Single Shot MultiBox Detector(SSD300) - Facade Parsing

Ilija Gjorgjiev - ilija.gjorgjiev@epfl.ch
Advisor: Radhakrishna Achanta
Swiss Data Science Center, EPFL Lausanne, Switzerland

***Abstract*—Deep learning has become a widely used field in today's world. By using deep learning and the computational power disposable at our hands, we can easily automate complicated processes. In this paper we tackle the problem of object detection, specifically facade parsing by using the Single Shot MultiBox Detector architecture[1]. The goal is develop a tool that would automate the process of facade parsing and therefore make life easier for Civil Engineers. In the report, we discuss the SSD300 architecture, its usage and performance in the facade parsing problem.**

I. Introduction

This project is a collaboration between the Civil Engineers and the Swiss Data Science Center(SDSC) at EPFL. The motivation behind the project is to help Civil Engineers in detecting the damage imposed on buildings by an earthquake. In this semester project, only a sub-part of the whole project has been tackled. By using deep learning, we automate the detection of important objects in an given image. In our case, objects of importance are defined as $\{$*buildings, doors, windows*$\}$.

The objectives are to use a suitable deep learning method to do these detections, and to make it reusable such that it could be merged or upgraded, if needed. The method chosen for doing the detections in our paper is the **Single Shot MultiBox Detector**(SSD)[1]. The semester project consisted of 5 parts:

1) Getting familiar with the **SSD** structure.
2) Generating suitable data used to train the SSD.
3) Implementing the SSD.
4) Apply the SSD on our dataset and obtain results.
5) Document the code, turn it into a library and write a report.

II. Model

A. Single Shot MultiBox Detector

In this subsection we describe the structure of an SSD300[1]. The function of an SSD is to perform object detection, which consists of 2 stages:

1) ***Object Localization***
2) ***Object Type Classification***

Earlier architectures used to tackle the object detection problem in 2 stages, making it computationally expensive and ill-suited for the real world. However, the SSD approach is based on a feed-forward[2] convolutional network encapsulating both localization and classification, which produces a fixed-size collection of bounding boxes and scores for the presence of objects' label for each respective box. By applying non-maximum suppression on the fixed-size collection, the SSD produces the final detections. The SSD300 structure is organized in 3 parts, see Figure 1:

1) **Base convolutions**: used to provide lower-level feature maps[3]. Helpful for detecting smaller objects.
2) **Auxiliary convolutions**: put on top of the base convolutions, used to provide higher-level feature maps[4]. Helpful for detecting bigger objects.
3) **Prediction Convolutions**: utilized to locate and classify objects from specific

intermediate layers found in the base and auxiliary convolutions.

For computing the actual predictions, the SSD architectures generates and utilizes a pre-calculated set of default bounding boxes with varying aspect ratios[5], [6] set at each feature map cell. The default bounding boxes are called prior boxes and they represent the universe of probable and approximate box predictions.

B. Base Convolutions - SSD300

As a base, we used an already known architecture **VGG-16**[7], though other networks are expected to produce good results, too. We use the VGG-16 architecture since it has already been proven to work nicely with images, and is able to capture the information we need from the image. In the original paper[1], it is recommended to use an already pretrained VGG-16. In the PyTorch library there is a VGG-16 that is trained on *ImageNet Large Scale Visual Recognition Competition (ILSVRC)*, and that is the one that we are using. The input of an SSD300 is an image of dimensions-$(300, 300)$, while the original architecture of VGG16 accepts $(224, 224)$. To be able to actually use the VGG-16 in our object detection task, we need to do some small modifications to its sturcture. Modifications applied are the following:

- Fully Connected Layer 8 is removed.
- Fully Connected Layer 6 and 7 are altered to Convolutional Layer 6 and 7, and their input and output channels are reshaped, see Figure 1.
- The **3rd** pooling layer used for halving dimensions, will use the ceil function instead of floor, for determining output size.
- The **5th** pooling layer will be modified to a kernel of size $(3, 3)$ and stride 1.

