Article review and implementation: Image-to-Image Translation with Conditional Adversarial Networks

Mariem MEZGHANNI et Cyrine CHTOUROU

January 31, 2018

1 Introduction

The automatic image-to-image translation problem is the task of translating one possible representation of the scene into another, given sufficient training data. These tasks are challenging since a single input image may correspond to multiple possible outputs. The convolutional neural nets CNN are commonly used for image prediction problems. Despite being an automatic process, CNN algorithm requires the user to specify the objective function to minimize, for instance one may choose to optimize the Euclidean distance between predicted and ground truth pixels. It would be therefore desirable to come up with an approach which only requires to specify what we want to obtain (e.g. realistic image or sharp output) and to automatize the learning of the relevant loss function.

Generative Adversarial Networks GANs were proposed to address this issue. GAN algorithms aim at making the output indistinguishable from reality by automatically learning a loss function appropriate for this goal. This work proposes a further developed GAN approach, the so called conditional Generative Adversarial Networks cGANs which are similar to the GAN with additional conditions on the input image.

2 Problem statement

In this project, we want to learn a building/non-building label for each pixel in an aerial image. To reach this goal, we aim at using adversarial networks with a surrogate of the Jaccard loss function. This problem where the output is less complex than the input, is a semantic segmentation problem. According to [2], simply using L1 regression gets better scores than using cGANs with a cross-mutual entropy objective. This project will allow us to either confirm that L1 is still better than cGANs for this kind of problems or to demonstrate that using Jaccard loss as the cGAN objective outperforms previously used methods.
The outline of this paper is as follows. In the next section we investigate the GAN and cGAN approaches. Then, we propose a new implementation of the GAN method by introducing a Jaccard loss function [1] to substitute the objective function of the GAN method. In the last section we will present our experimental results and conclude on the efficiency of this method compared to the state of the art.

3 Presentation of GAN and cGAN methods

GANs are generative models that learn a mapping from random noise vector z to output image y: $G : z \mapsto y$ while cGANs include the observed image x in the learning process: $G : \{x, z\} \mapsto y$.

The learning procedure is illustrated in figure1. Two networks need to be learned: the generator G and the discriminator D. The discriminator learns from the training data to distinguish between a real image and

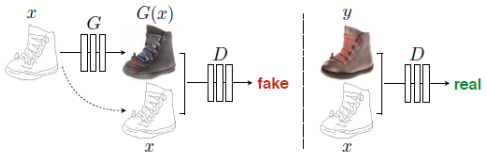


Figure 1 – Training a conditional GAN to map edges→photo.
D: the discriminator and G: the generator.

a fake one. The generator learns to fool the discriminator by producing the fake images to train this latter. The objective used to train the networks D and G of GAN is

$$\mathcal{L}_{GAN}(G,D) = E_y[logD(y)] + E_{x,z}[log(1 - D(G(x,z)))]. \tag{1}$$

One can understand that D tries to maximize this objective. In fact, the goal is to improve the discriminating ability of the network D by maximizing the evaluation of the real image and minimizing the evaluation attributed to the fake one given respectively by the first term and second term in equation (1). The G tries in opposite to minimize $\mathcal{L}_{GAN}$ by producing fake images that are more likely to fool the discriminator.

In the conditional setting cGAN, the discriminator observes in addition the input image x. Hence the objective of cGAN is as follows

$$\mathcal{L}_{cGAN}(G,D) = E_{x,y}[logD(x,y)] + E_{x,z}[log(1 - D(x,G(x,z)))]. \tag{2}$$

To optimize our networks, the G and D are obtained by iteratively computing one gradient descent step on D then one step on G.
The generator G learning can be improved by mixing the cGAN objective with a more traditional loss function such as L_1 distance.

$$\mathcal{L}_{L_1}(G) = E_{x,y,z}[\|y - G(x,z)\|_1]. \tag{3}$$

This would lead the generator to learn to fool the discriminator while keeping it near the ground truth in an L_1 sense. The final objective becomes

$$G^* = arg \;\; min_G max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L_1}(G) \tag{4}$$

In the present work, the cross entropy loss function is used to construct the objective. Our goal is to try other losses and to examine how they perform on the image segmentation problem image→label. Let's describe in the following section the two losses that we used in the present study.

