Cover sheet

Faculty Name: computer Science & artificial Intelligence

Course Name: Selected-2 In Computer Science

Team number:24

Full Name	ID
ايه نادر محمد عويس	202000189
الهام محمود محمد عيسى	202000147
ربى هشام عبدالدايم محمد	202000309
عمرو ايمن عابد عبدالواحد	202000615
كريم عماد حسن التهامى	202000668
ولاء يوسف عادل محمد	202001045

Lightweight Gaussian-Based Model for Fast Detection and Classification of Moving Objects

05.06.2023

Author Name: Joaquin Palma-Ugarte , Laura Estacio-Cerquin , Victor Flores-Benites4 and Rensso Mora-Colque

Publisher Name: by Scitepress -Science and Technology Publications, Lda. Under CC license

Year of Publication: 2023

Dataset: Kitti The KITTI Vision Benchmark Suite (cvlibs.net)

The implemented algorithms:

paper includes (TRG-NET, Faster R-CNN, SSD-Lite, Retina NET)

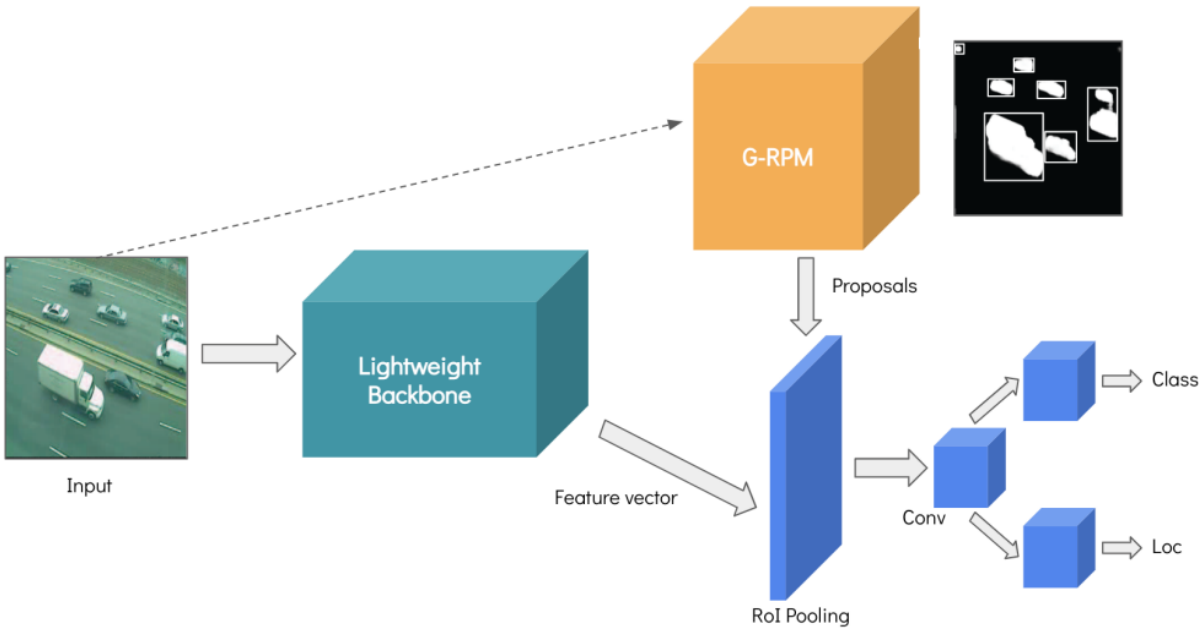
In our project we used (TRG-NET, Faster R-CNN)

TRG-NET: a unified model that can be executed on computationally limited devices to detect and classify just moving objects.

This proposal is based on the Faster R-CNN architecture, MobileNetV3 as a feature extractor, and a Gaussian mixture model for a fast search of Regions of Interest based on motion.

TRG-Net reduces the inference time by unifying moving object detection and image classification tasks, and by limiting the regions of interest to the number of moving objects.

TRG-Net uses two-stage architecture to generate region proposals first and then classify each proposal into their respective categories.