C. Auxiliary Convolutions

The auxiliary convolutions are added on top of the base convolutions. They provide feature maps, with each one being smaller than the previous, helpful in the detection of bigger

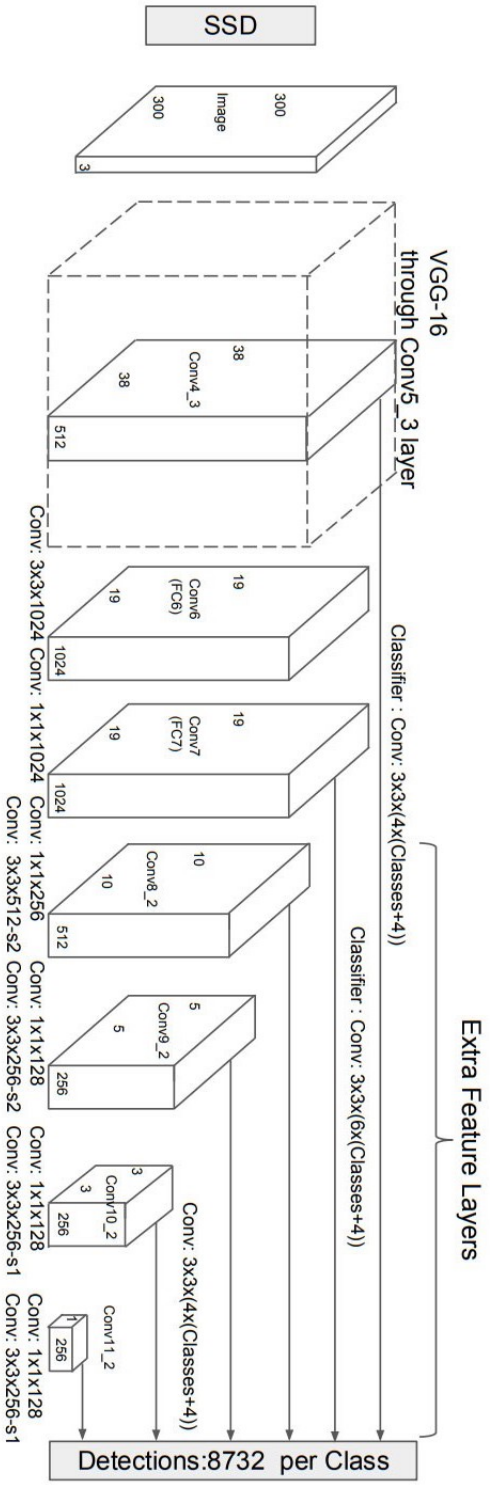

Figure 1: SSD300 Structure

objects. Explicitly four additional convolution blocks are added, each having 2 layers, see Figure 1.

D. Prediction Convolutions & Prior Boxes

Prior Boxes play the main role in computing the predictions. The key concept of the SSD architecture is to take the intermediate layers from base and auxiliary convolutions as feature maps. Convolution filters of kernel size 3x3 are then run on these feature maps in order to classify and predict, the offset to the already pre-computed prior boxes. Each position in each feature map has 4 or 6 prior boxes respectively. There are more prior boxes in the lower layers as they are smaller. The generation of the prior boxes is executed before building and running the SSD, and their number is fixed. Explicitly for the SSD300 the number of prior boxes is defined to be 8372. The aspect ratios and scales of prior boxes are chosen carefully as they should represent sizes and shapes of the ground truth objects. When talking about prior boxes and bounding boxes a couple of different representations will be used:

- $(x_{min}, y_{min}, x_{max}, y_{max})$ - which represent the minimum and maximum points found on x and y axis for that specific prior/bounded box.
- (c_x, c_y, w, h) - which represent center-size coordinates with $c_x = \frac{x_{max}+x_{min}}{2}$ and $c_y = \frac{y_{max}+y_{min}}{2}$. Where $\boldsymbol{w}$ the width and $\boldsymbol{h}$ represents the height of the box.
- $(g_{cx}, g_{cy}, g_w, g_h)$ - defined as the offset of a prior box with respect to a default box, see Figure 2 on how the prior box and the bounding box are defined. $g_{cx} = \frac{c_x - \tilde{c}_x}{\tilde{w}}$, $g_{cy} = \frac{c_y - \tilde{c}_y}{\tilde{h}}$, $g_w = log(\frac{w}{\tilde{w}})$ and $g_h = log(\frac{h}{\tilde{h}})$.