4 Loss functions

In this section, we introduce the losses that we use in the experiments. The main attention here is paid to the Jaccard loss which represents the object of the present study.

4.1 Jaccard loss and the Lovàsz Hinge extension

In binary classification, one of the most common performance measures is the Jaccard index which is defined as follows:

$$J(y^*, \tilde{y}) = \frac{|y^* \cap \tilde{y}|}{|y^* \cup \tilde{y}|}. \tag{5}$$

where:

- y^* is the ground truth labeling.
- $\tilde{y}$ is the labelling to be evaluated.
- J(0,0)=0.

The corresponding loss function is then defined as $\Delta_J(y^*, \tilde{y}) = 1 - J(y^*, \tilde{y})$.
Although this loss function is very intuitive, it has unwanted properties, properties that do not make the task of optimizing it easy. Which is why, we will be using a piecewise linear convex surrogate to it based on the Lovàsz extension of a set function. Lovàsz hinge loss allows the extension of a set-function defined on the vertices of the hypercube $\{0,1\}^p$ to the full hypercube $[0,1]^p$. Minimising the Lovàsz hinge loss is so far the best way to optimise the Jaccard index in a continuous optimisation framework. Jaccard loss is a submodular and monotonic set function [1] which entitles it to the Lovàsz hinge extension:

$$\tilde{F}(s \in \mathbb{R}^p) = \sum_{j=1}^{p} max(s^{\pi_j}, 0) * (F(\{\pi_1, ..., \pi_j\}) - F(\{\pi_1, ..., \pi_{j-1}\})) \tag{6}$$

where:

- F is any increasing submodular set function, here it is the Jaccard loss.
- $\tilde{F}$ is F's Lovàsz hinge extension.
- π is the permutation ordering the margins $s \in \mathbb{R}^p$ in decreasing order.

This extension is a convex, piecewise-linear surrogate to Jaccard loss but it is still difficult to optimise since it is not differentiable and hence calls for using a different method than standard gradient descent approaches. A good candidate for minimising this loss function is the proximal gradient algorithm where the proximal gradient operator of the Lovàsz hinge loss is defined as:

$$prox_{\tilde{F},\lambda}(x) = \underset{u}{\operatorname{argmin}}(\tilde{F}(u) + \frac{\lambda}{2}||u - x||^2) \tag{7}$$

Algorithm 1 Computation of $\text{prox}_{\tilde{F},\lambda}(\boldsymbol{\gamma})$

Input: Current margins $\boldsymbol{\gamma}$, F, λ
Output: $\boldsymbol{\gamma}^* = \text{prox}_{\tilde{F},\lambda}(\boldsymbol{\gamma})$

$\mathbf{v}^0$, $\pi \leftarrow$ decreasing ordering of $\boldsymbol{\gamma}$ and permutation
$\mathbf{v} \leftarrow \mathbf{v}^0$
$\mathbf{g} \leftarrow \mathbf{grad}\ \tilde{F}(v)$ (as a function of the sorted margins)
$\mathbf{E} \leftarrow \{$constraint $g_i = g_{i+1} = \ldots = g_{i+p}$ for each equality $v_i = v_{i+1} = \ldots = v_{i+p}\}$
$\mathbf{c}_z \leftarrow$ constraint $g_{z+1} = \ldots = g_d$ for z minimal index such that $\mathring{\gamma}_z < 0$
finished $\leftarrow$ False
while not finished **do**
  if $\mathbf{g} = \mathbf{0}$: **break**
  $\mathbf{g} \leftarrow \mathbf{proj}_{\mathbf{E} \cup \{\mathbf{c}_z\}}\ \mathbf{g}$
  $\mathbf{v}_{\text{next}} \leftarrow$ projection of $\mathbf{v}$ on the closest edge of $\tilde{F}$ in the direction $\mathbf{g}$
  stop $\leftarrow 1/\lambda + \langle \mathbf{v} - \mathbf{v}^0, \mathbf{g} \rangle / \langle \mathbf{g}, \mathbf{g} \rangle$
  if stop $< \|\mathbf{v}_{\text{next}} - \mathbf{v}\|$ **then**
    $\mathbf{v} \leftarrow \mathbf{v} + \text{stop} \cdot \mathbf{g}$
    finished $\leftarrow$ True
  else
    $\mathbf{v} \leftarrow \mathbf{v}_{\text{next}}$
    Add corresponding new constraint to $\mathbf{E}$ or update $\mathbf{c}_z$
  end if
end while
return $\boldsymbol{\gamma}^* = \mathbf{v}[\pi^{-1}]$

Figure 2 – Proximal Gradient Algorithm: optimising the Lovàsz hinge loss.

