Université de Montréal

MILA

Ville de Montréal

Semantic Segmentation of the Urban Context Using PTZ Cameras

Saber Benchalel

May 2020

Contents

1 Introduction **1**

2 Context **1**
2.1 Cameras . . . 1
2.2 Proposed task . . . 2
2.3 Fundamentals . . . 4
2.3.1 Semi-supervised Learning . . . 4
2.4 Dataset subdivisions . . . 5
2.4.1 Semantic Segmentation . . . 5
2.4.2 Metrics . . . 6

3 Data **7**
3.1 Carla . . . 7
3.2 Cityscapes . . . 8
3.3 CGMU . . . 9
3.4 Annotation campaign . . . 9

4 Methodology **10**
4.1 Baseline . . . 10
4.2 Adversarial Semi-Supervised Learning . . . 10
4.3 Model architectures . . . 11
4.3.1 SegNet . . . 11
4.3.2 Fast-SCNN . . . 12
4.3.3 Discriminator . . . 14
4.4 Hyperparameters . . . 14
4.5 Computational resources . . . 15

5 Experimental Performance **15**
5.1 Proof of concept . . . 16
5.2 CGMU Results . . . 16
5.3 Next Steps . . . 17

6 Conclusion **19**

A Labels **I**

B QA steps **IV**

C SegNet Parameters **V**

D Fast-SCNN **VI**

E Discriminator Parameters **X**

F Segmented Results **XI**

Acknowledgments

I would like to acknowledge Marie-Odette St-Hilaire, the data scientist in lead of this project at Ville de Montréal. Her help with the annotation campaign and interactions with CGMU greatly helped me complete this project. From Ville de Montréal, I wish to also thank my supervisor Michel Charest. His help to secure resources for this project was essential for the annotation contract and computation external resources.

Finally, I would like to express my great appreciation to Pierre-Luc Carrier from the applied research team at MILA for the weekly technical meetings we had. Many blocking elements were solved throughout the project after our regular discussions.

Lexicon

This section contains the definition of acronyms used in this work.

- **CGMU**: *Centre de Gestion de la Mobilité Urbaine*. CGMU is the depratment in charge of smart transport at Ville de Montréal. They own the cameras used in this project.
- **FPS**: *Frames per second*. The number of images per second produced by the camera or processed by an algorithm.
- **IoU**: *Intersection of Union*. Computer vision performance metric. See section 2.4.2
- **ReLU**: *Rectified Linear Unit*. An efficient activation function used in machine learning. It returns the positive part of an argument.
- **PoC**: *Proof of Concept*. In this project, the proof of concept refers to the work done on Carla and Cityscapes datasets.
- **PTZ**: *Pan-Tilt-Zoom*. The cameras that are used can be remotely controlled to pan left or right, tilt up or down and zoom to enlarge a section of the field of view.

1 Introduction

In the context of my Professional Masters internship, I worked for 7 months at Ville de Montreal from May to December 2019. The task at hand was to propose a machine learning model that could generate semantic segmentation of frames from a live video feed sourced on PTZ cameras. The following document presents the context and problem that needed to be solved in section 2, followed by the datasets and performed annotation campaign in section 3. Section 4 presents the machine learning methods and the two models that were used. The results for each explored path and suggestions to further improve the trained models are presented in section 5. The various problems encountered throughout the project and related solutions are described when adequate throughout the document. Further technical information about the datasets, annotations and model architectures are available in the Appendix.

2 Context

2.1 Cameras

Ville de Montréal has 530 PTZ cameras spread around the city to monitor traffic. Each camera has a set of predetermined angles to monitor each road leading to an intersection. Some presets are grouped in global city-wide sets to automatically point towards recurrent points of interest such as inbound car flow during the morning rush hour or outbound traffic after Montréal Canadiens home games. Operators can also manually control cameras for outlying events such as a broken down vehicle or a power outage rendering a traffic light temporarily out of service.

As the model of newly installed cameras changes through the years, some specs are not shared by all cameras currently in service. For example, cameras that use a wired connection to reach the central network have a higher bandwidth than wireless cameras. Additionally, there is not a single image resolution shared by all the cameras and some models have night vision capabilities.

The distribution of cameras through the city covers a wide range of scenery and backdrops that must be considered. The main camera contexts can be split as:

- Residential: This set contains the cameras at intersections in residential areas of the city. This includes front lawns, cars in driveways and housing facades.
- Commercial: These cameras near store fronts, restaurants and other commercial buildings. They are mostly located Downtown and on main corridors.
- Industrial: This set is for cameras in remote parts of town and have large corporate parking lots near wide arteries in their field of view. They are found in the West Island, Bois-Franc, St-Laurent and Pointe-aux-Trembles.
- Highways: This set includes the cameras near the multiple Autoroutes and bridges around the city. These cameras are mostly pointed towards structures like entry ramps, exits and overpasses.

As part of the open data policy of Ville de Montréal, each camera uploads a snapshot to a public portal[1] roughly every five minutes. Steps are taken to ensure that the privacy of the citizens is respected. The Zoom

[1] Available at http://ville.montreal.qc.ca/circulation/

and Tilt coordinates for each preset have been manually selected as to make facial features and license plates unintelligible. When a camera is being manually controlled to zoom on and monitor a specific element, the periodic snapshots are automatically halted until the camera is returned to a preset. This is seen on the portal as a black place holder image labelled `Image non disponible` (image unavailable).

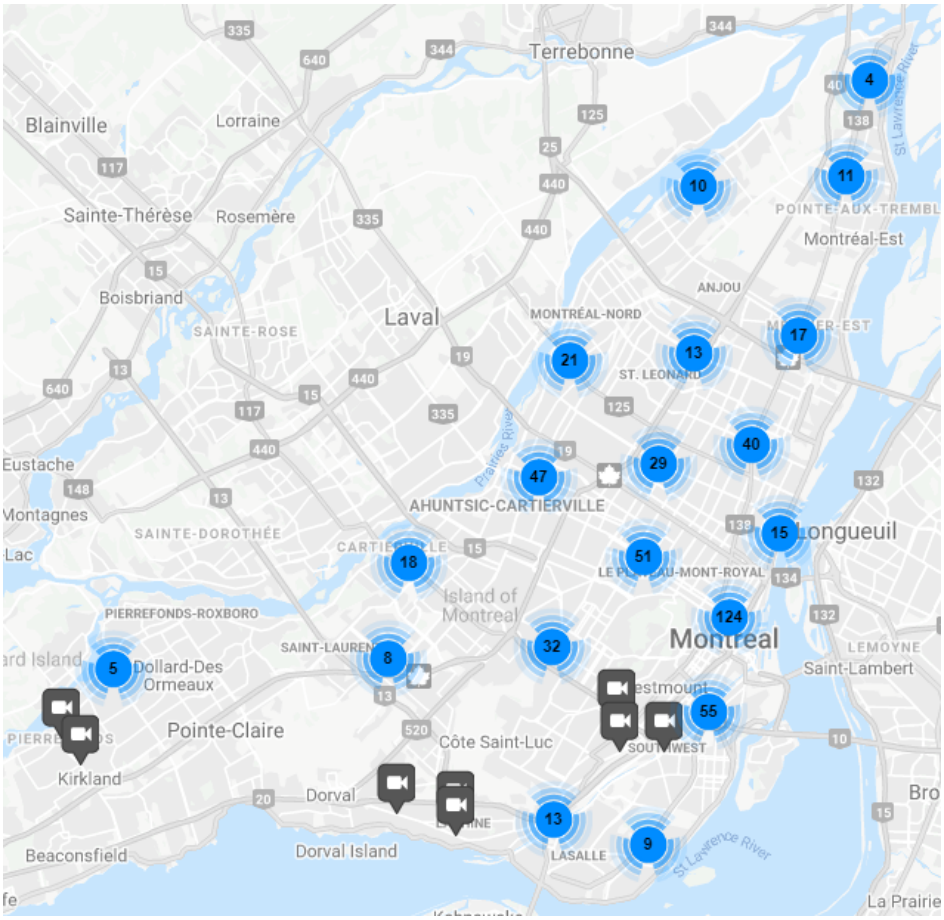


Figure 1: Distribution of the 530 PTZ cameras across the city of Montréal

2.2 Proposed task

As can be expected, it is near impossible for a few operators to simultaneously monitor all 530 cameras in real time. After discussions with the CGMU, it was concluded that a system that could automatically monitor all cameras and flag traffic hindrances would be useful for day to day operations. To be considered usable, such a system would need to meet the following set of minimum requirements. The system should:

- run in real-time or near real-time. With the current set of cameras in place, this would require an algorithm that can process at least 8 frames per second, or a frame every 125 milliseconds at most, while running on reasonable hardware.
- have a low false positive rate. Each incorrect flag of an anomaly would distract and slow down operators, leading to them disabling the system as it is unreliable. While false negatives also need

to be minimized, they are not quite as punishing as false positives. In this case, a false negative is a traffic hindrance that was not automatically flagged to operators, which is on par with the current system.

- be forward-compatible with future cameras. The models should not need retraining any time a new camera is installed.
- adapt to the daily and seasonal variations that affect Montréal. This includes the time of day, the cloud coverage and the seasonal weather.

After further discussion with the transport engineers, it was concluded that there was no official definition of a road hindrance. Operators use tacit knowledge that is learned through experience and difficult to transfer to qualify hindrances. For example, a stopped car could be waiting at a red light or could be broken down and unable to move forward. After making a list of frequent problems the operators monitor, a hindrance was narrowed down to an anomaly in the space *(location, object, movement)*. This definitions makes it possible to distinguish cases like a double-parked vehicle from a car in a regular parking spot, or a car in a reserved bike or bus lane.

In addition to this main project, a few other projects that could be achieved with these cameras have been suggested by other groups. While they haven't been selected as the main project, they are still of interest to Ville de Montréal. This adds another hidden requirement to the main task: the final product should be easily reusable to complete tasks like road marking wear detection, near-miss collisions between vehicle-vehicle and vehicle-pedestrian pairs and road work detection to compare to a work permit database. To satisfy this last criterion, the main project was split into three main parts according to the anomaly triplet.

1. Location: For static telemetry cameras, the standard procedure is to manually draw masks around the regions of interest when the camera is first installed. Since, PTZ cameras are rotated on a regular basis and with the camera meta-data being unavailable, static masks can not be used. The main objective of this block is to classify the image background to identify what the camera is capturing to dynamically generate masks. These masks can then be used to filter the roads, sidewalks, median strips and other areas where objects can be. This module can be reused to detect missing or worn out road marking.
2. Object: This block follows the first step and mixes the camera feed and the location masks for object detection. Cars, bicycles, pedestrians and other static objects in the foreground can be detected on the image. This module can be reused to detect construction sites and broken road signs.
3. Movement: This block uses the output of the previous one on a video feed to detect the trajectory of foreground dynamic objects. This module's main purpose is to detect anomalies, but it can also see usage for near-miss detection. As the current system is reactive and relies on reported accident to find problematic intersections, monitoring near-misses would generate insightful data to predict these locations in a proactive manner. As of December 2019, live video feed was not available outside of the CGMU control room. This block is contingent upon having access to pre-recorded or live video feeds.