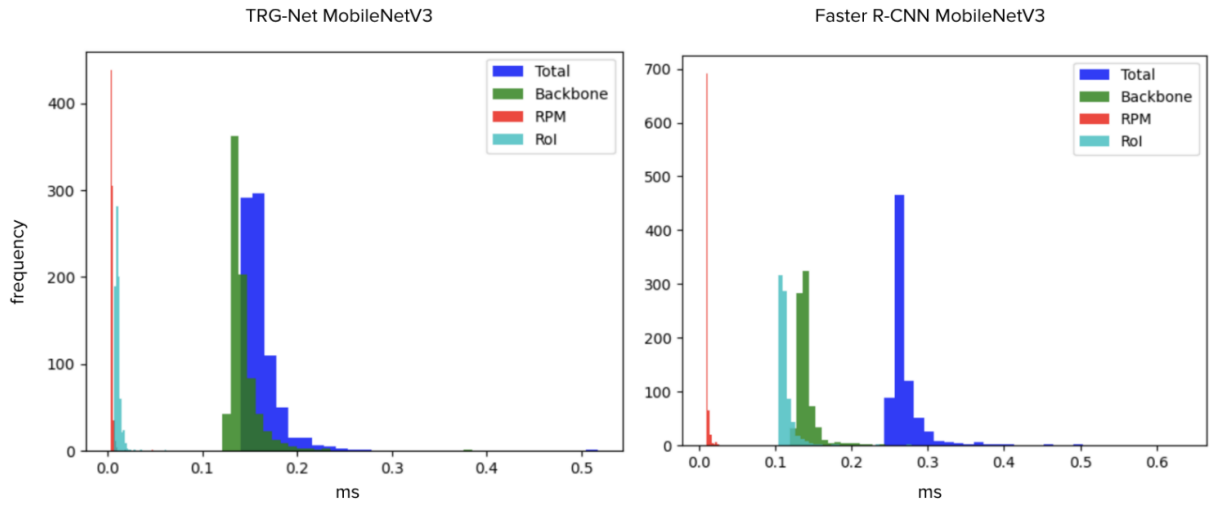
TRG-Net architecture. During inference, the input image passes through a backbone to obtain its feature vector. Then, the same input trains the G-RPM model that returns a list of region proposals. The previous two outputs pass through a pooling layer and fully connected layers that return the classes and locations of the moving objects.

Faster R-CNN:

- The TRG-NET model is inspired by Faster R-CNN.
- The most widely used two-stage architecture is Faster R-CNN
- Faster R-CNN is a deep convolutional network used for object detection, that appears to the user as a single, end-to-end, unified network. The network can accurately and quickly predict the locations of different objects.

- Faster R-CNN is based on MobileNetV3 as feature extractor

Mobile-NET V3:

- It is a backbone for TRG-NET & Faster R-CNN
- Mobile-NET is used to obtain feature vectors.
- Faster R-CNN with MobileNetV3 and TRG-net is due to the use of a traditional method to discover regions of interest.
- 

- We have another backbone called RES-NET50. When we use the RES-NET 50 we get the highest Average precision BUT nevertheless we get the highest inference time, so we decided to use the Mobile-NET cause there is a balance between average precision and inference time.

G-RPM:

- TRG-Net uses a G-RPM, which provides regions of interest based on motion.
- Using a GRPM would cause loss of information,This is because the number of training objects is reduced when considering only moving objects,So to use a G-RPM, we would need the bounding boxes and labels of all the moving objects within a video.