Let's describe how the proximal gradient algorithm works. For each image in the training set, we compute the Lovasz Hinge loss. The total loss is the average of the obtained losses.

The optimization of the loss function in question can performed via the Proximal Gradient Algorithm which, at each iteration, takes a margins vector γ and returns the optimal γ^* iteratively until the convergence.

Let's now focus on the computation of γ^* as illustrated in algorithm 1. In fact, this is the key step of the present approach. The goal of the algorithm is, given the margins γ, the loss F and the regularization parameter λ, to estimate γ^* by an iterative update of γ in the feasible region along the opposite of $\tilde{F}$'s gradient g until the estimated optimal step is valid. More precisely, we update the sorted margin vector v along the gradient g projected on $E \cup \{c_z\}$. In other words, when we project on the $\{c_z\}$, we set $g_i = 0$ for all i such that $\gamma_i < 0$. This means that terms in γ_i of $\tilde{F}$ are already minimal (=0) and no amelioration is needed for this term. For the projection on the E set, for all components of v verifying $v_i = v_{i+1} = ... = v_{i+p}$, we affect to the corresponding components in g the average of the corresponding $g_i = g_{i+1} = ... = g_{i+p}$ so they are updated in the same way. This would save us from computing similar margins separately. Once obtained, we compute the projection of the sorted margins on the closest edge of F in the direction g. In fact, this would give us the feasible region of the updates of the margins. When computing the optimal step *stop*, we start by verifying that this step is valid. If it is so, we update the sorted margins accordingly and return the obtained optimal margins. Otherwise, we set our sorted margins to its projection and we add the corresponding most violated constraint until an optimal feasible step is obtained.

After iterating this process on the γ values using an annealing schema of $(\lambda_t)_t$, an approximation of the optimal margin is obtained to estimate the Lovasz Hinge loss function used to train our model.

4.2 Hinge loss

When we are processing a binary classification problem i.e. $y \in \{\pm 1\}$, the hinge loss is an upper bound of the number of mistakes made by the classifier. It is estimated as follows

$$H(y^*, \tilde{y}) = max(0, 1 - y^*\tilde{y})$$

where $\tilde{y}$ is the output of the classifier's decision function (not classification label). Hence, when y^* and $\tilde{y}$ have the same sign, or equivalently when $\tilde{y}$ predicts the correct class, $H(y^*, \tilde{y}) = 0$. Otherwise H increases linearly with $\tilde{y}$.

The hinge loss function is both convex and continuous, but it is not smooth. Hence methods relying on the differentiability such as the gradient descent method method are not optimal in this case. However, the hinge loss has a subgradient which permits as to use subgradient descent methods to optimize the labeling.

4.3 Binary cross entropy loss

The cross entropy loss (or log loss) measures the performance of a classification model when its output is a probability between 0 and 1. Its expression is the following

$$BCE(\tilde{y}, y^*) = -\tilde{y}log(y^*) - (1 - \tilde{y})log(1 - y^*)$$

The intuition behind using BCE loss is that it penalizes wrong predictions especially those that give wrong labels with $\tilde{y}$ close to 1 since the $-log$ function increases rapidly close to zero. In other words, BCE penalizes mis-predictions especially those that are wrong and confident.

5 Network architectures

The network architecture used for both the generator and discriminator is of the form convolution-BatchNorm-ReLU.