Each prior serves as the starting point for making a prediction and how much it needs to be adjusted for making a more accurate prediction for a bounding box. Therefore, for each prior box at each location for each feature map, the prediction convolutions predict:

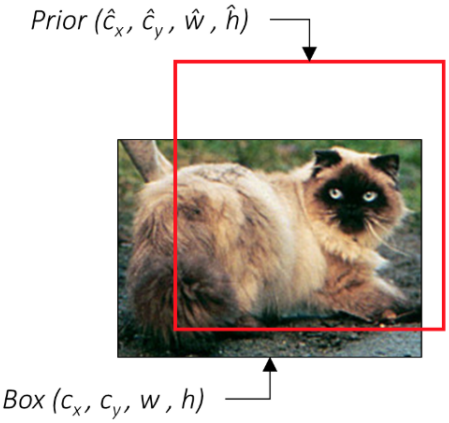

Figure 2: Prior Box and Bounding Box in Center Size Coordinates

- The offsets $(g_{cx}, g_{cy}, g_w, g_h)$ for a bounding box with respect to a prior boxes.
- Probability of belonging to a certain class, for the bounding box.

E. MultiBox Loss Function

As our model has a pretty unique output, we will need a pretty unique loss function for evaluation of how good the model is doing. As mentioned in subsection II-D, the model predicts both the class and location, therefore we are dealing with both regression and classification tasks. Therefore, we use the MultiBox Loss[8], [9] function.

Matching: The essence of every loss function, during training is to match the predicted results from the model to the corresponding ground truths. We begin by matching each ground truth object to the prior box with the best jaccard overlap, with jaccard overlap being defined as:

$$Jaccard\ Overlap = \frac{Box_A \cap Box_B}{Box_A \cup Box_B}$$

We then match each prior box to the best ground truth object with a jaccard overlap higher than a threshold in our case 0.5. If the threshold is not satisfied the prior box is set to be *background*. By doing this, the

problem has been simplified, as it lets the network give high scores to multiple overlapping prior boxes, even though they are not the best for any ground truth object. Once, we have the classified each prior box, we encode the ground truth objects with respect to the prior boxes in the $(g_{cx}, g_{cy}, g_w, g_h)$ representation. We then move onto the localization task.

1) Localization Task: In the localization task, the smooth L1 loss function is used for computing the loss. Now the ground truth objects and the predicted bounding boxes are both in the same representation with respect to the same prior boxes. Therefore, the loss function is applied to the encoded predicted bounding boxes and encoded the ground truth objects, which are labeled with a non-background/positive class.

$$L_{loc} = \frac{1}{n_{pos}}(\sum_{pos} L1_{smooth}(pred_{loc}, ground_{truth}))$$

2) Classificication Task: The classification task consists of giving a confidence score to each predicted bounding box for each class. Each predicted box has a class associated given by the model, with a certain score for each class. Therefore, we apply the *cross entropy*[10] function on all predicted classes for each encoded predicted bounding box and the already associated classes from the encoded ground truth objects.

Hard Negative Mining: After the matching, which was done above most of the boxes will actually be associated with the class *background*. This represents an extreme imbalance between *non-background* and *background* predictions. Therefore instead of using all the *background* predictions as loss, we sort them using the highest confidence loss for predicted bounding box and take out the top negative ones such that the ratio between background and non-background is at most 3:1. Then the confidence loss is evaluated in the following way:

$$L_{conf} = \frac{1}{n_{pos}}(\sum_{pos} CE_{loss} + \sum_{hneg} CE_{loss})$$

By combing the localization and confidence loss we define our objective loss function which is defined as:

$$L_{MultiBox} = L_{conf} + \alpha L_{loc}$$

Where $\alpha = 1$, as suggested in the original paper/implementation[11], but can also be subject to changes.