Result:

Model	Backbone	AP	# Parameters	Inference Time
TRG-Net (ours)	MobileNetV3	0.423	18.30 M	0.138 s
Faster R-CNN	MobileNetV3	0.423	18.91 M	0.221 s
	ResNet50	**0.519**	41.53 M	4.702 s
SSD Lite	MobileNetV3	0.283	**6.96 M**	**0.098 s**
RetinaNet	ResNet50	0.492	33.8 M	4.501 s

General information on the selected dataset:

The name of the dataset:Self-Driving Cars

Link: Self-Driving Cars | Kaggle

Total number of samples: 165K

The dimension of images: 300*480

Number of classes: 5 classes ->cars

-> trucks

->pedestrian

->bicyclist

->light

Implementation details:

Ratio that used for training : 70%

Ratio that used for validation : 15%

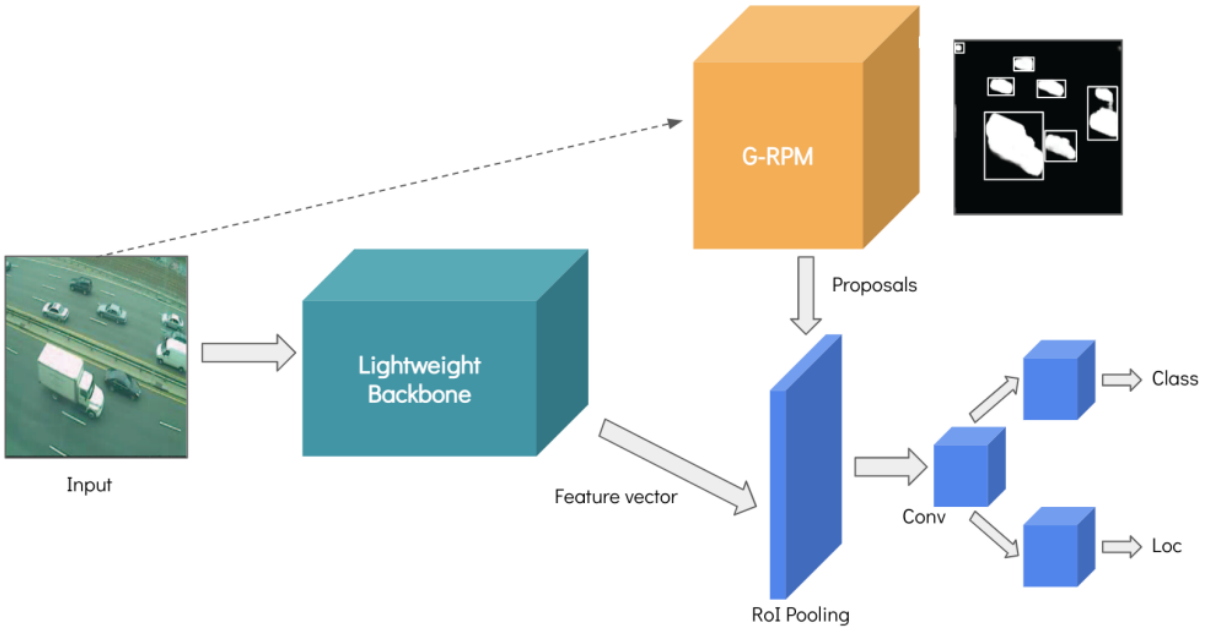
Ratio that used for testing : 15%

Number of images in training set : **952**

Number of images in validation set: **133**

Number of images in testing set: **133**

A block diagram of our implemented model :