5.1 Generator architecture features

In image-to-image translation problems in general, the ouput structure is supposed to be aligned with the input structure to some extent. Hence, it would be useful to be able to pass through this information in the network. However, encoder-decoder architectures do not allow to pass that information because they progressively downsample the input until a bottleneck is reached and then upsample it. A remedy to this loss of information is to use skip connections such that for each layer i , all channels are connected to channels at layer $n-i$ where n is the number of layers in the encoder-decoder network. This adaptation of encoder-decoder networks is called a "U-Net".
Convolutions used in the encoder decoder are 4*4 spatial filters with stride (i.e. the distance between each filter in the input image) 2. The encoder of the U-Net is of the form $CL64 - CL128 - CL256 - CL512 - CL512 - CL512 - CL512$ where CLk is a Convolution-BatchNorm-LeakyReLU layer with k filters. BatchNorm is not applied to the first layer for both the generator and discriminator. The Leaky ReLU used is of the form:

$$f(x) = \begin{cases} x & \text{if x>0} \\ 0.2 * x & \text{otherwise} \end{cases}$$

The decoder is of the form $CD512 - CD1024 - CD1024 - C1024 - C1024 - C512 - C256 - C128$ where CDk is a Convolution-BatchNorm-Dropout-ReLU layer with k filters and Ck is a Convolution-BatchNorm-ReLU layer with k filters. The dropout used in the decoder has a rate of 50%. We can see here that the decoder doesn't reverse exactly the encoder because of skip connections.

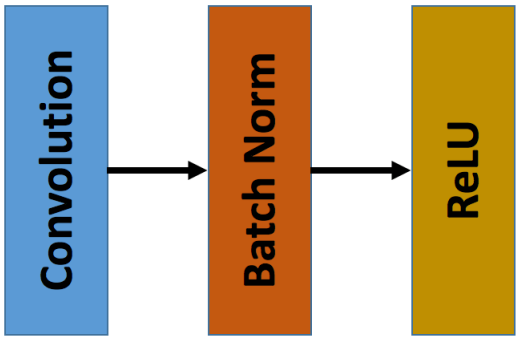


Figure 3 – General form of the network architectures: Convolution-BatchNorm-ReLU

5.2 Discriminator architecure features

The discriminator is also formed of CL-type layers. Another important feature used is that the discriminator works on separate PatchGANs, i.e. only penalizes structure at the scale of an $N * N$ patch at a time and convolutationally runs across the image to cover all patches. The ultimate output of the discriminator is finally the average over all its responses on patches. This means that the discriminator is "Markovian", as it models the image as a Markov Random Field by assuming that each patch is independent from the other. In other words, a pixel in a certain patch is only dependent on pixels from the same patch.

6 Optimization of the networks

For optimization purposes, an Adam solver with learning rate 0.0002 and momentum parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ is used to optimize both the discriminator and the generator networks. The former learns with half the rate of the latter.
The Adam solver used is torch.optim.Adam, it performs optimization step by step using the .step() function. We tried to mimic this solver to create a proximal gradient algorithm to no avail. When we are using the Lovàsz extension of the Jaccard loss, we need to use the proximal gradient algorithm because of the non-differentiability of the objective function. As we weren't able to build a solver that can be generalized to network optimization, we kept using the Adam solver.

7 Experiments

Let's investigate now how the mentioned losses perform on labeling the *AerialImageDataset*. The focus here is on classifying each pixel of the images to the building or non building classes. We use the described *cGAN* approach with binary cross entropy, Hinge and Lovasz Hinge losses.

7.1 Implementation and results

We first downloaded the data from `https://project.inria.fr/aerialimagelabeling` which is composed of separate folders for inputs and ground truth outputs for the train set and only input images for the test set. To be able to apply the cGan described in the article, we used the script given in `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix` called combine_A_and_B.py. This resulted in an image containing the input image on the right side and the output image (i.e. that containing building/non-building segmentation) on the left side. As the test data is not labelled, we used a random split of the training set into 70% destined for training and 30% destined for validation. We finally converted all the images from .tif to .jpg format as it turned out the code doesn't support the former extension.
We trained 3 main types of models on the Aerial dataset, each corresponding to a different loss function. As the article states, generally an L1 loss without using the cGAN framework is largely sufficient for semantic segmentation problems. It also states that cGAN is not only unnecessary in this case, but it is also responsible for making the results worse as it creates artefacts in the image. In order to confirm this hypothesis, we compared the case where $\lambda = 0$ i.e. where we remove L1 regularization with the case where $\lambda = 100$ using the binary cross entropy loss. The results are clearly in favour of using L1 to the detriment of cGan for image segmentation 5. Another parameter we tried to tune was the learning rate of the Adam solver used. We found that a learning rate in the range $0.0002 - 0.0005$ is optimal and that when we use a high learning rate the results are not satisfactory.
We ran the codes on our own computers' CPUs. Each epoch took more than 10 minutes to run, so we limited the number of epochs for the experiments to 100. Our second experiment corresponds to using the hinge loss described in section 4.