F. Detection

Once the model has been trained, the detection part comes into place. By applying the model on images the predictions will be in the format of $(g_{cx}, g_{cy}, g_w, g_h)$, see subsection II-D. This format is not easily interpretable, as the predicted bounding boxes are represented as an offset of their respective priors. Therefore, the following strategy is followed into detecting actual objects[12] from the predicted boxes:

1) For each image, 8732 predictions have been made, both for localization of the bounding box and the confidence score, for belonging to a certain class.
2) The localization predictions are decoded to $(x_{min}, y_{min}, x_{max}, y_{max})$ format.
3) For each non-background class, we take out the confidence score for each predicted bounding box.
4) Bounding boxes that do not satisfy the certain threshold level given by the user are removed.
5) **Non-Maximum Suppression:** For the remaining bounding boxes we apply non-maximum suppression(NMS). The bounding boxes that satisfy the score threshold level are sorted out. Then the Jaccard Overlap function is performed, such that we have an overlap score for how much each predicted bounding box overlaps with all of the other predicted bounding boxes. Based on an overlap threshold given by the user, predicted bounding boxes that overlap more than the overlap threshold are eliminated and only the one with highest confidence score is kept for that class. By applying

NMS, redundant predictions are being eliminated.

III. Implementation & Training

A. Input Data

The data with which our SSD300 was trained consists of 418 images. The images mainly contain buildings with windows and doors.

B. Generation of Ground Truth

In this subsection, we describe the data needed to train an SSD. Our model should be able to detect objects of different kinds and draw rectangles around those objects with their respective size and class(window, door or building). In our case, we create the SSD300, where the 300 implies that the images to train/test the model must have $(300, 300)$ dimensions.
The data used to train the SSD300 must have the following 3 inputs:

1) Original Image resized to dimensions $(300, 300)$.
2) Objects detected in the image. The objects will be referenced as bounding boxes and should be in the format of $(x_{min}, y_{min}, x_{max}, y_{max})$, where x describes coordinates on horizontal axis and y on vertical axis.
3) Objects' labels. Each object in the image must be classified as either (window, door or building).

To generate the specific ground truth for each image as defined above, we utilized a ground truth created for pixel-wise predictions from a last year student, and we extracted the needed data in the way defined above see Figure 3.

Figure 3: Original Image, Pixel-Wise Ground Truth from Last Year, Ground Truth for SSD

C. Generation of Prior Boxes

As mentioned in section II, prior boxes are pre-computed and generated in a certain way. From the SSD300, we choose 6 particular intermediate layers which are used as feature maps:

- Layer 3 from convolution block 4 and convolution layer 7, both of them belonging to the base convolutions.
- The second layer of convolution blocks 8, 9, 10, 11, all of them belonging to the auxiliary Convolutions.

For each of these feature maps a certain number of prior boxes is generated with corresponding aspect ratios and scales that should match the size and shape of ground truth objects. Therefore, for choosing the proper aspect ratios and scales for prior boxes, we inspected the training data and we got histograms Figure 4 and Figure 5. For more details on the exact aspect ratios, scales chosen

and the number of prior boxes being generated for each feature map level, see Table I and Table II.

