3D Perception & Learning-Based Data Fusion
Multi-Modal Semantic Segmentation

Trimigno Giuseppe
giuseppe.trimigno@studenti.unipr.it

October 2023

1 Introduction

In this project, our aim is to exploit multiple sensors, specifically *RGB*, *Depth*, *Event* cameras and *LiDAR* point clouds (projected onto 2D plane), in order to tackle a *Semantic Segmentation* task with Deep Learning models, by fusing synchronized and source-different images using traditional techniques, which doesn't involve fusion-learning.

2 Dataset

We chosen a really hard dataset, called *DeLiVER*, covering ***De**pth*, ***Li**DAR*, *multiple **V**iews*, ***E**vents*, and ***R**GB*.
In this dataset we have images coming from 4 different types of sensors, i.e. RGB, Depth, Event cameras and LiDAR (3D point clouds projected onto 2D plane).
The main difficult in dealing with this dataset are related to the following considerations:

- It contains four severe weather conditions (i.e. *cloudy*, *foggy*, *rainy* and *night*) other than *sunny*, as well as five sensor failure (*motion blur*, *over-exposure*, *under-exposure*, *LiDAR jitter* and *event low-resolution*), as represented in figure 1;
- It contains 25 different classes, which is a really high number compared to the number of samples and different situations in the dataset. Specifically, they are Building, Fence, Other, Pedestrian, Pole, RoadLine, Road, SideWalk, Vegetation, Cars, Wall, TrafficSign, Sky, Ground, Bridge, RailTrack, GroundRail, TrafficLight, Static, Dynamic, Water, Terrain, TwoWheeler, Bus and Truck;
- Although the full dataset consists of 47310 samples, only 7885 (5988 for training and 1897 for testing) of them are publicly available.

Based on these premises, we understand that it is really hard to train a model which can correctly segment every pixel between 25 classes in all those out-of-distribution scenarios, considering the low number of images that we have.

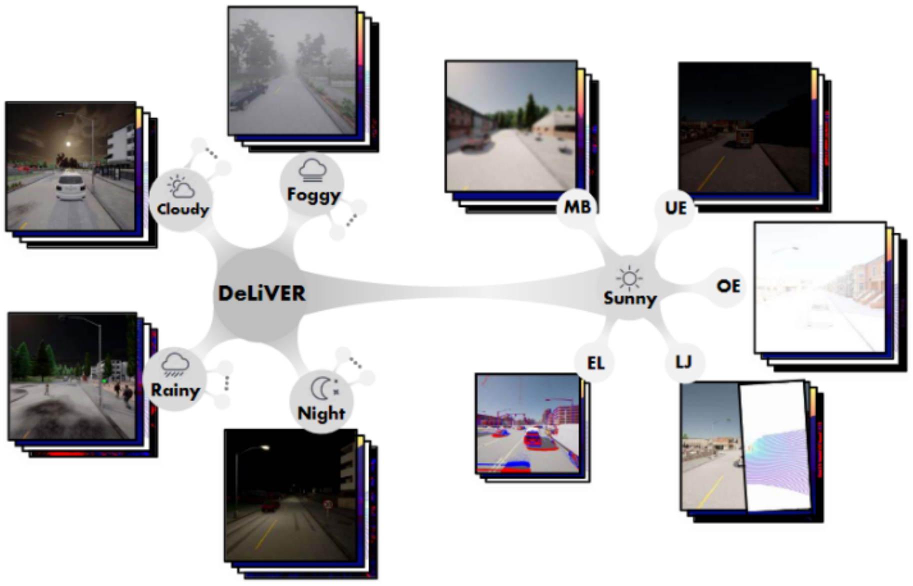

Figure 1: DeLiVER dataset composition.

3 Experiments

3.1 Models

We basically considered two models, which are:

- **FCN** (*Fully Convolutional Network*): basically a set of convolution, pooling and upsampling layers without any fully-connected layer;
- **DeepLabV3**: a more complex head which can give really good results in semantic segmentation tasks. It is based on *atrous convolutions* in cascade or in parallel to capture multi-scale context by adopting multiple atrous rates, and on *ASPPP* (*Atrous Spatial Pyramid Pooling*).

Both models were trained from scratch, using *ResNet50* and *ResNet101* as backbone, totalling 4 combinations: *ResNet50 + FCN*, *ResNet101 + FCN*, *ResNet50 + DeepLabV3* and *ResNet101 + DeepLabV3*.
Moreover, we considered to also add a simply auxiliary classifier (let's call it *aux classifier* from now on), whose architecture is shown in figure 2, in order to try helping the main model to converge better.