We then tried to implement the Lovàsz hinge loss to approximate the Jaccard loss. In order to facilitate the gradient computation, we didn't use the Lovàsz extension alone, we used an annealing scheme to obtain a loss that converges to it as the number of iterations (Forward and Backward steps) grows during a certain epoch.

$$Loss = (\tilde{F}(u) + \frac{\lambda'}{2}||u - x||^2) \tag{8}$$

With $\lambda' = \frac{0.01}{2(log(t)+1)}$ and $\tilde{F}$ is the Lovàsz extension of the Jaccard loss.

Using this loss combined with a $\lambda = 0.5$ for L1 regularization, we obtained the results in figures 8 and 9.

7.2 Conclusions

What stands out is that the $L1$ loss is very efficient in processing this kind of problems. In fact, when referring to the reviewed article [2], one can understand that when it comes to predicting output close to ground truth (labeling in this case), the $L1$ reconstruction is mostly efficient, comparing to graphics tasks for which it produces blurry results and thus need to be combined to the GAN the $cGAN$ approach in order to achieve good predictions.

Now that we have validated this hypothesis, we tried to reduce the effect of L1 regularization to be able to compare other loss functions applied to the Discriminator and Generator. Surprisingly, the hinge loss results using a slight L1 regularization (lambda =0.5 as compared to 100 in the previous experiments) has very good results. As we said in the theoretical description of this loss, it is a non-smooth function so it may be difficult to optimize in pytorch. But luckily, experiments say otherwise. The hinge loss cGAN has by far the best results, it scores an RMSE over our validation set of 0.22 while the BCE loss scores in the range of 0.3-0.33.

$Loss$	BCE	$BCE + L_1$	$Hinge$	$Jaccard - hinge$
$RMSE$	0.33	0.30	0.22	0.23

Figure 4 – RMSE losses measured on our validation set of 5 categories of aerial images after a training with cross-entropy loss, hinge-loss, and Jaccard-hinge.

8 What we learned and challenges that we faced

This project was an ideal opportunity to learn about the GAN approach although it is not the best approach to do semantic segmentation tasks. Besides, we worked on different loss function forms (mainly Jaccard, Lovasz extension, Hinge and cross entropy) and we could investigate their performance on this model. It is also important to highlight that we discovered how non regular submodular loss functions can be extended via the Lovasz extension into convex and piecewise linear loss function. We would also mention the chance we had to get insights into network architectures and optimizer models in the pytorch package.

However, in the implementation step, we faced difficulties to download the aerial images. Besides, the processing of the dataset is comptationally expensive due to its size and it takes up to 10 hours to train a single model for only 50 epochs on our computer.

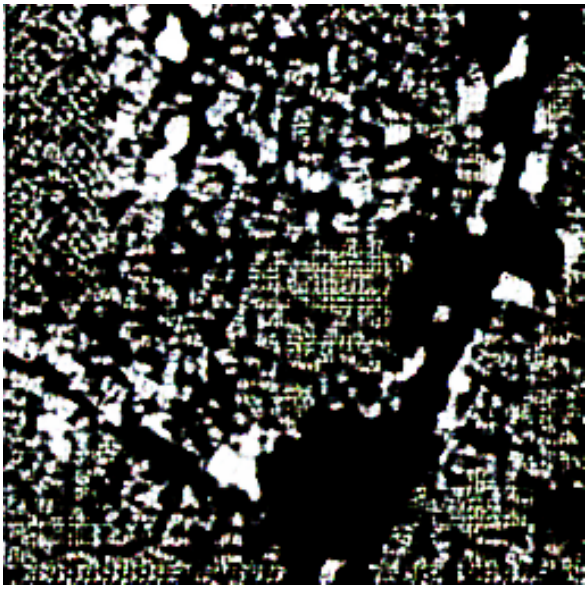
References

[1] M. Berman and M. B. Blaschko. Optimization of the Jaccard index for image segmentation with the Lovász hinge. *ArXiv e-prints*, May 2017.