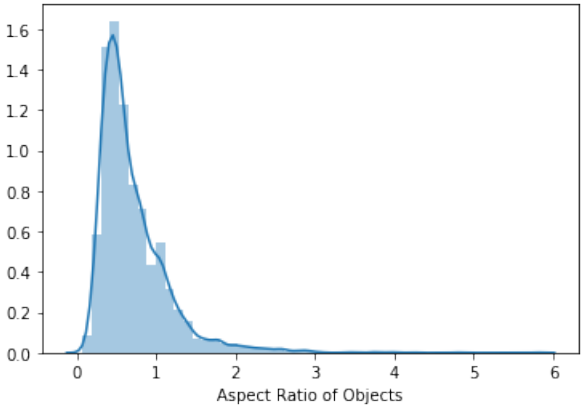

Figure 4: Aspect Ratios(w/h) of Training Data

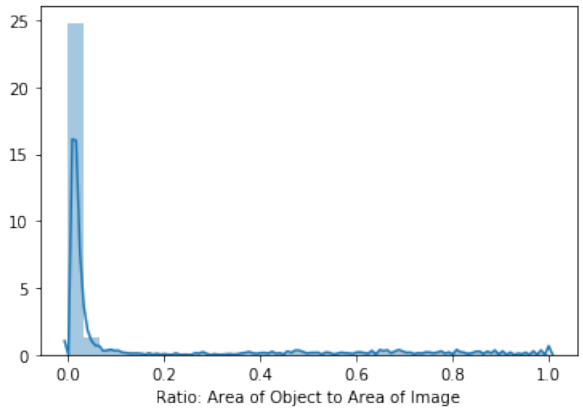

Figure 5: Ratio of area of objects to area of image - Training Data

D. Data Augmentation

To make the model more robust to different object sizes and shapes, we apply the following augmentations on images when we train the model:

1) Photometric Distortions: We apply a series of photometric distortions to the original image Figure 6, each with 50% chance of success. The distortions applied are adjusting brightness, contrast and saturation with an adjust factor in between 0.5 and 1.5, and hue with an adjust factor in between $-\frac{18}{255}$ and $\frac{18}{255}$, see Figure 7.

Feature Map	Size	Ratio	Scale
Conv4_3	(38, 38)	1, 1, .4, .333	0.02
Conv4_7	(19, 19)	1, 1, .6, .5, .4, 0.333	0.03
Conv8_2	(10, 10)	1, 1, .6, .5, .4, .333	0.03
Conv9_2	(5, 5)	1, 1, .6, .5, .4, .333	0.03
Conv10_2	(3, 3)	1, 1, .6, .5, .4, .333	0.03
Conv11_2	(1, 1)	1, 1, .6, .5, .4, .333	0.03

Table I: Generation of Prior Boxes

Figure 6: Original Image

Figure 7: Photometric Distortions

Feature Map	Prior Boxes Per Feature Map Formula: Size^2 $*$ Boxes Per Position
Conv4_3	$38 * 38 * 4 = 5,776$
Conv4_7	$19 * 19 * 6 = 2,166$
Conv8_2	$10 * 10 * 6 = 600$
Conv9_2	$5 * 5 * 6 = 150$
Conv10_2	$3 * 3 * 4 = 36$
Conv11_2	$1 * 1 * 4 = 4$
	Total: 8732

Table II: Prior Boxes Per Feature Map

2) Zoom Out - Expansion Operation: A zoom out operation is performed on the original image with a 50% chance of success. This operation is helpful in the detection of smaller objects. The zoom out factor must be in between 1 to 4, see Figure 8. The surrounding space is filled with the mean of the ImageNet data, since we are using the pre-trained **VGG-16** network on *ImageNet* data.

Figure 8: Zoom Out Operation

3) Random Crop: Random crop, helpful in detecting larger or partial objects, see Figure 9. There is chance that the image stays the same. The crop dimensions are in between 0.3 and 1. The aspect ratio is in between 0.5 and 2. The patch of the image are sampled such that minimum jaccard overlap with the objects is 0.1, 0.3, 0.5, 0.7 or 0.9. All objects which had their centers cropped from the image are discarded.

Figure 9: Random Crop

4) Horizontal Flip: The image is flipped horizontally with a 50% chance of success, see Figure 10.

Figure 10: Horizontal Flip

5) Resize Image: After all of the augmentations are performed the image is resized to dimensions $(300, 300)$ and is ready to be used as input by the model.

IV. Training, Validation & Results

For training and validation of the model, the following ratios and parameters were used:

- For training the model $\frac{7}{10}$ of the 418 images were used, while the other $\frac{3}{10}$ were used for validation.
- The $batch\_size = 8$ for both training and validation.
- Data Augmentations are performed only on training data.
- The $weight\_decay\_rate = 0.0005$ and the $momentum = 0.9$, used for the optimizer.
- The $lr = 0.001$ in the optimizer, however for bias parameters the $learning\_rate = 2 * lr$, following the original implementation[11].
- The best model was trained for *358 epochs,* and has a validation *MultiBox Loss = 2.559*.