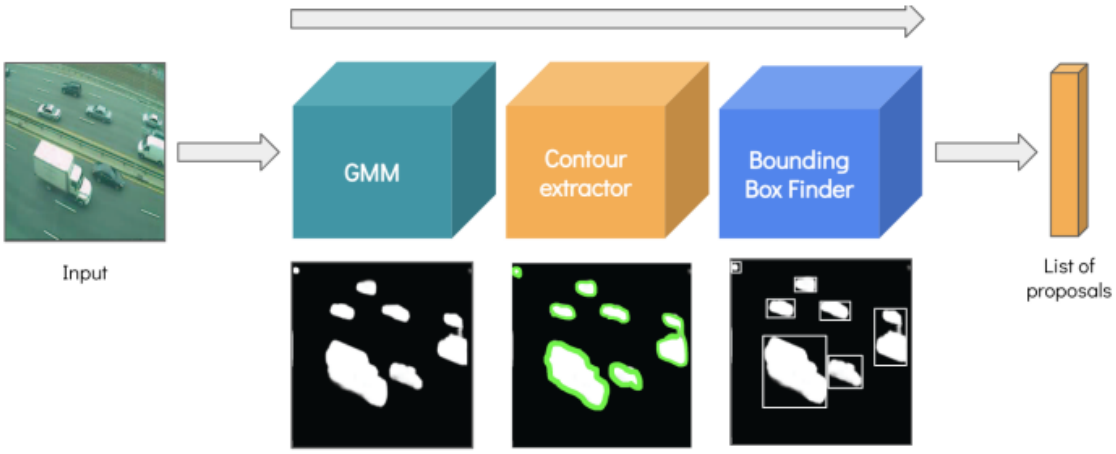


Figure 2: Gaussian-based Region Proposal Model.

Specify any hyperparameters used in our model:

Lr = 0.005

momentum=0.9

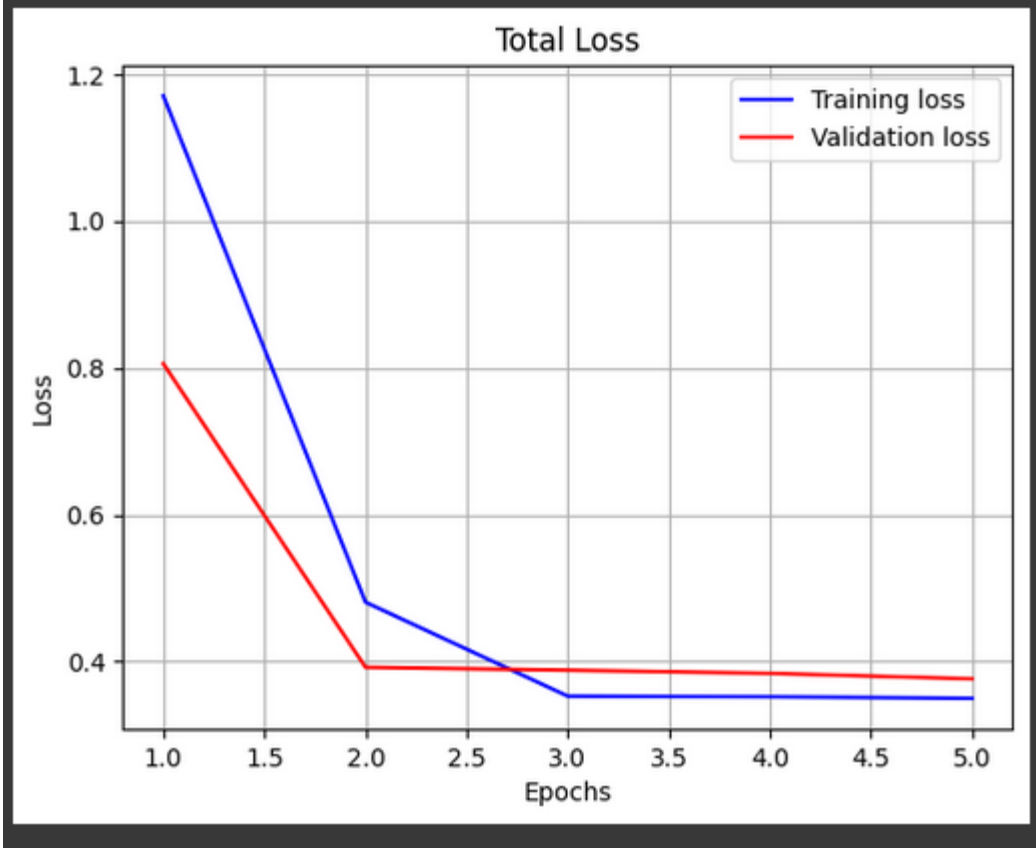
weight_decay=0.0005

Results details:

