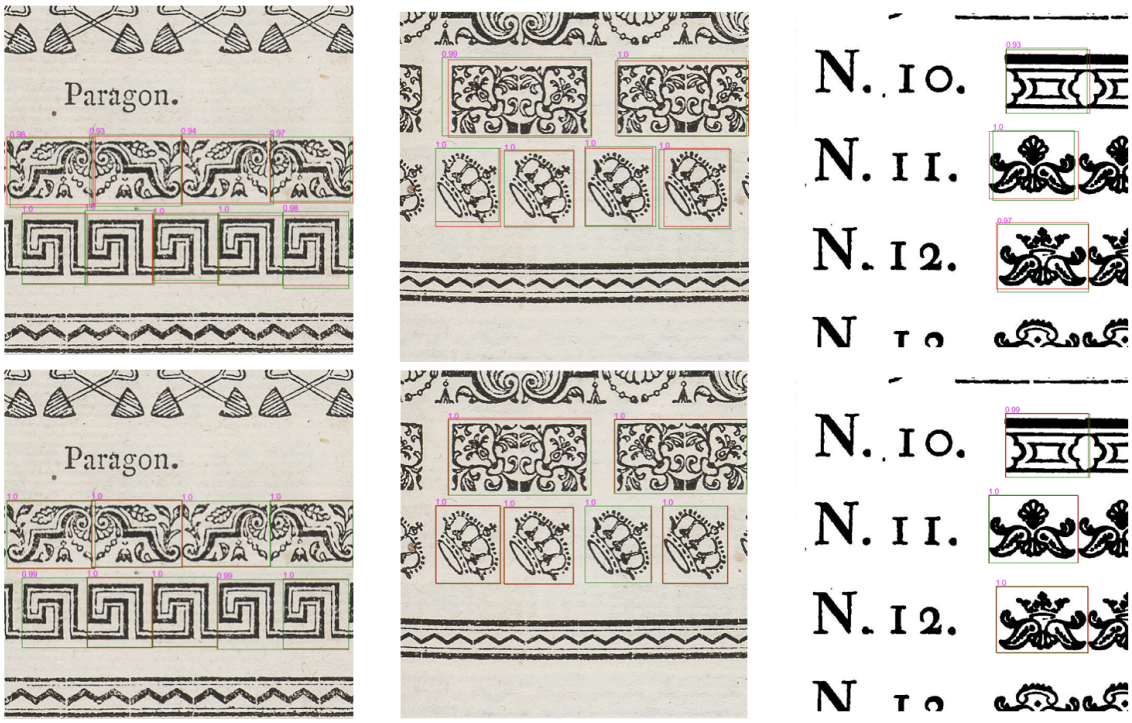


Figure 11: Optimized SSD results on the catalogue. The top row is SSD with l_1 loss and the bottom row is Optimized-SSD with GIoU loss. The red boxes are ground truth boxes and green boxes are predicted bounding boxes. Optimized-SSD has more precision in bounding boxes than SSD with l_1. The pink numbers are confidence score for the box.

4.3 MaskRCNN

Mask R-CNN [10] is an extending work to Faster R-CNN[2] mainly for instance segmentation task. Regardless of the adding parallel mask branch, Mask R-CNN can be seen a more accurate object detector. He et al. use Faster R- CNN with a ResNet [25]-FPN [15] (feature pyramid network, a backbone extracts RoI features from different levels of the feature pyramid according to their scale) backbone to extract features achieves excellent accuracy and processing speed. FPN contains a bottom-up pathway and a top-down pathway with lateral connections. The bottom-up pathway is a backbone ConvNet which computes a feature hierarchy consisting of feature maps at several scales with a scaling step of 2. The top-down pathway produces higher resolution features by upsampling spatially coarser, but semantically stronger, feature maps from higher pyramid levels. At the beginning, the top pyramid feature maps are captured by the output of the last convolutional layer of the bottom-