As you can see in Figure 11, the average loss per epoch for both training and validation is getting smaller as time passes. However, due

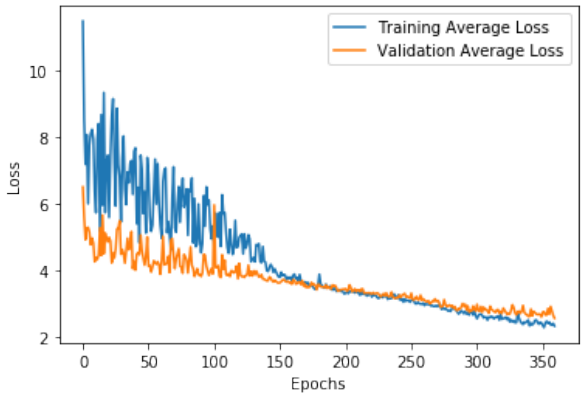

Figure 11: Training and Validation Average MultiBox Loss per Epoch

to the diversity of each image, and the variance of difficulties of detecting objects in different images, the average loss per epoch tends to deviate quite a lot at the beginning.

In Figure 12, we have taken one image value loss per epoch and as you can see at the beginning the training values show an even greater deviation. However, one unusual thing that we can notice is that the deviation is by far greater for the training than for the validation. Therefore we trained

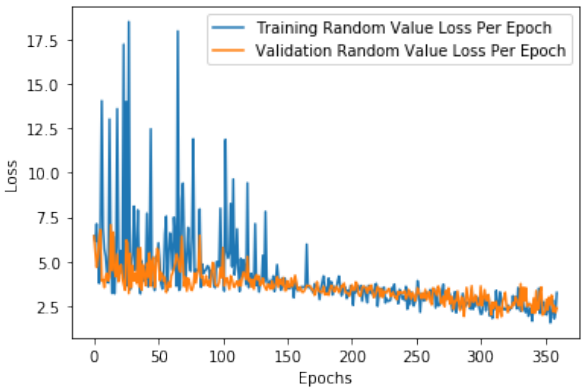

Figure 12: Training and Validation loss of a random image of batch per Epoch

a model without data augmentation and we saw that the deviation is due to the data augmentation. But another important result that we discovered is the fact the *best avg validation MultiBox Loss = 3.465*, while the training avg loss continued to decrease, see Figure 13. This validation loss is far worse than the model which had data augmentations, that is why we decided to keep the data augmentations in our best model, even though there are a lot of deviations at the beginning. The objects {*building, window, door*} are

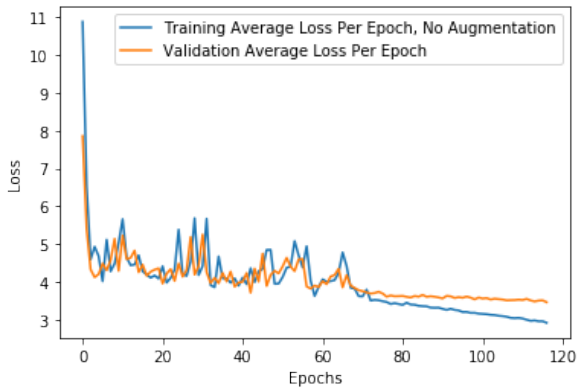

Figure 13: Model Trained Without Data Augmentation

associated with the colors {*yellow, green, red*} respectively, when detected. As you can see in Figure 14 and Figure 15, all objects are being predicted correctly with a

small localization error. The ground truth

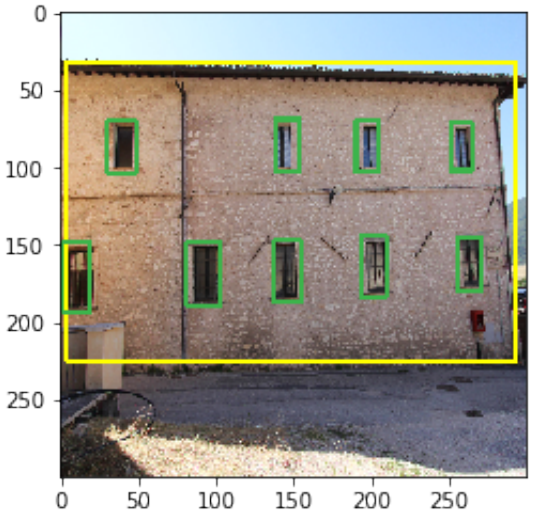

Figure 14: Result 1: Detections predicted by the Model.