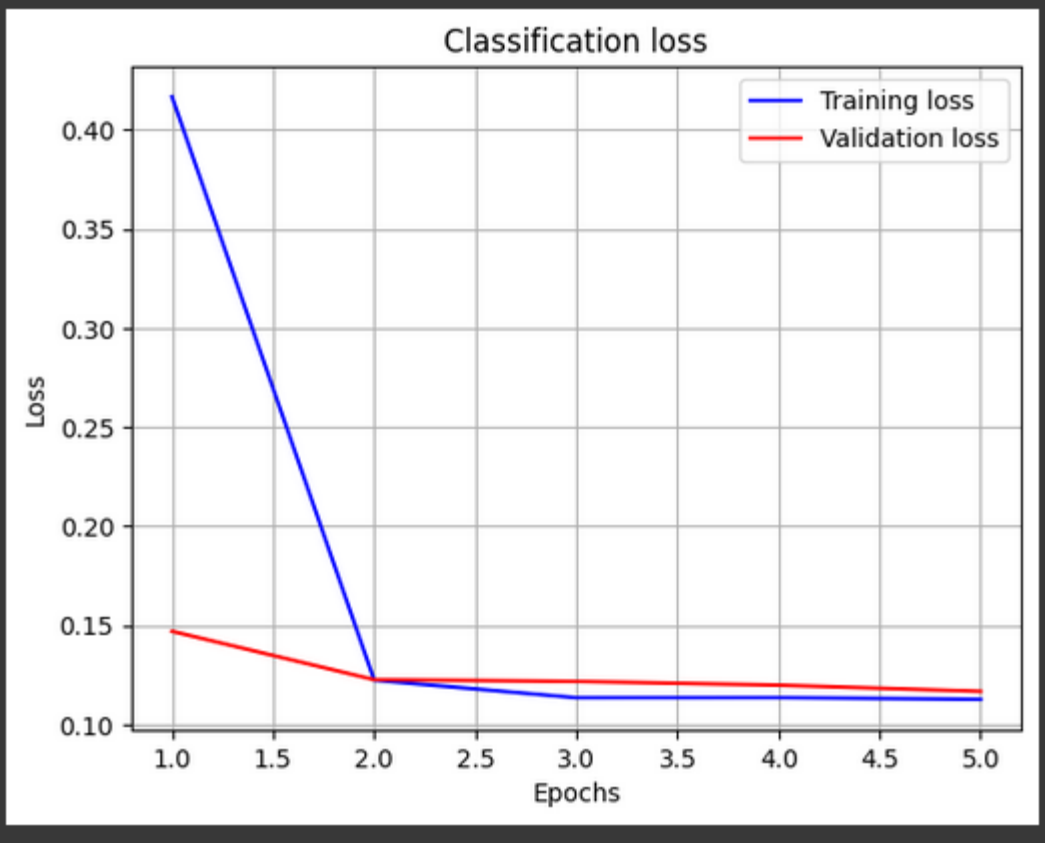
```
EPOCH 1
Training - Total Loss 1.17042 | Classification Loss 0.41674 | Regression Loss 0.03985  | Object Loss 0.62067 | RPN Loss 0.09315
Validation - Total Loss 0.80557 | Classification Loss 0.14701 | Regression Loss 0.03910  | Object Loss 0.52468 | RPN Loss 0.09479
Testing - Total Loss 0.80570 | Classification Loss 0.14702 | Regression Loss 0.03910  | Object Loss 0.52480 | RPN Loss 0.09479
================================================================================ Done

EPOCH 2
Training - Total Loss 0.48084 | Classification Loss 0.12235 | Regression Loss 0.02446  | Object Loss 0.26640 | RPN Loss 0.06763
Validation - Total Loss 0.39219 | Classification Loss 0.12260 | Regression Loss 0.01681  | Object Loss 0.18100 | RPN Loss 0.07179
Testing - Total Loss 0.39086 | Classification Loss 0.12259 | Regression Loss 0.01681  | Object Loss 0.17967 | RPN Loss 0.07179
================================================================================ Done

EPOCH 3
Training - Total Loss 0.35294 | Classification Loss 0.11348 | Regression Loss 0.01597  | Object Loss 0.16507 | RPN Loss 0.05841
Validation - Total Loss 0.38827 | Classification Loss 0.12172 | Regression Loss 0.01824  | Object Loss 0.17620 | RPN Loss 0.07211
Testing - Total Loss 0.38917 | Classification Loss 0.12171 | Regression Loss 0.01824  | Object Loss 0.17710 | RPN Loss 0.07211
================================================================================ Done

EPOCH 4
Training - Total Loss 0.35204 | Classification Loss 0.11351 | Regression Loss 0.01772  | Object Loss 0.16280 | RPN Loss 0.05801
Validation - Total Loss 0.38395 | Classification Loss 0.11983 | Regression Loss 0.01752  | Object Loss 0.17443 | RPN Loss 0.07217
Testing - Total Loss 0.38527 | Classification Loss 0.11982 | Regression Loss 0.01752  | Object Loss 0.17576 | RPN Loss 0.07217
================================================================================ Done

EPOCH 5
Training - Total Loss 0.34997 | Classification Loss 0.11268 | Regression Loss 0.01772  | Object Loss 0.16193 | RPN Loss 0.05764
Validation - Total Loss 0.37649 | Classification Loss 0.11672 | Regression Loss 0.01542  | Object Loss 0.17244 | RPN Loss 0.07191
Testing - Total Loss 0.37822 | Classification Loss 0.11674 | Regression Loss 0.01542  | Object Loss 0.17415 | RPN Loss 0.07191
================================================================================ Done

                ********** Training Completed **********
```

```
[48] print(f'Total Time Taken : {total_time} seconds')

     Total Time Taken : 204.25330543518066 seconds
```