up pathway. Each lateral connection merges feature maps of the same spatial size from the bottom-up pathway and the top-down pathway. While the dimensions of feature maps are different, the 1×1 convolutional layer can change the dimension. Once undergoing a lateral connection operation, there will form a new pyramid level and predictions are made independently on each level. Because higher-resolution feature maps are important for detecting small objects while lower-resolution feature maps are rich in semantic information, feature pyramid network extracts significant features. Another way to improve accuracy is to replace RoI pooling with RoIAlign to extract a small feature map from each RoI, as shown in Figure 12. Traditional RoI pooling quantizes floating number in two steps to get approximate feature values in each bin. First, quantization is applied to calculate the coordinates of each RoI on feature maps, given the coordinates of RoIs in the input images and down sampling stride. Then RoI feature maps are divided into bins to generate feature maps at the same size, which is also quantized during the process. These two quantization operations cause misalignments between the RoI and the extracted features. To address this, at those two steps, RoIAlign avoids any quantization of the RoI boundaries or bins. First it computes the floating number of the coordinates of each RoI feature map followed by a bilinear interpolation operation to compute the exact values of the features at four regularly sampled locations in each RoI bin. Then it aggregates the results using max or average pooling to get values of each bin.

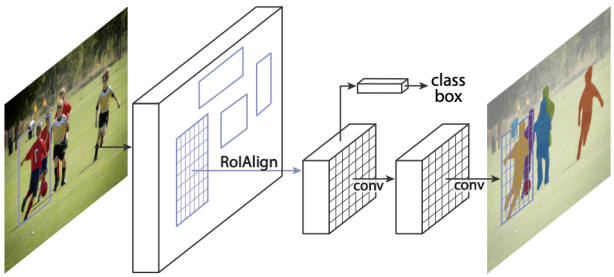


Figure 12: Mask RCNN architecture

4.4 AutoEncoder

Autoencoders are an unsupervised learning technique in which neural networks are trained for the task of representation learning. The network can be viewed as consisting of two parts: an encoder function Specifically, a neural network architecture is designed such that a bottleneck is imposed in the network which forces a compressed knowledge representation of the original input. If the input features were each in-

dependent of one another, this compression and subsequent reconstruction would be a very difficult task. However, if some sort of structure exists in the data (ie. correlations between input features), this structure can be learned and consequently leveraged when forcing the input through the network's bottleneck.

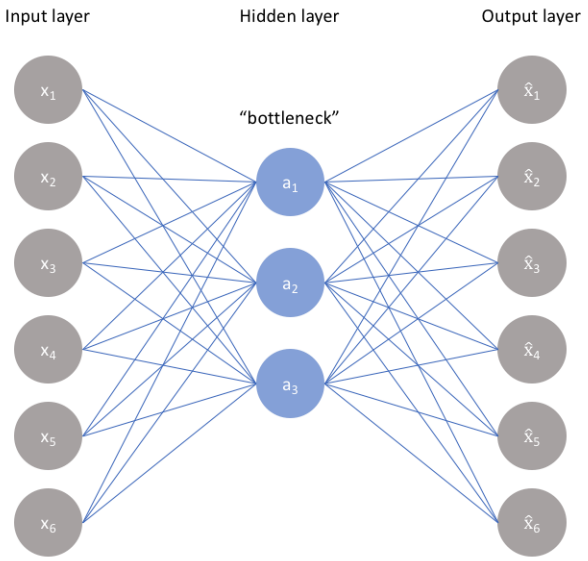


Figure 13: Autoencoder: Bottlenecks are provided as to force the network to learn the representation.

As in Figure 13, the problem can be transformed into a supervised learning tasked with outputting $\hat{x}$, a reconstruction of the input x. The network can be trained to minimize the reconstruction error $L(x, \hat{x})$, the differences between the training input and the reconstructed image.The bottleneck is a key attribute of the autoencoder network design without the presence of an information bottleneck, the network could easily learn to simply memorize the input values by passing these values along through the network. The ideal autoencoder model balances the following:

- Sensitive to the inputs enough to accurately build a reconstruction.
- Insensitive enough to the inputs that the model doesn't simply memorize or overfit the training data. Because of this, AutoEncoders are used for Anomaly Detection.