With all the blocks ready, a final module would take the feed of a camera and use the three outputs to cluster and detect the two main types of anomalies: an object that isn't where it should be and an object

that doesn't move like it should. This internship focuses only on the first block. The expected delivery is a model to perform semantic segmentation while respecting the global requirements.

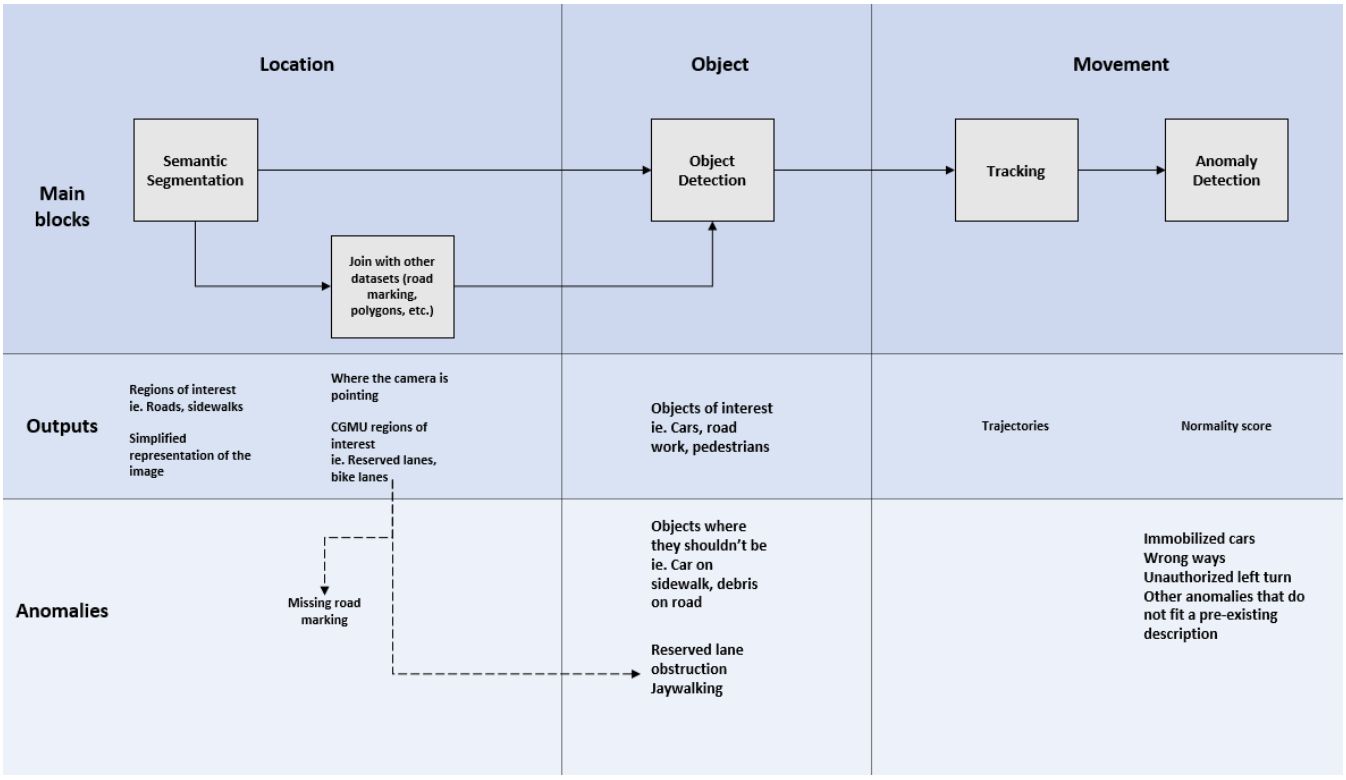


Figure 2: Road map of the anomaly detection project. This internship covers the semantic segmentation task.

2.3 Fundamentals

This section explains a few machine learning principles needed to understand the reasoning that led to the methods that were selected for this first task.

2.3.1 Semi-supervised Learning

Supervised learning is the name given to the ensemble of machine learning methods that use data paired with labels. During training, these methods use the available data to learn functions that output the label associated to the input sample. If the learned functions are adequately trained, they can be used for inference on new data for which labels are not available. For the current project, a large amount of unlabeled data is being generated daily. As using this additional data would allow for more complex models that are less likely to overfit, it is relevant to explore machine learning methods that do not require labels to perform a task. Such learning methods are called unsupervised learning. These tasks will often use the similarity between samples instead of the labels to make predictions. The most well-known unsupervised learning method is clustering, which is used to group similar samples together. Another emergent ensemble of methods that is gaining in popularity in the recent years is generative models. These models try to learn a latent space associated with the representation of the data. This latent space can

then be used to transform the input or extract key features from samples without the need for a label.

There exists a third ensemble of hybrid methods called semi-supervised learning. These methods employ a subset of labelled data along with a larger amount of unlabelled data to perform machine learning tasks. These methods will usually start by using a labelled dataset to learn a representation of the data. As the model starts to get an accurate representation of the data, the unlabelled dataset is used to enhance it. For this first block, a semi-supervised learning algorithm would be optimal, as it would be able to use new continuously generated data without the need for a new annotation campaign.

2.4 Dataset subdivisions

All these methods usually require the new samples to follow a distribution similar to the data used during training. If the process used to generate new samples is different from the one used to generate the training data, the model output becomes unpredictable and unreliable[18]. Furthermore, if the model is too complex for the available dataset, it will overfit the given samples and be unusable for inference on new data. As data is quite often expensive or impossible to acquire, there is a trade-off between the amount of data used and the cost of obtaining usable new data. Some techniques can be used to make sure that the trained models are not overfitting the available data. The most common method seen in machine learning is subdivision of the dataset in three parts: train, validation and test splits. The training split generally contains around 60% to 70% of the dataset and is used during training to tune the parameters of the used model. The validation split contains 10% to 20% of the data and is used during training to monitor the performance of the model. This data is not used to update the parameters of a model. It is used to confirm that the model is able to generalize to unseen data and is not overfitting the training split. This split is often used to guide the hyperparameter search to numbers that increase performance. The test split contains the final 20% of the data and is only used once after the hyper-parameters are selected and the parameters are tuned. This split is used to avoid the bias of the validation set and to ensure that the inference performance on never seen data matches the performance on the validation split.

2.4.1 Semantic Segmentation

Segmentation tasks are used when the objective is to divide the input into sections sharing similar properties. Some segmentation tasks start by clustering the input data into groups based on a similarity metric. If labels are available, the clusters can then be labelled to identify each element. In the particular case of images being used as an input, segmentation refers to dividing parts of the image based on what they represent. Two of the most seen forms of image segmentation are semantic and instance segmentation. Semantic segmentation is a classification task that applies labels to individual pixels or areas of an image, instead of a single label for the whole image. Instance segmentation pushes this concept one step further by also indexing each individual element. For example, a semantic segmentation algorithm on the following image would give a label to all the cars and another label to all the pedestrians. An instance segmentation algorithm would do the same, but it would also differentiate and give a unique index to each car and each pedestrian.
As the objective of this first task is to generate masks for lanes, sidewalks and other road elements, semantic segmentation is sufficient.

Figure 3: An RGB image from a PTZ camera and the associated ground truth labelled image.

Prediction	**True Label**	
	Positive	Negative
Positive	True Positive	False Positive
Negative	False Negative	True Negative

Table 1: Terms used to define accuracy and IoU

2.4.2 Metrics

Many metrics can be used to evaluate the performance of a model for any given task. For classification tasks, accuracy is one of the most prevalent metrics. It is defined as the ratio of samples that are correctly labelled for a given dataset. This ratio is obtained by comparing the number of true positives and true negatives to the total amount of predictions. For classification tasks with multiple labels, all the labels that are not the true label are considered as negative for each sample. In this case, the accuracy is obtained as given in equation 2.1.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

This metric alone gives a general idea of the performance of a task, but it is insufficient by itself. For example, if a label encompasses 95% of a skewed dataset, an algorithm that always predicts this label would have an accuracy of 95%, but it would be completely unusable. For semantic segmentation, a metric that is often used to supplement accuracy is Intersection-over-Union (IoU). This metric compares the area that is correctly labelled with the true area to generate a ratio. This metric is similar to accuracy, but will punish false negatives as well as false positives as shown in equation 2.2. Figure 4 shows the relation between intersection, union, positives and negatives.

$$\text{IoU} = \frac{\text{Intersection}}{\text{Union}} = \frac{\text{TrueArea} \cap \text{PredictedArea}}{\text{TrueArea} \cup \text{PredictedArea}} = \frac{TP}{TP + FP + FN} \tag{2.2}$$

Figure 4: A demonstration of IoU. The right circle represents a predicted area and the left circle represents the area that should have been predicted. The True Positive is the green area composed of the Intersection of both circles. The joint area composed of the True Positive, the False Positive and the False Negative is the Union. To obtain a high IoU, the green area must be maximized while minimizing the red and blue areas. In this diagram, the external area is the True Negative.

3 Data

At the start of the project, there was no available labelled data to use to try different models and compare results. For a first proof of concept, two open datasets were used to test various models and have a better idea of the possible performances.

3.1 Carla

Carla[4] is an open-source simulator with the main purpose of offering various environments for autonomous driving systems. This simulation comes with six different maps that can be loaded with up to one hundred autonomously driven vehicles. The vehicles come pre-equipped with various censors such as RGB and depth cameras. One of the available sensors outputs the semantic segmentation as seen by the camera. By manually moving and rotation the sensors to replicate the position and angles of real-life PTZ cameras, it was possible to generate a synthetic dataset along with true semantic segmentation labels. The simulator has a feature to manually change the weather and time of day.

To generate the dataset, the environment was recorded for up to two minutes from various points of view across all the available maps. The simulation was then re-run twice to save the RGB image and semantic image at each frame. Table 4 in Appendix A shows the available labels that come with the semantic segmentation camera. It was not possible to generate both the images in a single run, as the simulation would be at different time steps and there would be a small discrepancy between the source and the label. This simulator allowed me to generate data with the correct point of view for no cost other than time. However, the synthetic dataset was not fully representative of the real data. As the background is perfectly static for all the images taken from a given point of view, the model could easily overfit based on a few pixels that were unique to each angle. To mitigate this effect, multiple intersections from all the available

maps and a limited amount of consecutive frames were used for each point of view. Another problem with this synthetic dataset is the low texture quality of the environment. As seen in figure 5, the images lack shadows, textures, uneven edges, cracks, potholes and other imperfections of real life. As such, another dataset was needed to improve the initial proof of concept.

