Task Scientific Writing

- Write a one-pager about radar-camera perception in autonomous vehicles
- Include the motivation behind the research
- Clarity in defining and scoping of the problem
- A brief review of related work
- An overview of the possible research directions
- There is no abstract required
- You can use the provided latex template
- Please send the PDF !

Task: Computer Vision and Machine Learning

Software requirements: Python, openCV(cv2) or PIL, numpy, PyTorch (no GPU required).

Please send the code in a ZIP file!

Dataset

The SemanticKITTI dataset is a popular benchmark for LiDAR perception on Autonomous Driving.

Together with KITTI Odometry, it conforms a dataset with LiDAR 3D semantic annotations and synchronized camera images for every scan.

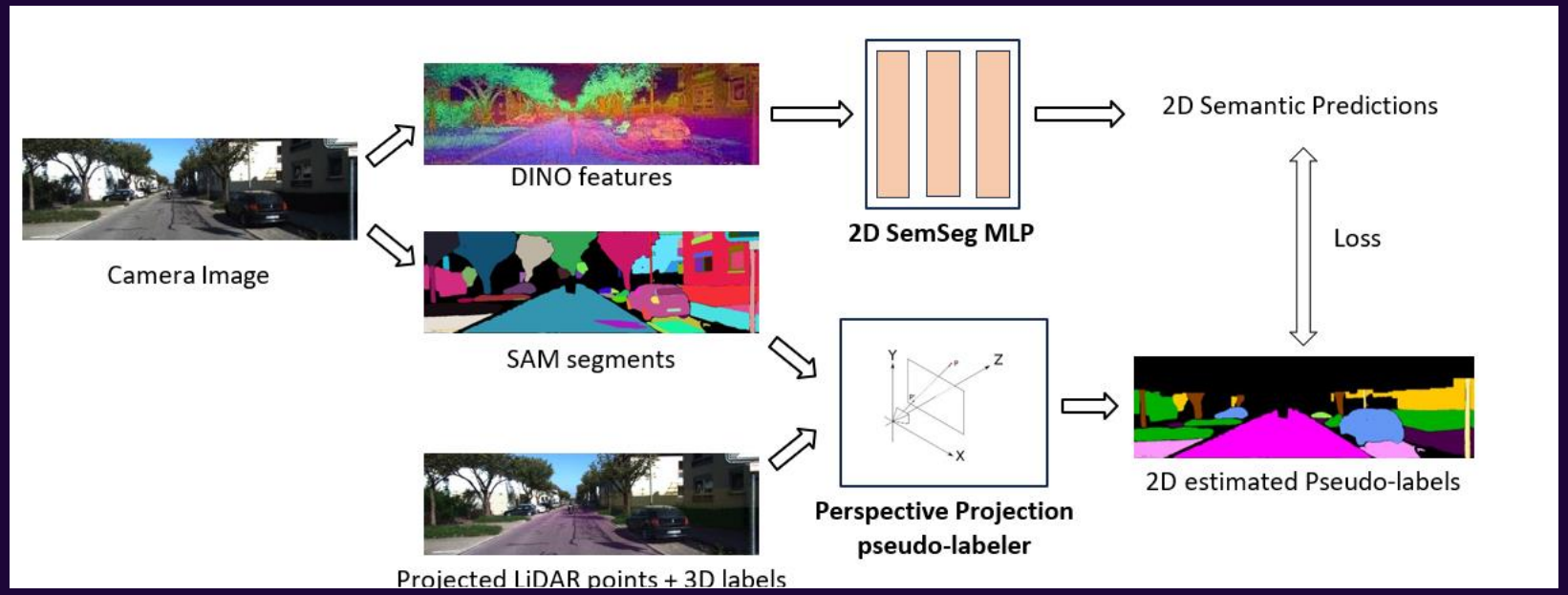
The official dataset contains 19130 training scans + images, and 6019 test scans + images distributed in 11 sequences. For this project, we only use a small subset of the dataset, with 25 scans for training and 5 for testing.

This dataset does not have official 2D Semantic labels. Instead, in this project we will automatically generate 2D Semantic labels from the 3D Semantic labels and some unlabeled generic masks from a 2D foundation model.

Goal

The goal of this task is to implement a simple multi-modal perception model for 2D Semantic Segmentation in Autonomous Driving. For each Camera Image, we provide its extracted DINO features and SAM segments. DINO and SAM are two public foundation models from *Meta AI Research* based on Vision Transformers (ViT).