Convolutional Autoencoders

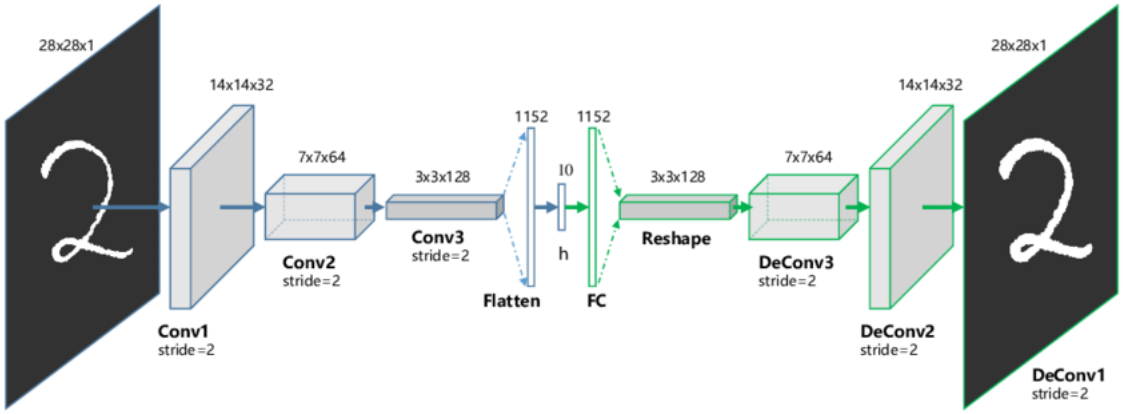


Figure 14: Convolutional Autoencoders

Modeling image data requires a special approach in the neural network world. The best known neural network for modeling image data is the Convolutional Neural Network (CNN, or ConvNet). It can better retain the connected information between the pixels of an image. The particular design of the layers in a CNN makes it a better choice to process image data.
The CNN design can be used for image recognition/classification or be used for image noise reduction or coloring. CNN also can be used as an autoencoder for image noise reduction or coloring or anomaly detection.
When CNN is used for anomaly detection, it is applied in an Autoencoder framework, i.e, the CNN is used in the encoding and decoding parts of an autoencoder. Figure 14 and 15 show an CNN autoencoder. Each of the input image samples is an image, and each of the output image samples is the corresponding image. If the input image is an anomaly, the output image is blurred and the reconstruction error is large (Figure 15).

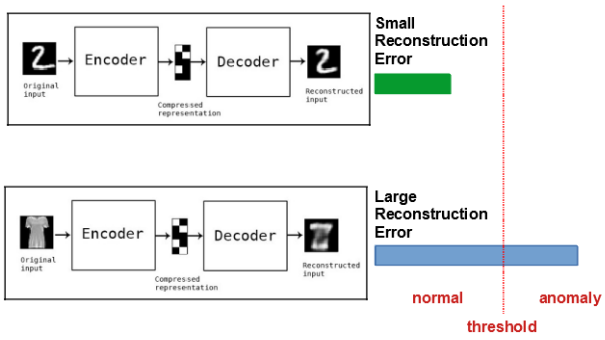


Figure 15: Convolutional Autoencoders for anomaly detection

5 Contribution

5.1 Data Preparation

We have 18 images consisting of components/vignettes(see Figure 16). All the images are of high resolution ($\sim$ 3000$\times$ $\sim$ 2000) and for detection, we have used SSD300 which takes as input $300 \times 300 \times 3$ images. Upon resizing the image from its high

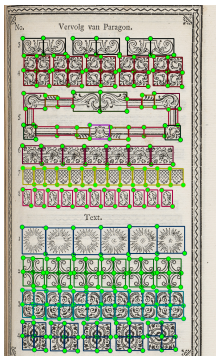


Figure 16: Example of an image.

resolution to SSD input size, a lot of information is being lost and SSD performs poor. To address this issue, 100 random crops are taken of 640×640 size from each of the 18 images. During resizing phase, if an object has less than 75% visibility, we disregard the box for that object.