As for hyperparameters, we chosen for the following ones:

- **Criterion**: we chosen 2 different losses, i.e. *CrossEntropy* loss and *Boundary* loss (a modified version of *Dice* loss, which is indeed to better boundaries segmentation). Specifically, models are trained only with CrossEntropy when they don't make use of the aux classifier, while they are trained with a combination, i.e. sum, of Boundary loss (evaluated over the main classifier predicted mask) and CrossEntropy loss (evaluated over aux classifier predicted mask) when they use both classifiers;
- **Optimizer**: we chosen *AdamW* as optimizer, with a 0.01 *weight decay*;

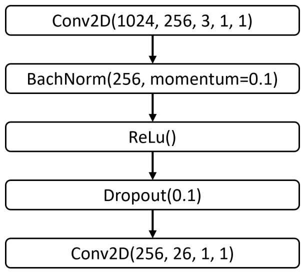

Figure 2: Auxiliary classifier's architecture

- **Learning rate**: we set learning rate to $7 * 10^{-5}$, using the Pytorch so-called *ReduceLROn-Plateau*, set up to monitor the validation mean IoU (maximization mode), in order to reduce the learning rate by a 0.2 factor (until it reaches 10^{-7}, which has been set as lower bound) if the monitored metric doesn't increase for 2 consecutive epochs.

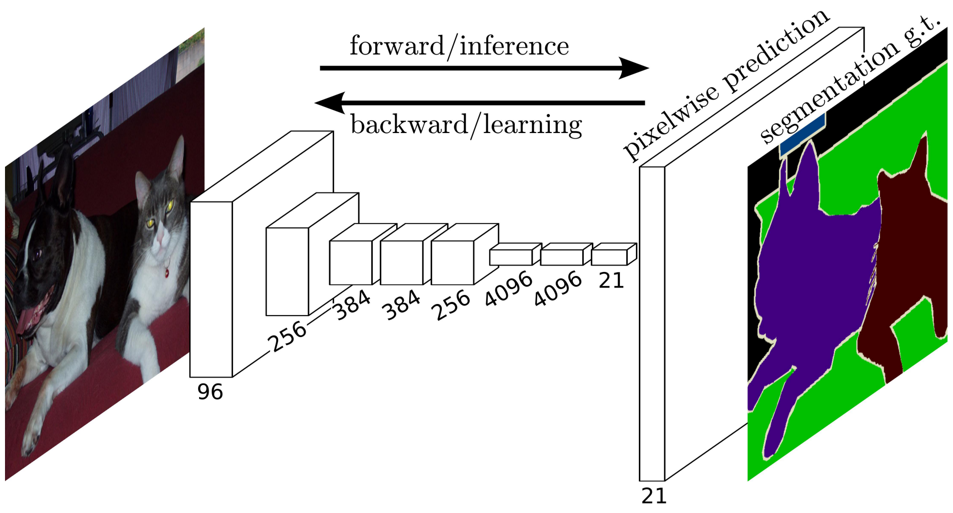

Figure 3: General architecture of FCN

3.2 Data Fusion

In novel proposed traditional data-fusion techinque, in order to fuse images coming from 4 different modalities in a single one, we try to exploit two main non-learnable dimensionality reduction techniques:

- **PCA** (*Principal Component Analysis*): is a linear dimensionality reduction algorithm, which works by linearly transforming the data into a new coordinate system where (most of) the variation in the data can be described with fewer dimensions than the initial data, i.e. it makes use of *Singular Value Decomposition* of the data;

- **Isomap**: is a non-linear dimensionality reduction algorithm, which provides a simple method for estimating the intrinsic geometry of a data manifold based on a rough estimate of each data point's neighbors on the manifold. At high level, the algorithm works in the following way: (1) Determines K nearest neighbors for each point; (2) Construct a neighborhood graph where each node is a point connected to its K nearest neighbors, with an edge weight equal to the *Euclidian* distance between the two points; (3) Compute shortest path between two points, e.g. using *Dijkstra* or *Floyd-Warshall* algorithm; (4) Compute lower dimensional embedding using *Multidimensional scaling*.

In order to make it possible, we defined a short fusion function, algorithmically defined as shown in pseudo-code 1.