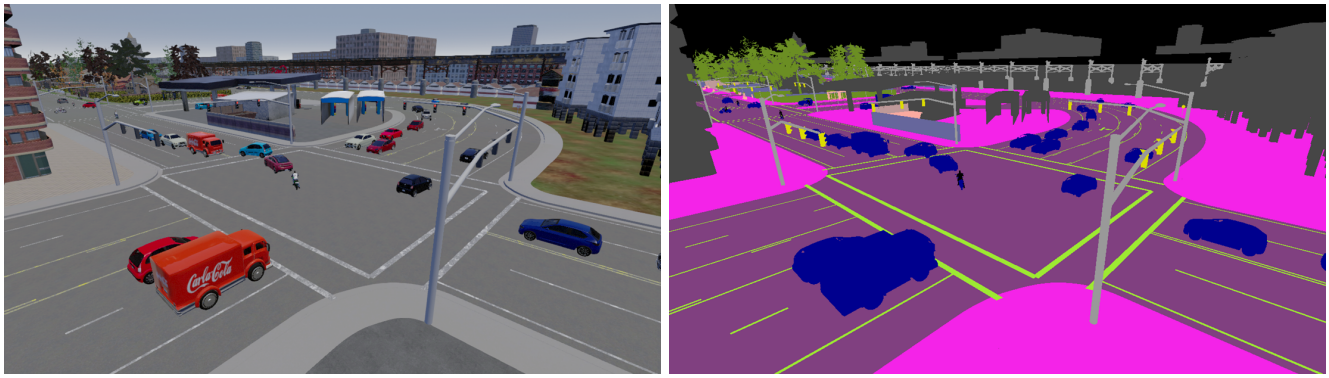


Figure 5: On the left, an image extracted from a RGB camera in the Carla simulator. On the right, an image extracted at the same time step and position using a semantic segmentation sensor instead.

3.2 Cityscapes

The Cityscapes[3] dataset is an open dataset containing dashboard camera photographs taken while driving through 49 German cities and Zurich. The main purpose of this dataset is to train autonomous driving cars, but the available labelled data can be used for a proof of concept. The dataset contains 5000 images with pixel-level semantic annotations. The dataset contains samples with clear and clouded weather spanning from spring to fall. The authors also provide an extra dataset containing 20 000 additional coarsely annotated images. For this work, the annotations are ignored and the samples are used in an unsupervised manner. This dataset has a finite amount of samples and the point of view of the images is different from the CGMU and Carla datasets. The images are 2048 pixels wide by 1024 pixels long, which is 6 to 22 times higher than CGMU cameras. Despite these factors, the images accurately represent the imperfections of reality and a proof of concept that works well with both Cityscapes and Carla datasets would indicate that similar results might be reproducible on CGMU data. Table 5 in Appendix A lists the labels provided with Cityscapes images, their value and their colour.

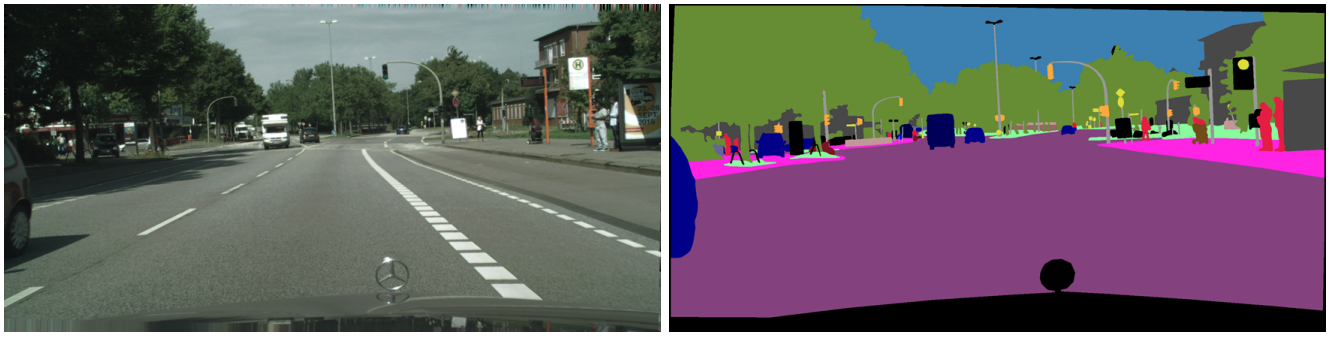


Figure 6: On the left, an RGB image from the Cityscapes dataset. On the right, the annotated version of the image.

3.3 CGMU

To create a diverse dataset that would accurately encompass all the currently available cameras, CGMU experts were tasked with generating a list of ten cameras that they judge to be a fair representation. These cameras were picked based on their model, urban context and resolutions. These cameras were selected for the validation and test datasets. For the training dataset, 64 cameras were selected from the remaining 520. Once these cameras were isolated, 20 random days between May 24$^{\text{th}}$ and July 23$^{\text{rd}}$ were sampled. For each day, 10 random images were randomly sampled from all the samples for that given day. Given the large amount of images sampled this way, this dataset should accurately reflect the weather and lighting of Montréal in the early summer. Around 10% of the sampled images had to be removed, as they were the blocked camera placeholder. A final manual vetting of each individual image was done to ensure that no outliers were kept in the dataset. Such outliers were images completely obstructed by rain drops, sun flares, spiders and trees, or night-time that were too dark for a human to discern any object. The end result is a set of 10 000 images that give an accurate representation of monitored intersections. A similar process was done to generate 17 230 additional images for the extra unlabelled dataset.

As experiments were progressing, it was noticed that the average performance results were different for high and low resolution images. To address this, the dataset was split into two subsets:

> CGMU_S groups the smaller images with a size of 352×288 pixels and 352×240 pixels. This dataset contains 4326 labelled images and 7289 unlabelled images.
>
> CGMU_L groups the larger images with a size of 704×480 pixels. This dataset contains 5672 labelled images and 9932 unlabelled images.

3.4 Annotation campaign

The next step after creating the CGMU dataset was to annotate each image to obtain the ground truth used for supervised semantic segmentation. The final set of labels that were deemed important for this project and future projects is listed in table 6 in Appendix A. To help with quality assurance (QA), a few duplicate images were added throughout the dataset. The annotations for these pairs would be compared to each other to monitor the consistency of annotators.

During the first round of QA, large variations were noticed between images that share the same point of view. These variations were seemingly caused by the bias between annotators, as they had varying thresholds of precision. To demonstrate this effect, heat maps of label variations were generated for batches of images with a similar background. To generate the heat maps, the label distribution for each pixel is generated from a set of annotations. Pixels that have the same label in more than 85% of the images are ignored. The remaining pixels are coloured based on the proportion of pixels that share the same label. Figure 7 shows three of these heat maps. Black pixels are either pixels of another label or pixels that do not have a high variation. Some variation is expected, as background entities are sometimes occluded by cars or pedestrians. However, recurrent bright pixels near the edges of static labels show that there is high variation near their edge.

To fix this problem, the implemented solution is to split the annotation process in three steps. First, a Mother Frame annotation was generated from the static objects in the background of images that share the exact same point of view. Then, an Occlusion image was generated from static objects in the foreground. As

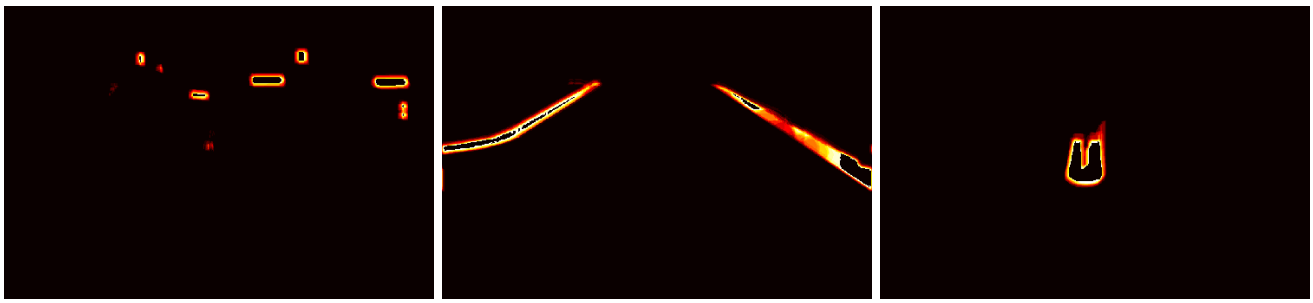


Figure 7: Variation heat maps for three labels in the annotated data. From left to right: traffic sign, sidewalk and median strip. Brighter pixels have the same label in more images.

the images were sampled over multiple weeks, some small variations were noticeable even if the camera was on the same preset. These variations can be attributed to wind, heat and small errors in camera movement when returning to a preset. If a variation of more than two pixels was present in the background, separate Mother Frame and Occlusion layers were generated. Once all the layers were generated, a separate round of QA took place to ensure that the labelling was accurate. The third step was to generate an individual middle layer for each image. This layer contains the dynamic objects that change from frame to frame. The three layers were then combined to generate the final annotated image. A final round of QA was done to catch any remaining errors in the annotations. Figure 15 in appendix B shows the various steps in the annotation and QA process. This process took a total of two months to complete and the dataset was available in October.

4 Methodology

4.1 Baseline

The closest research papers found were either semantic segmentation using dynamic LIDAR cameras[13] or topic signal extraction using traffic cameras[12]. As little to no research was found for semantic segmentation using low resolution PTZ cameras, a baseline needed to be created. To do so, two segmentation models were selected: SegNet and Fast-SCNN. A description of each model architecture is given in section 4.3. Both models were run on all three datasets to generate the baseline accuracy and IoU values, and to assert their strengths and weaknesses.

4.2 Adversarial Semi-Supervised Learning

According to recent work in computer vision [5][7][9], semi-supervised approaches can be used to enhance semantic segmentation tasks on multiple points. In particular, some work[5] focuses on increasing accuracy by using unlabelled data in an adversarial context. To confirm the possibility of harnessing the data generated everyday by CGMU cameras without the need of a costly annotation period, this adversarial method was tested on all datasets and each value was compared to the corresponding baseline.

This adversarial method can use any semantic segmentation model to generate segmentations from RGB images. If the annotated ground truth is available, training is done as normal for the model. In the case where no labels are available, a discriminator model is used to predict which pixels it thinks are generated by a model and which pixels come from the ground truth. The pixels that successfully fool

the discriminator are then used to update the generator. Both models need to be trained with labelled data at first to help convergence. As the discriminator gets progressively better at detecting annotations made by the generator, the latter will get better at fooling the discriminator. This duality, akin to a metaphorical tug-of-war between an investigator and a forger always trying to one-up each other, is illustrated in figure 8. When labelled data is used, multi-class cross-entropy is used to update the generator. To update the discriminator model, the spatial cross-entropy loss function in 4.1 is used. $S(\cdot)$ is the output of a segmentation model and $D(\cdot)$ is the output of a discriminator model. y_n is either 0 if the sample comes from the generator or 1 if the sample comes from a ground truth annotation. h and w refer to the height and width indexes of a pixel. $\mathbf{X}_n$ is a RGB input and $\mathbf{Y}_n$ is a ground truth annotation.