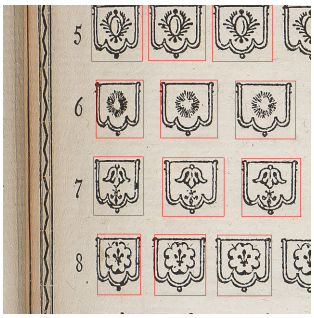


Figure 17: On the top right and bottom right two boxes are disregarded.

5.2 Training SSD

We used the pytorch implementation of the SSD (Kaggle SSD 300) with some modification. We use Google Colab with 12 GB of NVIDIA Tesla K80 GPU. For baseline results, smooth l1 loss was used first and then L_{GIoU} and L_{IoU} was used. The learning rate was initialized at 0.0001 and was decreased by factor $\frac{1}{10}$ when the loss didn't decrease more than 2%. For optimization, Adam optimizer was used.

5.3 DAE: Decoder-Detector AutoEncoder

The idea of Decoder-Detector AutoEncoder was motivated by the concept of Mask-RCNN[10]. In maskrcnn, the mask branch is responsible for the segmentation. In DAE, the decoder branch is resposible for reconstruction of the vignettes.(See Figure 18)

5.3.1 Encoder

In AutoEncoder, the encoder learns to encode the image from which decoder reconstructs the image. Our goal is to detect the vignettes which are anomaly in an ornament. The goal of DAE is to reproduce the ornaments input image with a output image that has blurry or no vignettes which are anomaly. In this way, DAE can tell which of the vignettes are anomaly and which are not. But AutoEncoder doesn't detect and if we only use autoencoder, then we will not be able to know which vignettes are anomaly. To address this issue, we replaced the encoder with SSD. SSD is trained with the catalogue of vignettes and if in an ornament, a vignette has an anomaly, it won't be detected. SSD has several feature maps for the purpose of detection. All these feature maps encode information about the image as well as their spatial information. In each one of the feature maps, the detected boxes specify the position of the object in original image. The spatial information are the detected boxes which

we call **ROI**(Region of Interest). But these ROI's are of different shape and aspect ratios. So we used ROI-Alignment to make all the ROIs of the same shape. This is the encoded information about the vignettes which we can use.

5.3.2 Decoder

The Decoder consists of one input layer, two deconvolutional layers and one output layer. The information about the layers are as follows:(See in Figure 19)

- Input Layer: Input Size ($5 \times 5 \times 1024$), Kernel Size (5×5).
- Deconvolutional Layer: Input Size ($14 \times 14 \times 256$), Kernel Size (5×5).
- Deconvolutional Layer: Input Size ($32 \times 32 \times 32$), Kernel Size (5×5).
- Output Layer: Output Size ($64 \times 64 \times 3$).

5.3.3 Data Preparation

We first train the Optimized-SSD to detect the vignettes. The output of the SSD is the offsets for each of the each of the 1940 prior boxes and the probability score for every class. Most of the boxes are of the background classes. Through the indices of the boxes, we can get the information about which feature maps they are from. Then we crop the feature maps with the boxes that have objects in it and use ROI Alignment for transforming the cropped feature maps into same size (For example 5×5. All the feature maps in the SSD don't have same size. The largest is $38 \times 38 \times 1024$ and the smallest is $3 \times 3 \times 256$ (We removed the Conv11_2 since it's $1 \times 1 \times 256$. If we get cropped feature maps from other feature maps, we simply upsample it to 1024. If SSD detects n objects, then we get $n \times 1024 \times 5 \times 5$.

5.3.4 Training

The code is written in Pytorch and it's available in Internship Hubert Curien 2021. We use the same Google Colab platform for training as Section 5.2. The loss function is $l_2 loss$ and optimizer was Adam with learning rate=0.0001.
The results are in Figure 20.