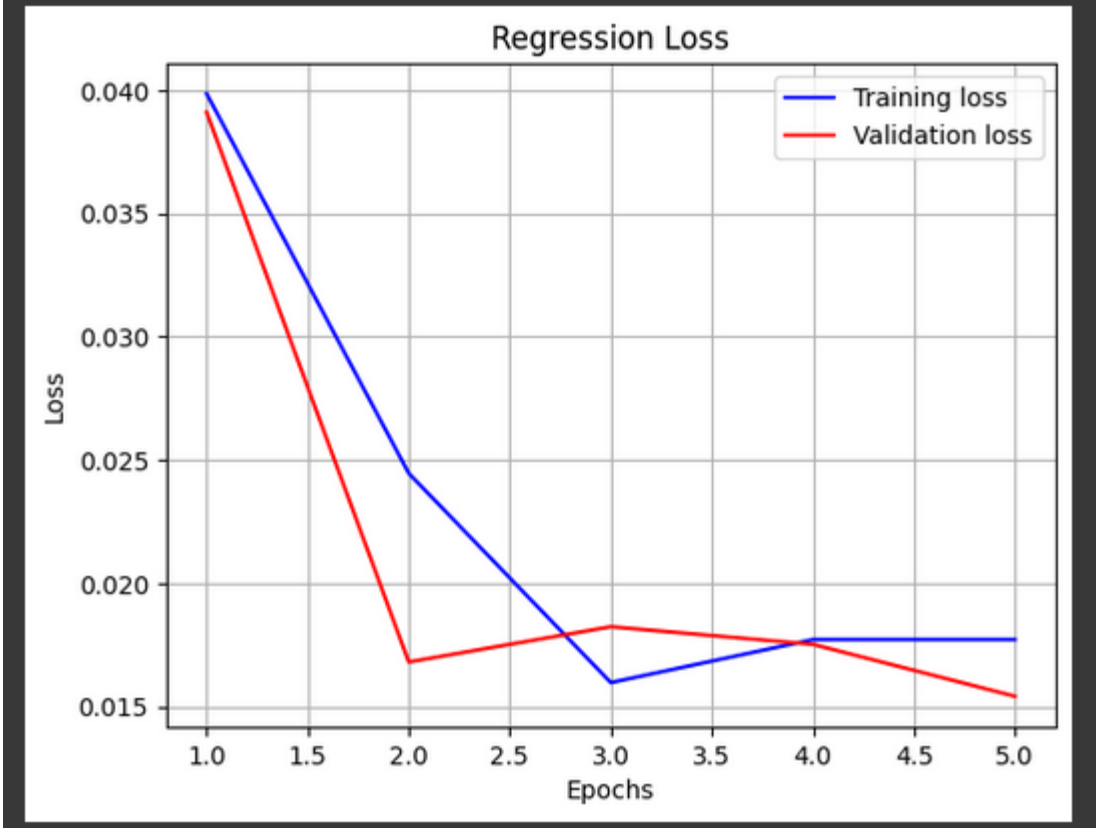
```
plot_pf(train_tl, val_tl, "Total Loss")
```