To train the generator, a multi-task loss function (eq 4.2) is used. This loss agglomerates the cross entropy-loss of the model (eq 4.3), the adversarial loss (eq 4.4) and the semi-supervised loss(eq 4.5). λ_{adv} and λ_{semi} are hyperparameters used to control the scale of each loss.

$$L_D = -\sum_{h,w}(1 - y_n)\log(1 - D(S(\mathbf{X}_n))^{(h,w)}) + y_n \log(D(\mathbf{Y}_n)^{(h,w)}) \tag{4.1}$$

$$L_{seg} = L_{CE} + \lambda_{adv} L_{adv} + \lambda_{semi} L_{semi} \tag{4.2}$$

$$L_{CE} = -\sum_{h,w}\sum_{c\in C} \mathbf{Y}_n^{(h,w,c)} \log(S(\mathbf{X}_n)^{(h,w,c)}) \tag{4.3}$$

$$L_{adv} = -\sum_{h,w} \log(D(S(\mathbf{X}_n))^{(h,w)}) \tag{4.4}$$

$$L_{semi} = -\sum_{h,w}\sum_{c\in C} I(D(S(\mathbf{X}_n))^{(h,w)} > T_{semi}) \cdot \hat{\mathbf{Y}}_n^{(h,w,c)} \log(S(\mathbf{X}_n)^{(h,w,c)}) \tag{4.5}$$

In eq 4.5, $I(D(\cdot) > T)$ is either 1 if the output of the discriminator is greater than a given threshold T_{semi} or 0 otherwise. $\hat{\mathbf{Y}}_n$ is the element-wise one-hot encoded prediction of $S(\cdot)$. The threshold is a hyperparameter suggested to be between 0.1 and 0.3. $I(\cdot)\hat{\mathbf{Y}}_n$ can be seen as the self-taught ground truth used in a multi-class cross-entropy loss similar to eq 4.3. This value is either 1 if the predicted label for a pixel fools the discriminator or 0 otherwise.

4.3 Model architectures

To perform semantic segmentation, two approaches were tested. The first method that was tested is to use generative models that learn a representation of the input image and use this representation to generate a segmented image. Each pixel of the output image is compared to the labelled data using multi-class cross-entropy loss.

4.3.1 SegNet

The first model that was tested is SegNet[1]. This model uses an encoder-decoder architecture to generate the segmentations. Deep convolutions and max-pooling layers are used to encode the input image in a latent representation. This representation is then used in a decoder to generate an image where each pixel of the output is the predicted label of the equivalent input pixel. The decoder is symmetric to the encoder,

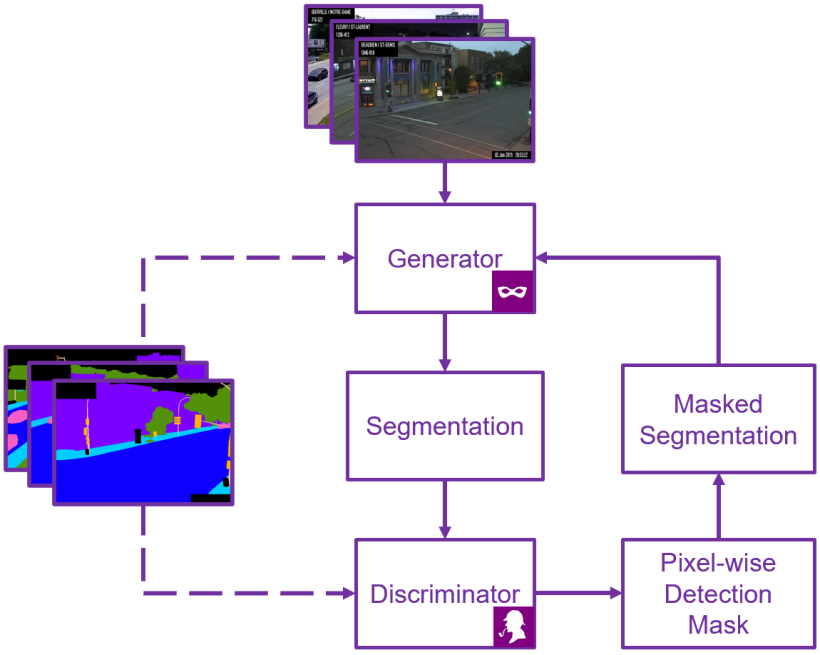


Figure 8: Adversarial Architecture. When labelled samples are fed to a model, supervised training is used to update the generator and discriminator models. If there is no available annotation, unsupervised learning is used instead.

and uses the same pooling indexes from the encoder to perform the upsampling. An alternative to using the encoder indexes is to use skip-connections to concatenate the max-pooling outputs in the encoder to the equivalent upsampling outputs in the decoder, as seen in U-Net models[15]. Another option is to combine both skip-connections and map-pooling indices, as used in U-SegNet variations[8]. As resources were limited, skip-connections were not used to reduce the number of parameters in the model. The final convolution output is run through a softmax layer to get a probability distribution for each pixel. Multi-class cross-entropy loss is used during training to tune the convolution weights. Batch normalization and ReLU activation are used after each convolution layer. Figure 9 shows the SegNet architecture that was used. To help with training, the encoder half is initialized with weights from the pre-trained VGG16[17] model included with Torchvision. This model is commonly used for classification tasks in computer vision. An extensive list of the parameters for each layer is given in Appendix C.

4.3.2 Fast-SCNN

The second semantic segmentation model that was tested is Fast-SCNN[14]. This model's main purpose is to perform faster than real-time semantic segmentation. The architecture builds on an encoder-decoder structure with two branches and adds multiple improvements to enhance efficiency. The two branches of the model are used to combine spatial information at higher resolutions and deep features obtained from lower resolution approximations of the input. One of the main performance enhancers is the usage of depth-wise separable convolutions as seen in MobileNet[16]. This method splits usual `conv2d` layers into a depth-wise convolution followed by a point-wise convolution. In a regular `conv2d` layer, each kernel is convoluted with all the input channels. For a depth-wise convolution, each kernel is convoluted

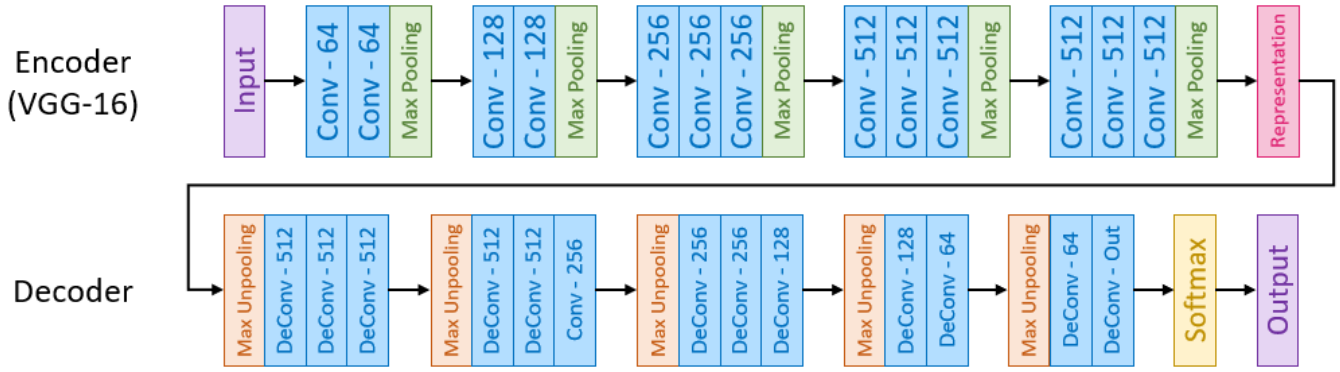


Figure 9: SegNet Architecture. Each `Conv` module includes a 2D Convolution followed by `Batch Normalization` and `ReLU` activation. The specified number for each `Conv` module is the number of channels of the output. The `Max Pooling` indexes are reused in the corresponding `Max Unpooling` layers.

with a single input channel, which greatly reduces the number of operations. This action is followed by the point-wise convolution to ensure that information is propagated through the channels. The point-wise convolution is a regular convolution with 1×1 kernels. Doing so greatly reduces the number of weights and floating-point operations when using a high number of channels for convolutions. The extensive usage of batch normalization after most convolutional layers is expected to reduce training time[6].

As seen in figure 10, the Fast-SCNN model architecture can be split into four distinctive sections:

1. Learning to Downsample: This first section uses a deep convolutional network to get features from several downsampled resolutions at the same time and pass them to the following sections. The first `conv2d` layer does not need to be a depth-wise separable convolution, as the benefit is marginal when the input has three channels.

2. Global Feature Extraction: This section uses residual bottleneck blocks taken from MobileNet[16] and pyramid pooling modules as seen in scene parsing[19] and segmentation[2] networks to get contextual information from the low resolution transformation of the input.

3. Feature Fusion: This section uses convolutional layers to get image features from the contextual information and spatial information obtained from the previous two modules. The features are then merged using a simple addition to maintain efficiency. Better performance at the cost of time efficiency could be obtained by using convolutional information merging methods. The merged features give the encoded representation of the input.

4. Classifier: This last section acts as a decoder to obtain semantic segmentation from the latent representation of the input. This mirrors the Learning to Downsample section with a three-layered deep convolutional network, followed by a final upsampling layer to match the input resolution.

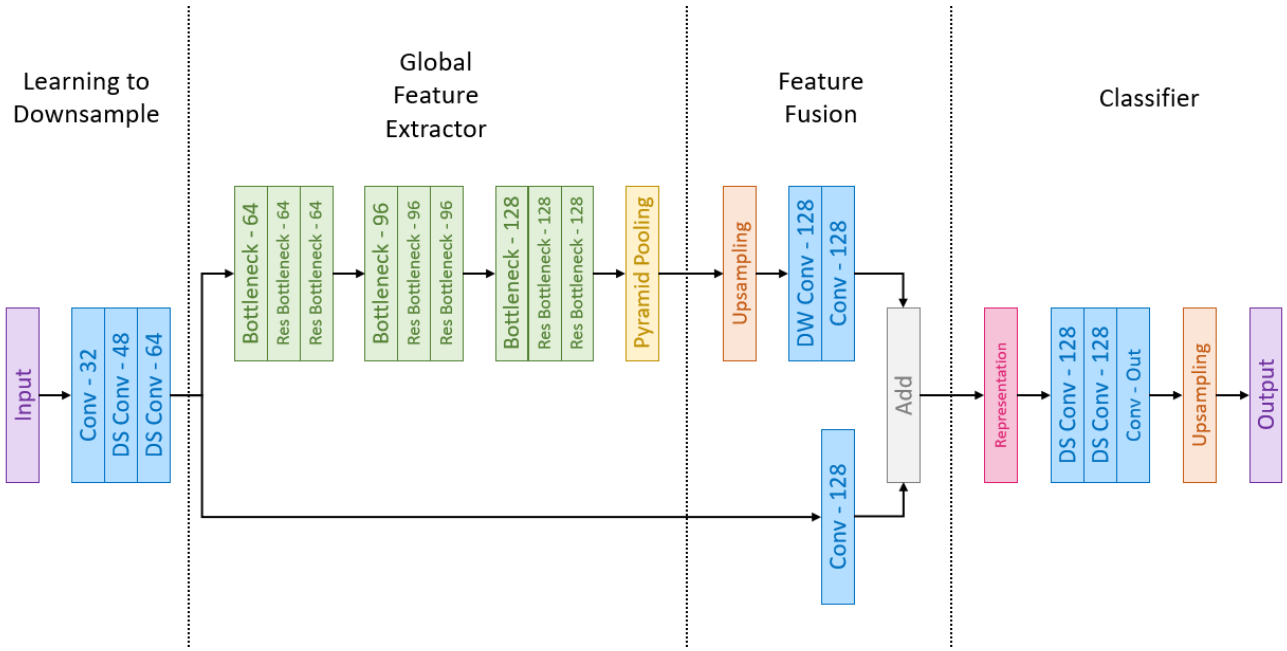


Figure 10: Fast-SCNN Architecture. The two branches harness both contextual and spatial features at the same time to encode the representation of the input. The specific architecture of the Bottleneck and Pyramid Pooling modules are shown in figures 17 and 18 in Appendix D.