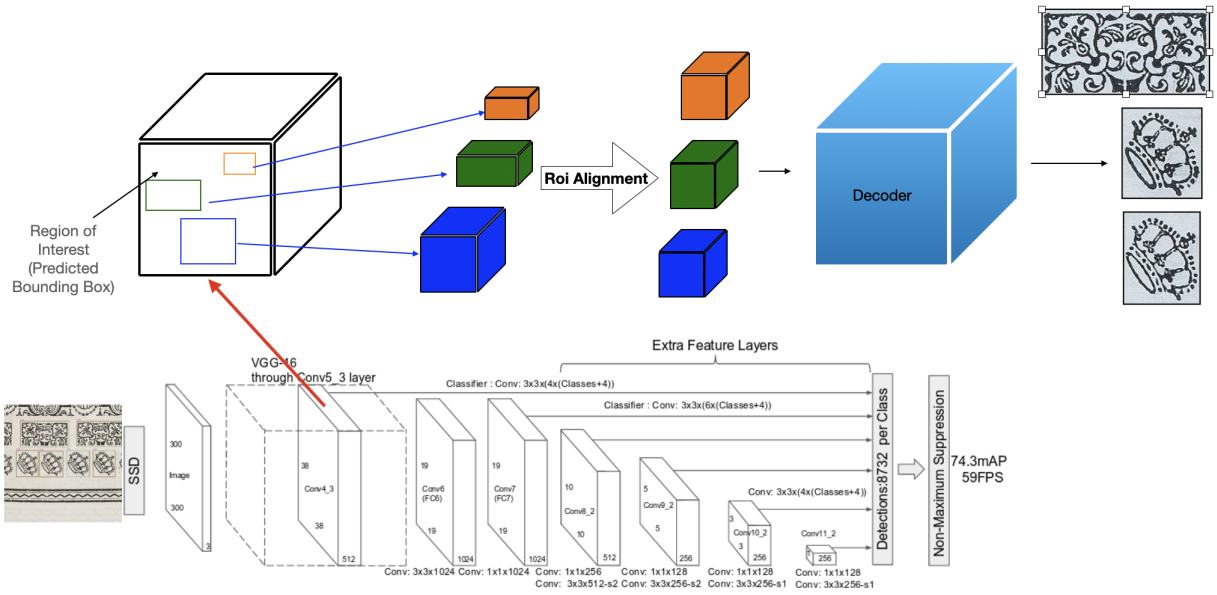


Figure 18: DAE Architecture. The top row is the Decoder branch. The Bottom row is the SSD detector whose feature maps are used for representation learning.

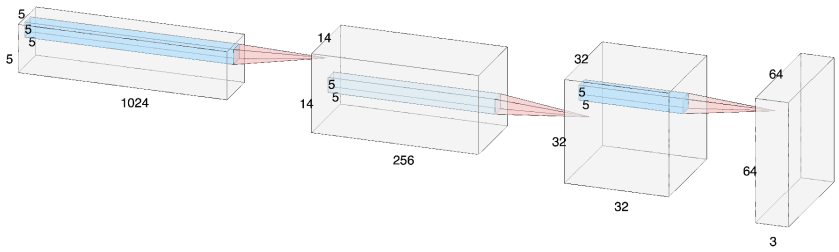


Figure 19: Decoder in DAE.

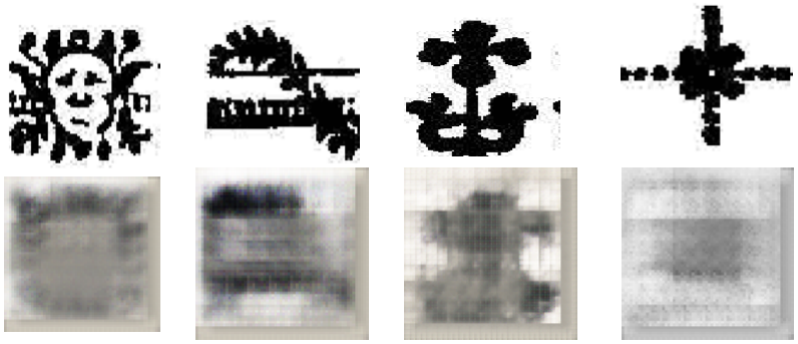


Figure 20: On the top row are SSD detected objects and on the bottom row are the reconstruction of those objects.