that was used to train the model, mostly consisted of 1 or 2 buildings with only their front side being detected per image with its respective windows and doors. The negative side of a ground truth like this is the fact that images having multiple buildings shown, but are labeled as background, would make the MultiBox Loss worse than it is, if the model detects such a *building*, *window* or *door*. For

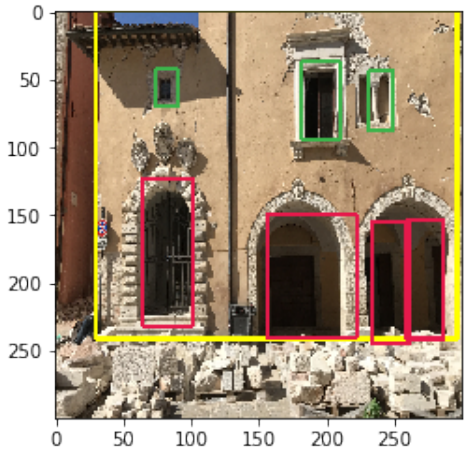

Figure 15: Result 2: Detections predicted by the Model.

Figure 16: Comparison of actual ground truth with prediction of the model.

example in Figure 16, as it can observed the ground truth, with which the model is trained has only one building with its respective windows, while the model accurately detectes the building and its windows, but it also detected other windows of other buildings present in the image. Same result can be observed in Figure 17. Therefore, the model would perform way better if there were more well-defined data available to train on, rather than a set of 418 images. By well-defined we mean, if we are expecting from the model to predict 1 or 2 buildings in an image, then it would make sense to have only 1 or 2 buildings in all images of the training set. This would yield a better accuracy of the model as observed in Figure 14, Figure 15 and Figure 18. Another improvement that could be done on the ground truth is to define whether the ground truth should contain multiple

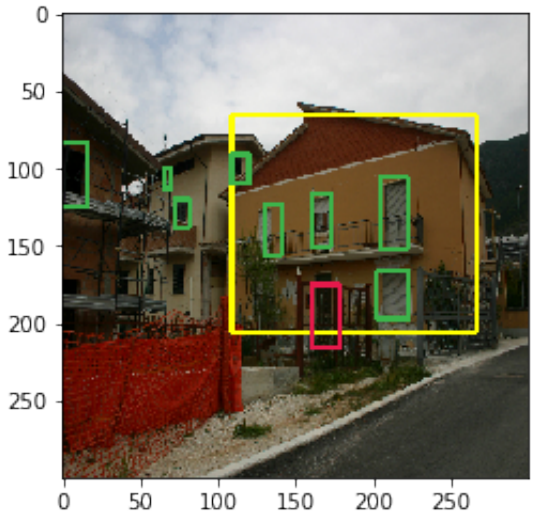

Figure 17: Multiple buildings in an image.

sides or only one side of a building, and that to be the only side shown on the ground truth. Imperfections of the model are the too many detections predicted by the model, in our case 8732. Even though the usage of the Non-Maximum Suppression technique leads to elimination of redundant results, this technique is not fully automated, as it requires tuning parameters {*min_threshold_score, max_overlap_threshold, top_objects_to_show*} for each image differently. Thus, by setting a higher score of detected objects will lead into detecting only the best results, which does not imply that all of the ground truth objects will be detected, as some objects may be more difficult for detection than others. However, regarding the max jaccard overlap threshold parameter, we have concluded that as we are detecting {*buildings, windows and doors*}, non of them should have an jaccard overlap higher than 0 per class, since we are taking for granted that windows do not overlap with other windows and same interpretation following for doors and buildings. Therefore, the only parameters to be tuned are the min score threshold parameter and how many of the top detected objects should be taken into account,

Figure 18: Examples with one building.

while satisfying the min threshold score. An appropriate example of this is Figure 19, which depicts what exactly happens with the detections as the min score parameter gets smaller.