We want to use the LiDAR 3D labels from SemanticKITTI and the SAM segments from the camera image, to generate 2D Semantic pseudo-labels with a **Perspective Projection pseudo-labeler.** Then, we want to train an **MLP** Neural Network that inputs the DINO features extracted from the camera image, and outputs 2D Semantic Predictions. We use the 2D pseudo-labels to supervise the MLP.



Info

For each train/test scan, we provide the following:

-***velodyne***: Contains the *xyz* coordinates of each LiDAR point and the reflectivity of each point. The binary file is read with:

```
velo = np.fromfile(<path>, dtype=np.float32).reshape(-1,4)
```

-***velodyne_labels***: Contains the semantic label ID of each LiDAR point. The binary file is read with:

```
velo_label = np.fromfile(<path>, dtype=np.uint32)
```

-***image_2***: RGB image of the front left camera.

-***K_cam2.npy***: camera intrinsics for all the frames of the sequence. $K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$

-***T_cam2_velo.npy***: camera extrinsics. Contains the rotation and translation between the Camera and LiDAR. $[R \mid t] = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix}$

-***dino_features***: extracted DINO features from the camera image. Each *.npy* file contains a float array of shape `(h/down, w/down, 384)`, where *h* and *w* are the *heights* and *widths* of the camera image, and *down* is approximately 5.4 . This means the resolution of the DINO features is ~5.4 smaller than the camera image dimensions. We additionally provide *dino_features_vis* with the 3 main components of the PCA of the features. The *dino_features_vis* is not needed for your model, it‘s only to help you understand what the DINO features mean.

-***sam_segments***: extracted SAM segments from the camera image. Each *.png* image contains the label IDs of the segment to which that pixel belongs. The ID value 0 means that the pixel does not belong to any segment and should be ignored.

NOTE: The images from distinct sequences have different *heights* and *widths*, and so do their DINO features.

Consider zero-padding to address this.

Additional metadata

```python
def create_semkitti_label_colormap():
    """Creates a label colormap used in SEMANTICKITTI segmentation benchmark.

    Returns:
        A colormap for visualizing segmentation results in BGR format.
    """
    colormap = np.zeros((256, 3), dtype=np.uint8)
    colormap[0] = [0, 0, 0]
    colormap[1] = [245, 150, 100]       # "car"
    colormap[2] = [245, 230, 100]       # "bicycle"
    colormap[3] = [150, 60, 30]         # "motorcycle"
    colormap[4] = [180, 30, 80]         # "truck"
    colormap[5] = [255, 0, 0]           # "other-vehicle"
    colormap[6] = [30, 30, 255]         # "person"
    colormap[7] = [200, 40, 255]        # "bicyclist"
    colormap[8] = [90, 30, 150]         # "motorcyclist"
    colormap[9] = [255, 0, 255]         # "road"
    colormap[10] = [255, 150, 255]      # "parking"
    colormap[11] = [75, 0, 75]          # "sidewalk"
    colormap[12] = [75, 0, 175]         # "other-ground"
    colormap[13] = [0, 200, 255]        # "building"
    colormap[14] = [50, 120, 255]       # "fence"
    colormap[15] = [0, 175, 0]          # "vegetation"
    colormap[16] = [0, 60, 135]         # "trunk"
    colormap[17] = [80, 240, 150]       # "terrain"
    colormap[18] = [150, 240, 255]      # "pole"
    colormap[19] = [0, 0, 255]          # "traffic-sign"
    return colormap
```

To visualize the labels, you can use the following function that follows the standard Cityscapes label mapping (in **BGR**):
- Code is provided in colormap.py

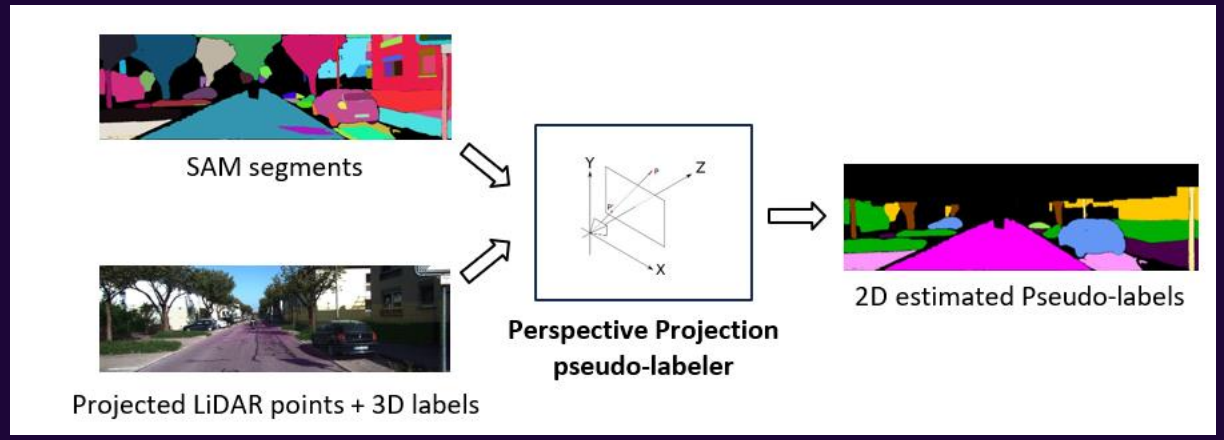
```
cmap = create_semkitti_label_colormap()
labelmap_vis = cmap[labelmap]
```