6 Future Work

DIOU Loss:

GIOU loss has some limitations:-

- GIoU loss intends to increase the size of predicted box at first, making it have overlap with target box, and then the IoU term in in L_{GIOU} will work to maximize the overlap area of bounding box.
- GIoU loss will totally degrade to IoU loss for enclosing bounding boxes. Due to heavily relying on the IoU term, GIoU empirically needs more iterations to converge, especially for horizontal and vertical bounding boxes. Usually GIoU loss cannot well converge in the state-of-the-art detection algorithms, yielding inaccurate detection.

So, we trained the SSD with DIOU loss and found the following results on test dataset.(see Table 1) We use the standard evaluation metric average precision(AP). We took AP for $IoU = 0.75, 0.8, 0.85, 0.9, 0.95, 0.99$. Since the DAE depends heavily upon the region of interests of the detector, so the AP for the higher IoU is more desirable.

Self Supervised DAE

The DAE will be improved to make better reconstructed image and we will make the DAE self-supervised without any need for detector. Patch detection and auto-encoding will be developed to be confronted or combined in order to deliver tangible

Table 1: Average Precision for the training dataset

Mean Average Precision						
Model	$AP_{0.75}$	$AP_{0.8}$	$AP_{0.85}$	$AP_{0.9}$	$AP_{0.95}$	$AP_{0.99}$
SSD_{GIOU}	0.62	0.48	0.288	0.081	0.007	1.25×10^{-5}
SSD_{DIOU}	0.647	0.54	0.3713	0.1328	0.0125	0.00057
Relative Increase	4.35%	12.5%	28.9%	64%	78.5%	4460%

visual elements on demand, with patch analysis approximating the mechanisms implemented during a visual comparison of facing images, the auto-encoders producing an efficient representation of the data in unsupervised mode. This tool will be blindly tested on collections of ornamental images attributed to Marc Michel Rey, but also on fakes recognized by experts, before being tested online and in open access, and eventually transferred from Marc Michel Rey's collection to other heritage collections.

7 Conclusions

In this paper, we introduced two novel ideas.(1) DAE, detector-encoder AutoEncoder is a novel approach in anomaly detection. By using this approach, we not only detect fake ornaments but also the part which is an anomaly. The advantage of the DAE over any detector is that it can be self-supervised forcing DAE to learn the sub-patterns for which an ornament is true. (2) AutoLabelme is an automatic image annotator, which can be used for creating dataset for object detection purposes. By reducing the image size and creating more user-friendly GUI, Autolabelme can be very fast making it very helpful for researchers and practitioners alike.

References

[1] Xi Shen, François Darmon, Alexei A. Efros, Mathieu Aubry. *RANSAC-Flow: generic two-stage image alignment.*.

[2] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.*

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. *SSD: Singe Shot Detector.*

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection.*

[5] Joseph Redmon, Ali Farhadi. *YOLO9000: Better, Faster, Stronger.*

[6] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, Silvio Savarese. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression.*

[7] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, Ruigang Yang *IoU Loss for 2D/3D Object Detection*

[8] Sven Kosub *A note on the triangle inequality for the Jaccard distance*

[9] Yuanzhou Yao, Yuhang Yang, Xinyue Su, Yihang Zhao, Ao Feng, Yiting Huang, Haibo Pu *Optimization of the Bounding Box Regression Process of SSD Model*

[10] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick. *Mask RCNN.*

[11] Licheng Jiao, Fellow, IEEE, Fan Zhang, Fang Liu, Senior Member, IEEE, Shuyuan Yang, Senior Member, IEEE, Lingling Li, Member, IEEE, Zhixi Feng, Member, IEEE, and Rong Qu, Senior Member, IEEE. *A Survey of Deep Learning-based Object Detection.*

[12] *SSD object detection: Single Shot MultiBox Detector for real-time processing.*