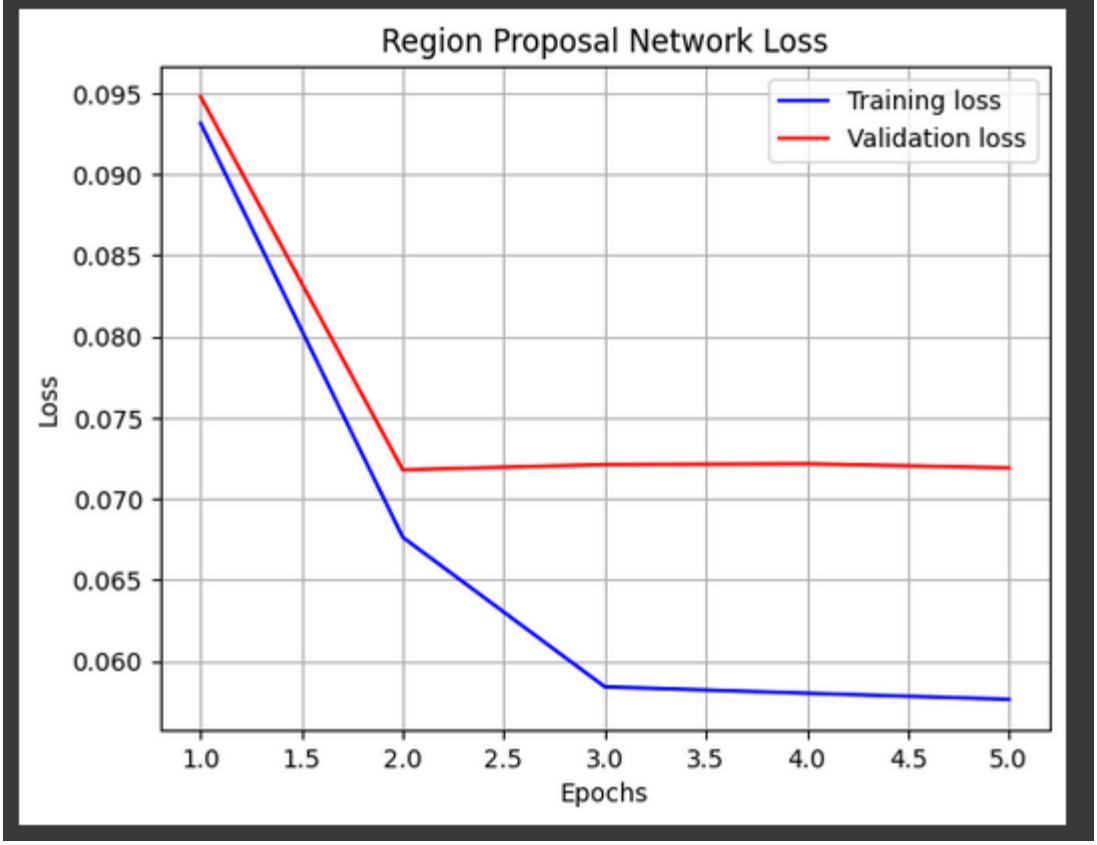
```
plot_pf(train_cl, val_cl, 'Classification loss')
```



```
plot_pf(train_rl, val_rl, "Regression Loss")
```



```
plot_pf(train_rpnl, val_rpnl, "Region Proposal Network Loss")
```



```
plot_pf(train_ol, val_ol, "Object Loss")
```