A table with the parameters for all Fast-SCNN layers can be found in Appendix D.

4.3.3 Discriminator

When using a semi-supervised architecture, the generator model is either SegNet of Fast-SCNN. The discriminator model used is a deep convolutional network. The first five layers use a Leaky ReLU activation. Instead of discarding the negative values like ReLU does, Leaky ReLU attenuates them by a factor between 0 and 1. In this case, all Leaky ReLU activation functions use a factor $\alpha = 0.2$. This is done to help against vanishing gradients when training adversarial networks. The last layer uses a sigmoid activation to generate the output. Figure 11 shows the architecture of the discriminator model. A table with the parameters for each layer can be found in Appendix E.

4.4 Hyperparameters

As the models to be trained are taken from literature, the final architectural hyperparameters were left unchanged for both SegNet and Fast-SCNN. For supervised training, this leaves only the learning rate as a hyperparameter to be tuned. For semi-supervised training, additional hyperparameters are required to control the unsupervised iterations, namely λ_{adv}, λ_{semi}, T_{semi}. As this hyperparameter space has a high dimensionality, using grid-search to tune the hyperparameters would be too resource-intensive. For this reason, Bayesian optimization was selected as the tuning method. This method treats the training as a black-box function that takes the selected hyperparameters as input to output the model performance. After each training iteration, a prior distribution of the black-box is updated to form the posterior distribution of the performance function. Subsequent hyperparameters are then selected as to enhance this

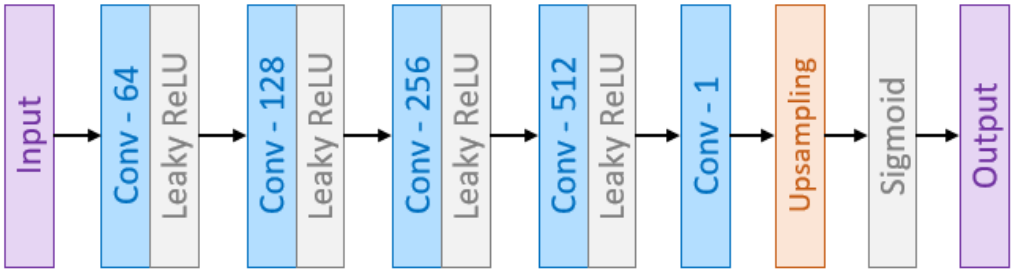


Figure 11: Discriminator Architecture for adversarial semi-supervised learning.

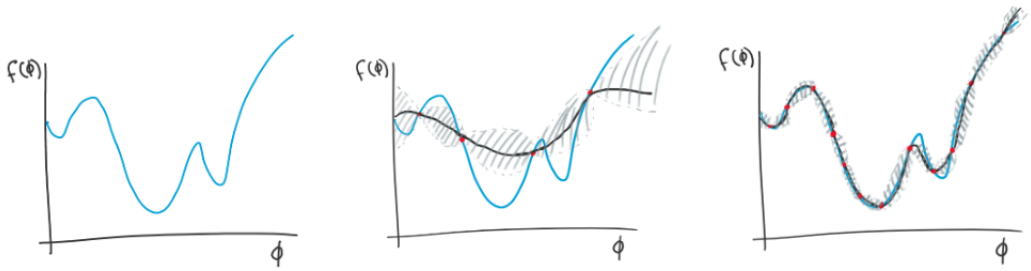


Figure 12: Bayes optimization for hyperparameters ϕ. As more experimental points are available, the estimate (gray) of the performance function (blue) gets more accurate.

prediction. Using this method, 20 sets of hyperparameters were run for each experiment to find the best set of hyperparameters.

4.5 Computational resources

For the largest part of the project, training was done using Google Colab notebooks and a shared Google Drive. This limited the training time to 12 hours per experiment and limited the amount of experiments that could be run on any given week, as the transfer limit of the shared drive was sometimes reached when loading 60GB of data daily. Once the CGMU dataset was available and the initial proof of concept was completed, everything was transferred to a Google Cloud Platform instance with a permanent drive for the datasets and model weights. The GPU used was upgraded from Colab's NVIDIA K80 to a NVIDIA P100. It was also possible to queue longer experiments without needing to restart from a checkpoint. This greatly enhanced the training process.

5 Experimental Performance

The best results for each experiment can be found in table 2. Due to resource constraints, not all experiments could be run. Namely, the semi-supervised proof on concept was only run on Cityscapes and Carla was omitted. Of the two datasets, Cityscapes was selected for this test as the image features were similar to the ones in CGMU. The superior results were enough to justify using semi-supervised learning with the

Dataset	Supervised				Semi-supervised			
	Fast-SCNN		SegNet		Fast-SCNN		SegNet	
	Acc	IoU	Acc	IoU	Acc	IoU	Acc	IoU
Carla	0.89	0.47	0.82	0.39				
CityScapes	0.85	0.47	0.75	0.33	0.92	0.90	0.85	0.81
CGMU	0.76	0.27	0.58	0.15	0.88	0.39	0.87	0.39
CGMU_S	0.71	0.21	0.65	0.17				
CGMU_L	0.79	0.29	0.63	0.17	0.89	0.40		

Table 2: Best experimental results for each dataset and method. Grayed out cells are omitted experiments. All experiments were run with 20 sets of hyperparameters and the best results on an unseen test set are shown here.

CGMU dataset. Additionally, as the resolution-based split of the CGMU dataset was done near the end of the project to explore the effect of using subsets on performance, a single semi-supervised experiment was completed.

5.1 Proof of concept

The first set of experiments was run in a supervised manner on the Carla and Cityscapes datasets to get a comparison of the results of both models. As seen in table 2, SegNet is globally outperformed by Fast-SCNN . This is expected, as there were multiple advancements in the field of computer vision during the 4 years that separate both models. For both models, the results were better on Carla than Cityscapes. This behavior can be explained by the similarity between inputs and the lack of noise when using Carla. Furthermore, results using the CGMU dataset are expected to be lower than those on Cityscapes, as the images are of lower resolution and noisier.

Adding unlabelled images to allow for semi-supervised training lead to a great jump in performance when tested using the Cityscapes dataset. For both Fast-SCNN and SegNet, almost half of the previously wrongly annotated pixels were now correctly annotated. This is concordant with a massive jump in average IoU.

5.2 CGMU Results

The first experiments ran on the CGMU dataset did not include the resolution-based split. As predicted, the final results were significantly lower than with Carla or Cityscapes datasets. The SegNet results being significantly worse could be explained by the low resolution of the input images. However, when using the semi-supervised pipeline, the results for both models were similar, showing an accuracy of 88% with an IoU of 0.39.

The samples that generated poor results were explored and a trend was noticed with samples with a lower resolution. To validate this intuition, new models were trained using the small and large splits of the CGMU dataset in a supervised manner. It was found that lower resolution images were indeed lowering the global accuracy and IoU, suggesting that a different model dedicated to only high resolution images could be used for newer cameras to improve results. However, when testing the large image dataset in a semi-supervised manner, the results were similar to those obtained using the full dataset.

Dataset	Supervised		Semi-supervised	
	Acc	IoU	Acc	IoU
CGMU_S	0.76	0.31	0.82	0.43
CGMU_L	0.82	0.42	0.90	0.68

Table 3: Best experimental results for the small and large resolution CGMU datasets using Fast-SCNN and static object labels.

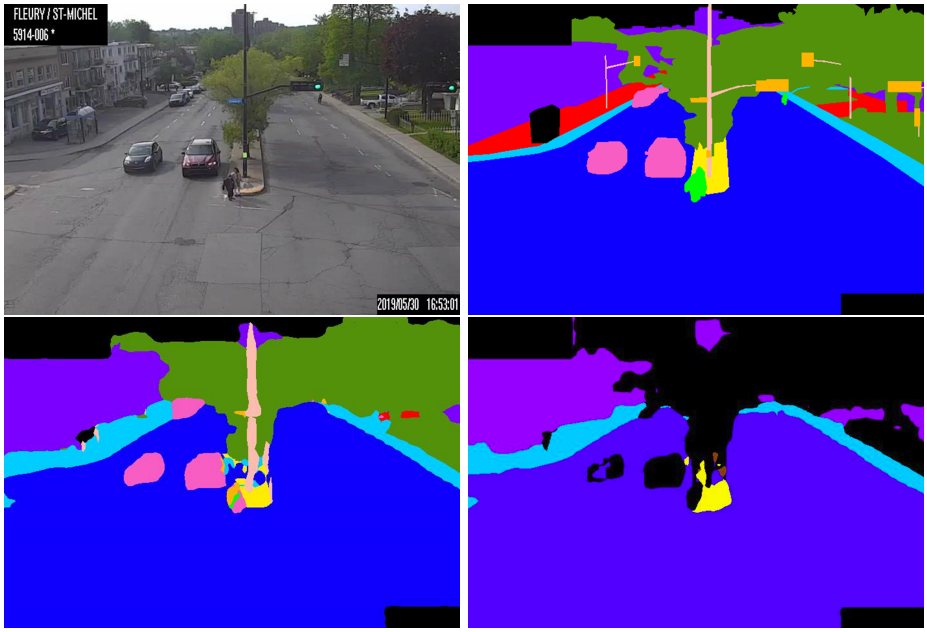


Figure 13: Example of a correctly annotated image. In order from top to bottom, left to right: RGB image, Ground truth annotation, Annotation with all labels, Annotations without dynamic objects

Discarding Labels from Dynamic Objects

As the objective of this block is to generate masks for regions of interest and that dynamic object detection is planned for a future block, some experiments were done with all the dynamic object annotations replaced by the void label. The labels that were kept are *void, median strip, building, sidewalk, road* and *structure*. These experiments were performed on the small and large resolution CGMU datasets using Fast-SCNN. Table 3 shows the results. This change lead to a slightly better accuracy and a significant rise in IoU. Figure 13 shows an example of an image annotation from a Fast-SCNN model trained with a semi-supervised approach. Both models are unable to distinguish the top-left private area from the sidewalk. The model with dynamic labels has difficulty with the pedestrian in the middle of the image and the median strip whereas the model with static labels performs well.