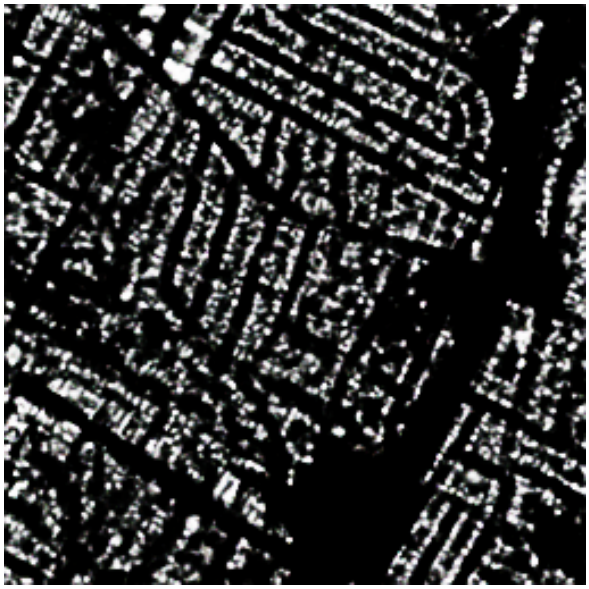
[2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. *ArXiv e-prints*, November 2016.

(a) Input Image

(b) Ground truth



(c) BCE without L_1 regularization



(d) BCE+100*L_1

Figure 5 – Austin12 input image - Binary Cross Entropy loss

(a) Input Image

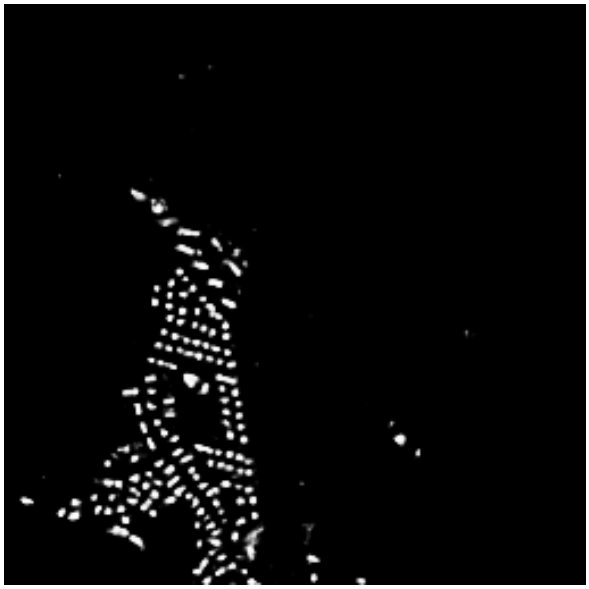
(b) Ground truth

(c) Hinge+0.5*L_1

Figure 6 – Vienna14 input image - Hinge loss

(a) Input Image

(b) Ground truth



(c) Hinge+0.5*L_1

Figure 7 – Tyrol-w27 input image - Hinge loss

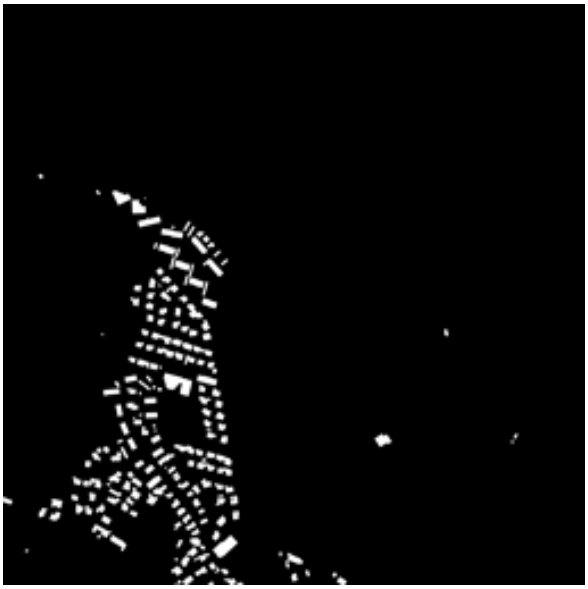
(a) Input Image

(b) Ground truth

(c) Lovasz+0.5*L_1

Figure 8 – Vienna28 input image - Lovasz Hinge loss

(a) Input Image



(b) Ground truth

(c) Lovasz+0.5*L_1

Figure 9 – Tyrol-w27 input image - Lovasz Hinge loss