For additional examples, see Figure 20 and Figure 21.

Figure 19: Image 1 Min Score=0.26, Image 2 Min Score = 0.1, Image 3 Min Score = 0.01, while top detected objects shown in all images is 50.

Figure 20: Examples of Detections

Figure 21: Example of Detections

V. Library

The library is named `ssd_project`, it has been well documented and pretty straightforward to understand, taking into account that all of the code is heavily commented. The SSD300 network has been implemented in **PyTorch**. In the subsections to follow, we explain the modules that the project is comprised of.

A. *`ssd_project.model.*`*

This module contains the construction of the SSD300 and the generation of prior boxes function, which is part of the SSD300 class.

B. *`ssd_project.functions.*`*

This module contains the `MultiBox Loss` class and `Detections` functions. There are a couple of functions used for prediction, detection and drawing in the `detections.py` file. `MultiBox Loss` class is used as a criterion for training and evaluating the model.

C. *`ssd_project.utils.*`*

This module contains the data augmentations applied on the data, all the encodings and decodings for all the representations, and a lot of functions used as helpers.

VI. Conclusion

As a whole, the SSD300 architecture proved to be a great tool in addressing the object detection problem, and it definitely can be used for the facade parsing problem. As mentioned before, more data would come in handy with a better defined ground truth than this one. Altogether, the best trained model managed obtain **mean average precision**(mAP)[13] of `64.4%` in object detection for the classes {*buildings, windows, doors*}, with individual mAPs of:

1) Buildings `mAP = 80.99%`
2) Windows `mAP = 58.65%`
3) Doors `mAP = 53.75%`

For generating the `mAP` the following parameters were used:

1) `min_score_threshold = 0.01`
2) `max_overlap_threshold = 0.6`
3) `top_k_objects = 200`

The results are satisfactory, considering the fact that **mAP** of the model was tested against the ground truth, which has its imperfections too. The results could obviously be improved, especially for the doors and windows, if the improvements brought up in section IV are implemented. Ultimately, the whole code structure is straightforward and easy to understand, making it easily upgradeable or mergeable with another tool.

VII. Future Improvements

A couple of improvements that could be done to this project are:

1) A better ground truth could be devised, which should be well defined.
2) For tuning the parameters a *U-Net*[14] could be used. The U-Net could be used for detecting the exact number of objects that there are in the picture, and then we could use the SSD300 to actually predict only these objects by fixing the `top_k` parameter.

VIII. Repository & Acknowledgements

The whole documentation about project and the code can be found at https://github.com/ilijagjorgjiev/SSD_FacadeParsing. I would like to thank the Swiss Data Science Center at EPFL, and especially my advisor *Radhakrishna Achanta* for all the help and technical support provided. Implementations followed and being helpful for the actual implementation of this project are [11], [15], [16].

REFERENCES

[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2014.

[4] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," 2015.

[5] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2013.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Lecture Notes in Computer Science*, p. 346–361, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10578-9_23

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[8] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," 2013.

[9] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe, "Scalable, high-quality object detection," 2014.

[10] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," 2018.

[11] D. E. C. S. S. R. C.-Y. F. A. C. B. Wei Liu, Dragomir Anguelov. (2016) Understand single shot multibox detector (ssd) and implement it in pytorch. [Online]. Available: https://github.com/weiliu89/caffe/tree/ssd

[12] D. Hoiem, Y. Chodpathumwan, and Q. Dai, "Diagnosing error in object detectors," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 340–353.

[13] J. Hui. (2018) Mean average precision for object detection. [Online]. Available: https://bit.ly/34FY4Cf

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[15] S. Vinodababu. (2018) Ssd: Single shot multibox detector — a pytorch tutorial to object detection. [Online]. Available: https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/

[16] M. deGroot. (2018) A pytorch implementation of single shot multibox detector. [Online]. Available: https://github.com/amdegroot/ssd.pytorch