5.3 Next Steps

When exploring the results of the semi-supervised method, all models were having problems segmenting the same images, thus lowering the global accuracy. The two main sources of error were images taken during the night or images taken from unique points of view. Figure 14 shows two examples where this is

the case. The second example proves that the model is still able to perform segmentation on the right side of the image, but the left side with the underpass is poorly segmented. This finding suggests that the 90% accuracy limit displayed by most models might be a result of images that are underrepresented or harder to label. To verify this hypothesis, a new dataset with images taken during a sunny day could be tested.

To test the impact of unique points of view, it would be possible to test the results of models trained on a subset of cameras. These subsets could be generated using the camera contexts listed in section 2.1 or by manually selecting a few cameras with a particular feature like Décarie Boulevard or the presence of a median strip.

Another feature that seems to have an impact on the results is the weather. As a wet road and sidewalk are darker than normal, the models have a harder time with segmentation near these edges. A new dataset could be generated with weather in mind to record its impact on the results.

A different approach that could improve the results for different weather conditions and unique points of view would be transfer learning. Recent articles[10][11] explore the impact of various regularization schemes to fine-tune pre-trained models for specific tasks or conditions. Using similar methods, the models presented in this document could be fine-tuned to each subset to improve individual performances. However, this improvement would require multiple models and would have additional storage and implementation overheads. The camera id, current weather conditions and/or time of day would need to be provided along with the video feed to identify which model to load. This might be unneeded if the current models perform adequately to generate masks for future blocks.

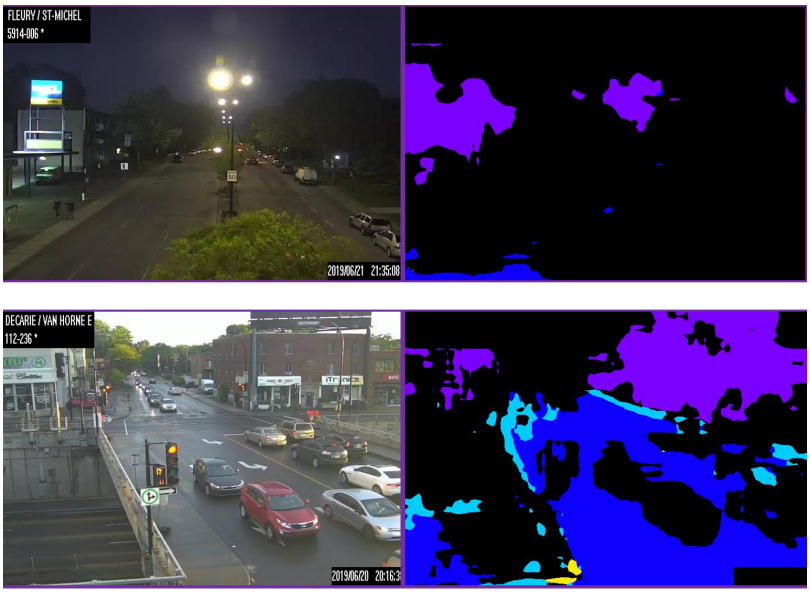


Figure 14: Two examples of images that are poorly segmented by all models.

6 Conclusion

With the results provided in this work, a fast Fast-SCNN model trained in a semi-supervised fashion seems like an adequate solution proposed by Ville de Montréal. While the final model performs well in most cases, some corner-cases related to weather, time of day or unique points of view still need to be fixed. However, with the long-term goal being to help operators passively monitor live feeds for anomalies and the short-term goal being a model to generate masks for a soon-to-be object detection task, this project is a success.

However, there is still room for improvement if resources are deemed available and necessary. With machine learning and especially computer vision being in great expansion lately, old findings are quickly rendered obsolete by new studies. Fast-SCNN was published two months before the internship started, and the semi-supervised approach that was employed was published around the middle of the internship. As such, it could prove necessary to review the latest findings periodically, as new state-of-the-art methods are frequently found.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[2] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018.

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[4] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[5] Wei-Chih Hung, Yi-Hsuan Tsai, Yan-Ting Liou, Yen-Yu Lin, and Ming-Hsuan Yang. Adversarial learning for semi-supervised semantic segmentation. *CoRR*, abs/1802.07934, 2018.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[7] Tarun Kalluri, Girish Varma, Manmohan Chandraker, and C. V. Jawahar. Universal semi-supervised semantic segmentation. *CoRR*, abs/1811.10323, 2018.

[8] Pulkit Kumar, Pravin Nagar, Chetan Arora, and Anubha Gupta. U-segnet: Fully convolutional neural network based automated brain tissue segmentation tool. *CoRR*, abs/1806.04429, 2018.

[9] Jungbeom Lee, Eunji Kim, Sungmin Lee, Jangho Lee, and Sungroh Yoon. Ficklenet: Weakly and semi-supervised semantic image segmentation\\using stochastic inference. *CoRR*, abs/1902.10421, 2019.

[10] Xuhong Li, Yves Grandvalet, and Franck Davoine. A baseline regularization scheme for transfer learning with convolutional neural networks. *Pattern Recognition*, 98:107049, 2020.

[11] Xuhong Li, Yves Grandvalet, Franck Davoine, Jingchun Cheng, Yin Cui, Hang Zhang, Serge Belongie, Yi-Hsuan Tsai, and Ming-Hsuan Yang. Transfer learning in computer vision tasks: Remember where you come from. *Image and Vision Computing*, 93:103853, 2020.

[12] Jeffrey Liu, Andrew J. Weinert, and Saurabh Amin. Semantic analysis of traffic camera data: Topic signal extraction and anomalous event detection. *CoRR*, abs/1905.07332, 2019.

[13] Jilin Mei, Biao Gao, Donghao Xu, Wen Yao, Xijun Zhao, and Huijing Zhao. Semantic segmentation of 3d lidar data in dynamic scene using semi-supervised learning. *CoRR*, abs/1809.00426, 2018.

[14] Rudra P. K. Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-scnn: Fast semantic segmentation network. *CoRR*, abs/1902.04502, 2019.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[16] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[18] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. *CoRR*, abs/1802.10349, 2018.

[19] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

Appendix

A Labels

Carla

Tag	Value	RGB	Colour
None	0	(0, 0, 0)	
Buildings	1	(70, 70, 70)	
Fences	2	(190, 153, 153)	
Other	3	(250, 170, 160)	
Pedestrians	4	(220, 20, 60)	
Poles	5	(153, 153, 153)	
Road Lines	6	(157, 234, 50)	
Roads	7	(128, 64, 128)	
Sidewalks	8	(244, 35, 232)	
Vegetation	9	(107, 142, 35)	
Vehicles	10	(0, 0, 142)	
Wall	11	(102, 102, 156)	
Traffic Signs	12	(220, 220, 0)	

Table 4: List of semantic tags pre-built in Carla, as well as their colour.

Cityscapes

Tag	Value	RGB	Colour
Unlabeled	0	(0,0, 0)	
Self Vehicle	1	(0,0, 0)	
Rectification Border	2	(0,0, 0)	
Out of ROI	3	(0,0, 0)	
Static	4	(0,0, 0)	
Dynamic	5	(111,74, 0)	
Ground	6	(81,0, 81)	
Road	7	(128,64, 128)	
Sidewalk	8	(244,35, 232)	
Parking	9	(250,170, 160)	
Rail Track	10	(230,150, 140)	
Building	11	(70,70, 70)	
Wall	12	(102,102, 156)	
Fence	13	(190,153, 153)	
Guard Rail	14	(180,165, 180)	
Bridge	15	(150,100, 100)	
Tunnel	16	(150,120, 90)	
Pole	17	(153,153, 153)	
Polegroup	18	(153,153, 153)	
Traffic Light	19	(250,170, 30)	
Traffic Sign	20	(220,220, 0)	
Vegetation	21	(107,142, 35)	
Terrain	22	(152,251, 152)	
Sky	23	(70,130, 180)	
Person	24	(220,20, 60)	
Rider	25	(255,0, 0)	
Car	26	(0,0, 142)	
Truck	27	(0,0, 70)	
Bus	28	(0,60, 100)	
Caravan	29	(0,0, 90)	
Trailer	30	(0,0, 110)	
Train	31	(0,80, 100)	
Motorcycle	32	(0,0, 230)	
Bicycle	33	(119,11, 32)	
License Plate	34	(0,0, 142)	

Table 5: List of semantic tags used to annotate Cityscapes images, as well as their colour.

CGMU

Tag	Value	RGB	Colour
Void	0	(0,0,0)	
Pole	1	(255,189,176)	
Traffic Sign	2	(255,181,0)	
Vehicle	3	(247,93,195)	
Vegetation	4	(83,145,11)	
Median Strip	5	(255,236,0)	
Building	6	(123,0,255)	
Private	7	(255,0,0)	
Sidewalk	8	(0,205,255)	
Road	9	(14,0,255)	
Pedestrian	10	(0,255,4)	
Structure	11	(134,69,15)	
Construction	12	(255,99,0)	

Table 6: List of semantic tags used to annotate CGMU images, as well as their colour.