Task 1.1: Perspective Projection pseudo-labeler

Implement the **Perspective Projection pseudo-labeler** and generate the 2D pseudo-labels for each camera image.

For this, you should first write a code to project the LiDAR points into the camera plane. Then, for each of the SAM segments, you should take the label of all the LiDAR points that fall within that segment, and take the *argmax* of them. Then, all the pixels of that segment should be assigned to the estimated Semantic label (*argmax*).

It is recommended that you visualize the projected lidar points and your 2D pseudo-labels with the provided colormap
before you continue further.



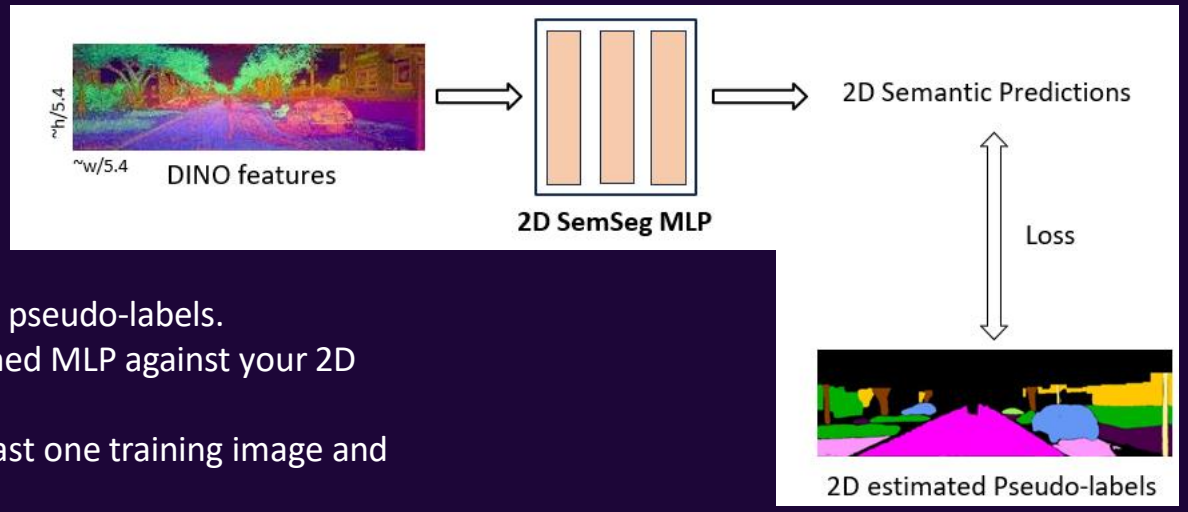
Task 1.2 : Predictions

Implement, train, and evaluate the **2D Semantic Segmentation MLP Network**.

For this task, you will use your generated 2D pseudo-labels to supervise a 2D Semantic Segmentation MLP Network. The MLP network, takes as inputs the downsampled DINO features from the image, and predicts a 2D semantic label for each pixel in the original image. You should include in your MLP network an operation to address the shape mismatch between the features and the target 2D semantic labels. Use an appropriate loss function for the task, such as Cross-Entropy loss.

Use sequences **[00, 01, 02, 03, 04, 05, 06, 07, 09, 10]** for training and sequence **08** for testing.

NOTE: use only very few small layers to implement the MLP. The goal of this task is to see your code writing style, not to obtain a high performance.



a) Report the training & testing pixel accuracy of your trained MLP against your 2D pseudo-labels.
b) Report the training & testing mean Intersection-over-Union (mIoU) of your trained MLP against your 2D pseudo-labels.
c) Provide a visualization of the Semantic predictions of your trained MLP for at least one training image and one testing image.
d) Share the scripts with the code that you used for the task alongside the visualizations that you want to share and the reported model performances in a single compressed file.