B QA steps

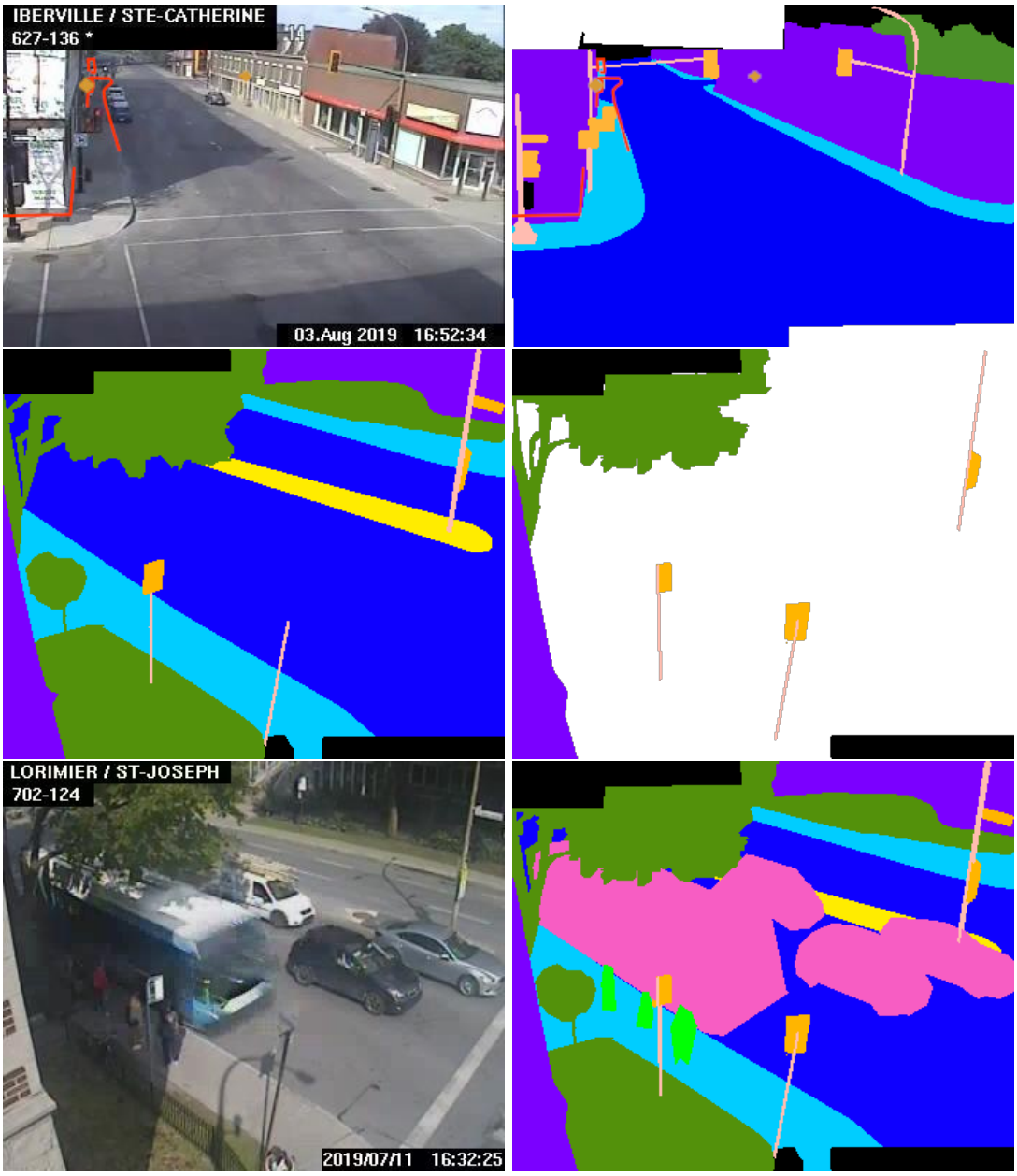


Figure 15: The QA process. The first row shows and example of feedback given to annotators about mislabelled areas on the RGB image (top left) and annotated image (top right). The second and third rows show the different layers used to create an annotation. The Mother Frame (middle left) and Occlusion (middle right) layers are shared for multiple images that have the same PTZ alignment. The manual annotation on this particular RGB image (bottom left) are stacked with the shared layers to obtain the final annotation (bottom right).

C SegNet Parameters

Layer	Channels	Kernel Size	Stride	Padding
Encoder				
Stage 0				
Conv 2D 0	64	3	1	1
Conv 2D 1	64	3	1	1
Max Pooling	-	2	2	0
Stage 1				
Conv 2D 0	128	3	1	1
Conv 2D 1	128	3	1	1
Max Pooling	-	2	2	0
Stage 2				
Conv 2D 0	256	3	1	1
Conv 2D 1	256	3	1	1
Conv 2D 2	256	3	1	1
Max Pooling	-	2	2	0
Stage 3				
Conv 2D 0	512	3	1	1
Conv 2D 1	512	3	1	1
Conv 2D 2	512	3	1	1
Max Pooling	-	2	2	0
Stage 4				
Conv 2D 0	512	3	1	1
Conv 2D 1	512	3	1	1
Conv 2D 2	512	3	1	1
Max Pooling	-	2	2	0
Decoder				
Stage 4				
Max Unpooling	-	2	2	0
Conv Transpose 2D 0	512	3	1	1
Conv Transpose 2D 1	512	3	1	1
Conv Transpose 2D 2	512	3	1	1
Stage 3				
Max Unpooling	-	2	2	0
Conv Transpose 2D 0	512	3	1	1
Conv Transpose 2D 1	512	3	1	1
Conv Transpose 2D 2	256	3	1	1
Stage 2				
Max Unpooling	-	2	2	0
Conv Transpose 2D 0	256	3	1	1
Conv Transpose 2D 1	256	3	1	1
Conv Transpose 2D 2	128	3	1	1
Stage 1				
Max Unpooling	-	2	2	0
Conv Transpose 2D 0	128	3	1	1
Conv Transpose 2D 1	64	3	1	1
Stage 0				
Max Unpooling	-	2	2	0
Conv Transpose 2D 0	64	3	1	1
Conv Transpose 2D 1	*Nb labels*	3	1	1

D Fast-SCNN

Fast-SCNN Module Architectures

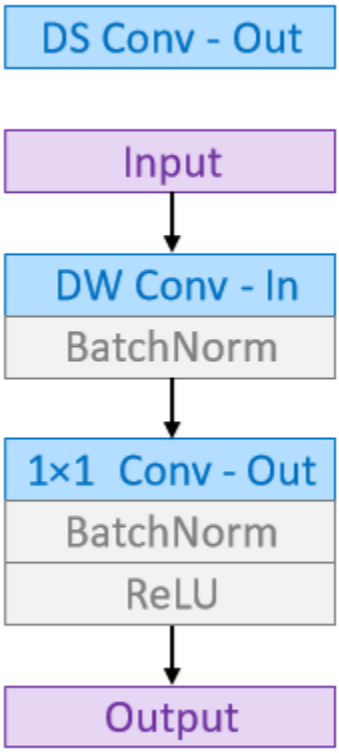


Figure 16: Layers in a Depth-wise Separable Convolution. The depth-wise convolution uses the `groups` argument of `torch.nn.conv2d`. The separable convolution uses 1×1 kernels.

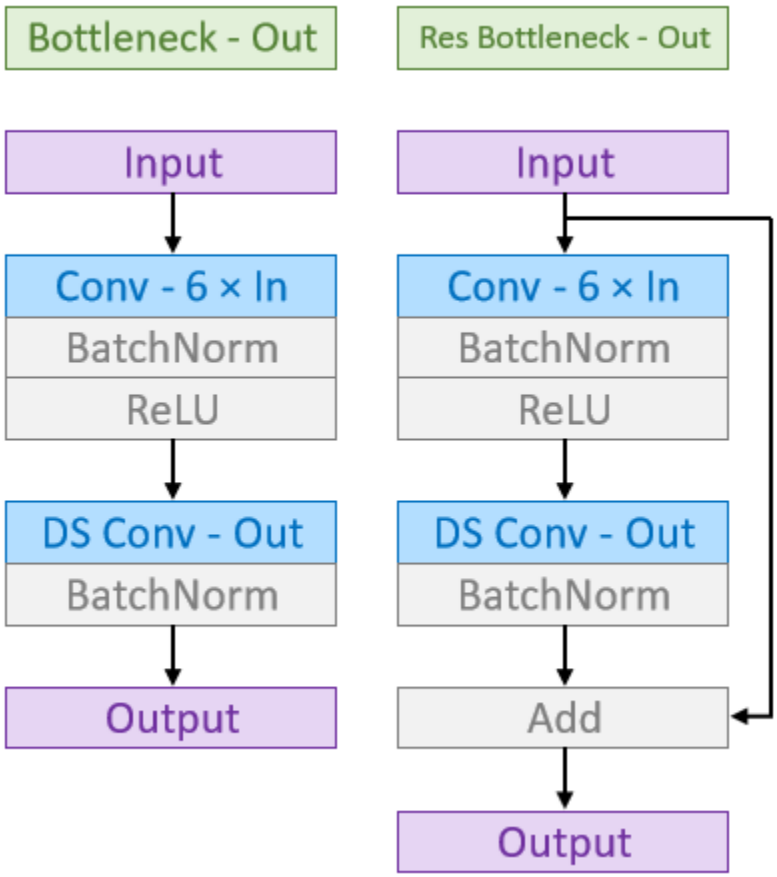


Figure 17: Bottleneck and Residual Bottleneck architectures as used in Fast-SCNN. The first convolution of the Bottleneck module uses an expansion factor to raise the number of channels. The residual variant can be used when the input and output have the same size and number of channels.

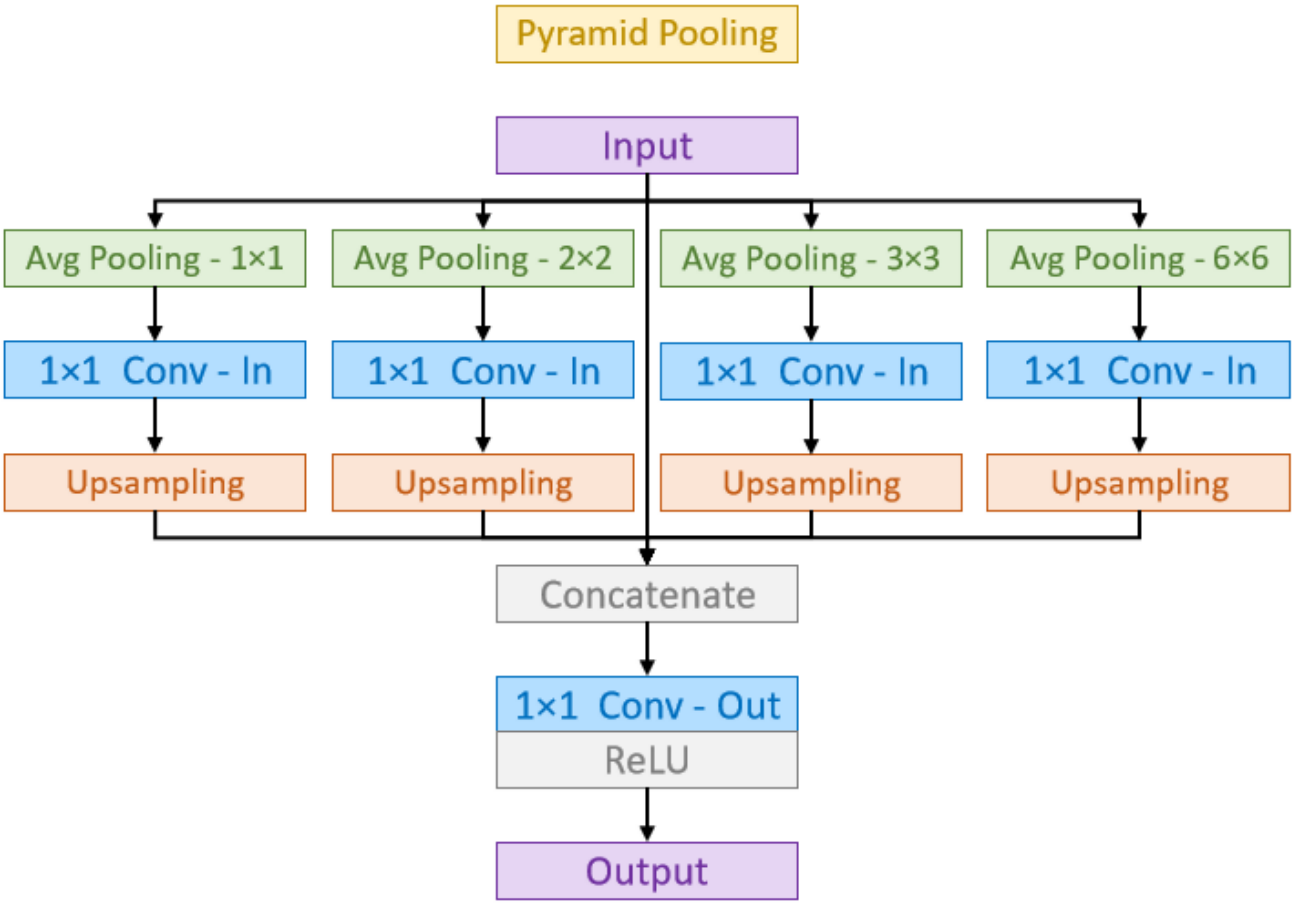


Figure 18: Pyramid Pooling Module architecture as used in Fast-SCNN. The 1×1 Conv modules are convolutions with single-pixel kernels.

Fast-SCNN Parameters

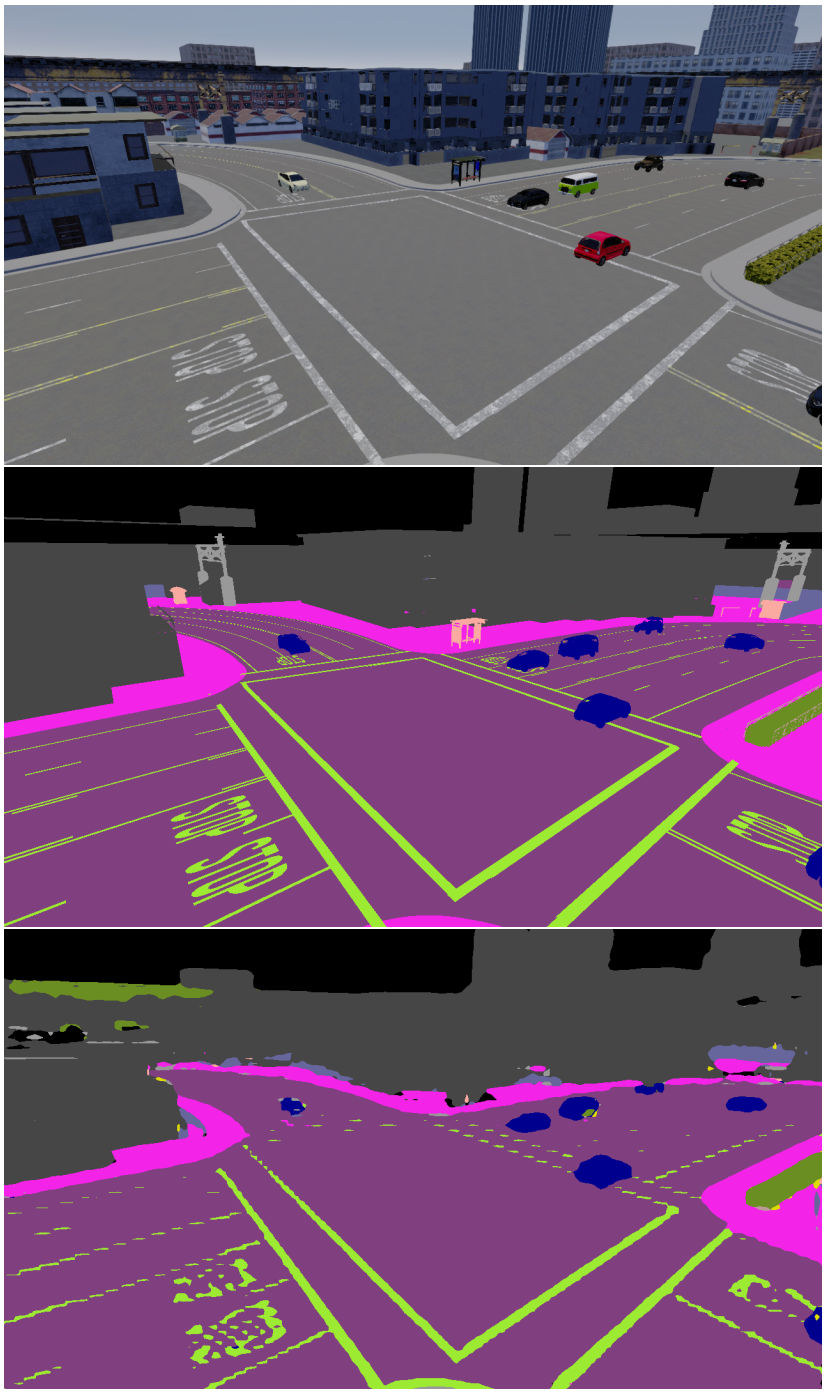
Layer	Channels	Kernel Size	Stride	Padding	Groups	Dilation
Bottleneck / Residual Bottleneck						
Conv 2D	$6 \times$ *In channels*	1	1	0		
Conv 2D (depth-wise)	$6 \times$ *In channels*	3	*stride*	1	$6 \times$ *In channels*	
Conv 2D (point-wise)	*channels*	1	1	0		
Learning to Downsample						
Conv 2D	32	3	2	1		
DS Conv 1						
Conv 2D (depth-wise)	32	3	2	1	32	
Conv 2D (point-wise)	48	1	1	0		
DS Conv 2						
Conv 2D (depth-wise)	48	3	2	1	48	
Conv 2D (point-wise)	64	1	1	0		
Global Feature Extractor						
Bottleneck Module 1						
Bottleneck	64		2			
Residual Bottleneck	64		1			
Residual Bottleneck	64		1			
Bottleneck Module 2						
Bottleneck	64		2			
Residual Bottleneck	64		1			
Residual Bottleneck	64		1			
Bottleneck Module 3						
Bottleneck	64		1			
Residual Bottleneck	64		1			
Residual Bottleneck	64		1			
Pyramid Pooling Module						
$4 \times$ Conv 2D	128	1	1	0		
Conv 2D	128	1	1	0		
Feature Fusion						
Conv 2D (depth-wise)	128	3	1	4	128	4
Conv 2D (context)	128	1	1	0		
Conv 2D (spatial)	128	1	1	0		
Classifier						
DS Conv 1						
Conv 2D (depth-wise)	128	3	1	0	128	
Conv 2D (point-wise)	128	1	1	0		
DS Conv 2						
Conv 2D (depth-wise)	128	3	1	0	128	
Conv 2D (point-wise)	128	1	1	0		
Conv 2D	*Nb labels*					

E Discriminator Parameters

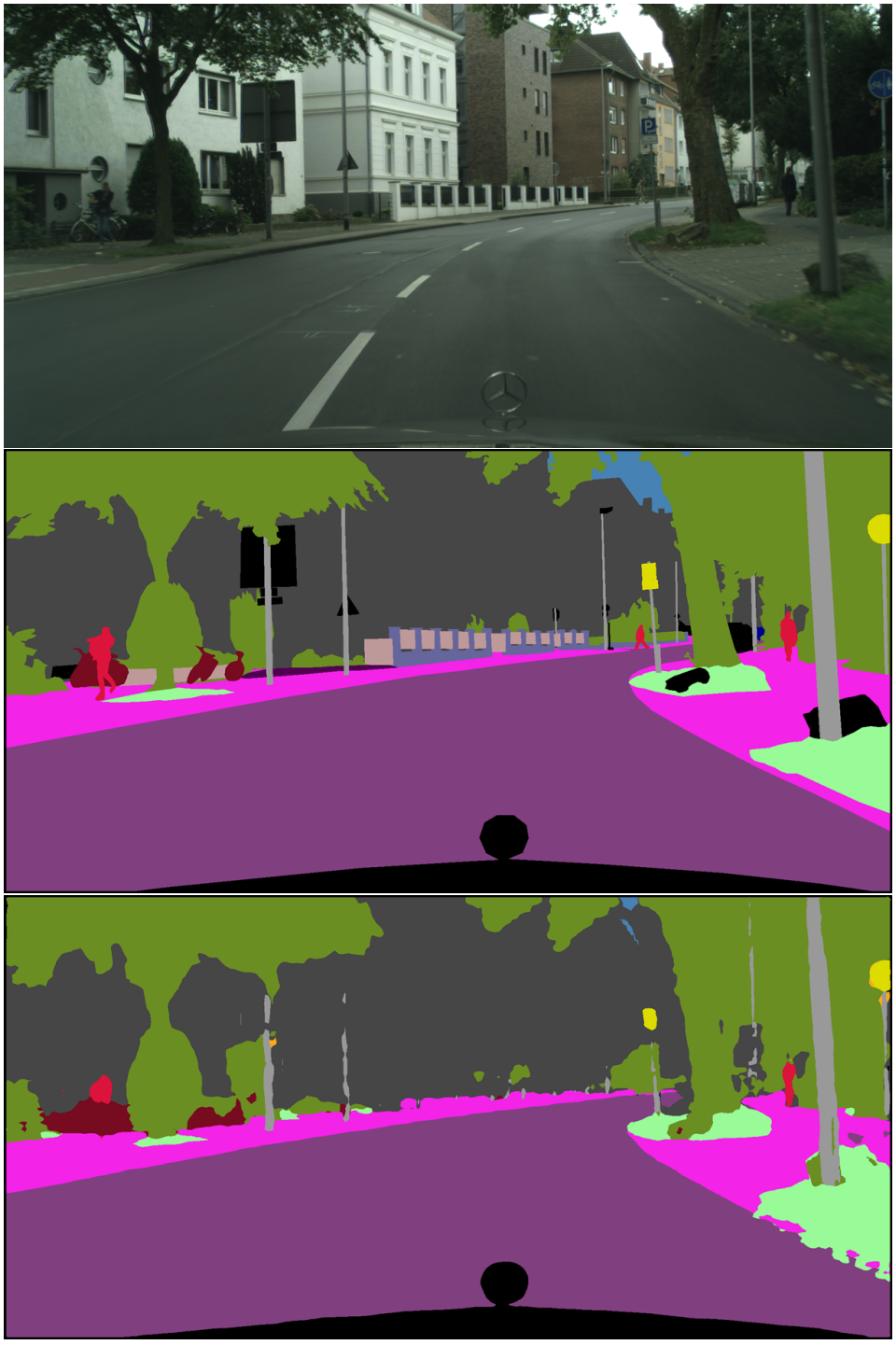
Layer	Channels	Kernel Size	Stride	Padding
Conv 2D	64	4	2	1
Conv 2D	128	4	2	1
Conv 2D	256	4	2	1
Conv 2D	512	4	2	1
Conv 2D	1	4	2	1

F Segmented Results

Carla



CityScapes



CGMU