Algorithm 1 Proposed data-fusion algorithm

```
Require: $decomposition \in \{\text{PCA}, \text{Isomap}\}$
Require: $features \in list[Tensor(N \times C \times H \times W)]$
Ensure: $fused = Tensor(N \times C \times H \times W)$
  assert($feature\_set$ is detached and on cpu $\forall feature\_set \in features$)
  $required\_shape \leftarrow features[0].shape$
  for $feature\_set \in features$ do
      $feature\_set \leftarrow flatten(feature\_set)$
  end for
  $features \leftarrow transpose(numpy(features))$
  $fused \leftarrow decomposition(features)$
  $fused \leftarrow flatten(tensor(fused))$
  $fused \leftarrow reshape(fused, required\_shape)$
```

At this point, we defined our data-fusion function, but we need to choose when the fusion will be done. Independently from the decision, we need to overload the forward method of model classes, because data fusion will be located there.
Specifically, three approaches were taken into consideration:

- **Sensor-level** abstraction: in this case, given four synchronized images coming from four different sensors, they are fused together before going as input to the model, i.e. fusion acts on raw sensor data.
  We can refer to algorithm 2 for pseudo-code of consequently overloaded forward method;

- **Single-backbone Feature-level** abstraction: in this case, given four synchronized images coming from four different sensors, they are sequentially given as input to the model, then we take *latent features* as output, i.e. the deepest representation of the images, learned by the model (we so take the *backbone*'s output, not the *classification head*'s one).
  In this way, fusion acts on latent features, but keep in mind that we use a *single* backbone to extract feature from all sensors. We can refer to algorithm 3 for pseudo-code of consequently overloaded forward method.

- **Multiple-backbone Feature-level** abstraction: as in previous case, but in this one we have four different backbones (and a single classification head), where each one acts as feature extractor from a single sensor modality.
  We can refer to algorithm 4 for pseudo-code of consequently overloaded forward method.

Refer to figure 4 for a graphical representation of these architectures.

Algorithm 2 Proposed overloaded forward method for sensor-level abstraction fusion

```
Require: $decomposition \in \{\text{PCA}, \text{Isomap}\}$
Require: $features \in list[Tensor(N \times C \times H \times W)]$
Ensure: $pred\_mask_1, pred\_mask_2 = logits$
  $input\_shape \leftarrow features[0].shape[-2:]$
  $fused \leftarrow fuse\_features(decomposition, features)$
  $fused \leftarrow fused.cuda().requires\_grad\_()$
  $fused \leftarrow backbone(fused)$
  $x_1 \leftarrow classifier(fused['out'])$
  $x_2 \leftarrow aux\_classifier(fused['aux'])$
  $pred\_mask_1 \leftarrow interpolate(x_1, input\_shape, bilinear)$
  $pred\_mask_2 \leftarrow interpolate(x_2, input\_shape, bilinear)$
```

Algorithm 3 Proposed overloaded forward method for single-backbone feature-level abstraction fusion

```
Require: $decomposition \in \{\text{PCA}, \text{Isomap}\}$
Require: $features \in list[Tensor(N \times C \times H \times W)]$
Ensure: $pred\_mask_1, pred\_mask_2 = logits$
  $input\_shape \leftarrow features[0].shape[-2:]$
  for $feature \in features$ do
      $feature \leftarrow backbone(feature)$
  end for
  $fused \leftarrow fuse\_features(decomposition, features)$
  $fused \leftarrow fused.cuda().requires\_grad\_()$
  $x_1 \leftarrow classifier(fused['out'])$
  $x_2 \leftarrow aux\_classifier(fused['aux'])$
  $pred\_mask_1 \leftarrow interpolate(x_1, input\_shape, bilinear)$
  $pred\_mask_2 \leftarrow interpolate(x_2, input\_shape, bilinear)$
```

Algorithm 4 Proposed overloaded forward method for multi-backbone feature-level abstraction fusion

```
Require: $decomposition \in \{\text{PCA}, \text{Isomap}\}$
Require: $features \in list[Tensor(N \times C \times H \times W)]$
Ensure: $pred\_mask_1, pred\_mask_2 = logits$
  $input\_shape \leftarrow features[0].shape[-2:]$
  for $i \in range(len(features))$ do
      $feature_i \leftarrow backbone_i(feature_i)$
  end for
  $fused \leftarrow fuse\_features(decomposition, features)$
  $fused \leftarrow fused.cuda().requires\_grad\_()$
  $x_1 \leftarrow classifier(fused['out'])$
  $x_2 \leftarrow aux\_classifier(fused['aux'])$
  $pred\_mask_1 \leftarrow interpolate(x_1, input\_shape, bilinear)$
  $pred\_mask_2 \leftarrow interpolate(x_2, input\_shape, bilinear)$
```

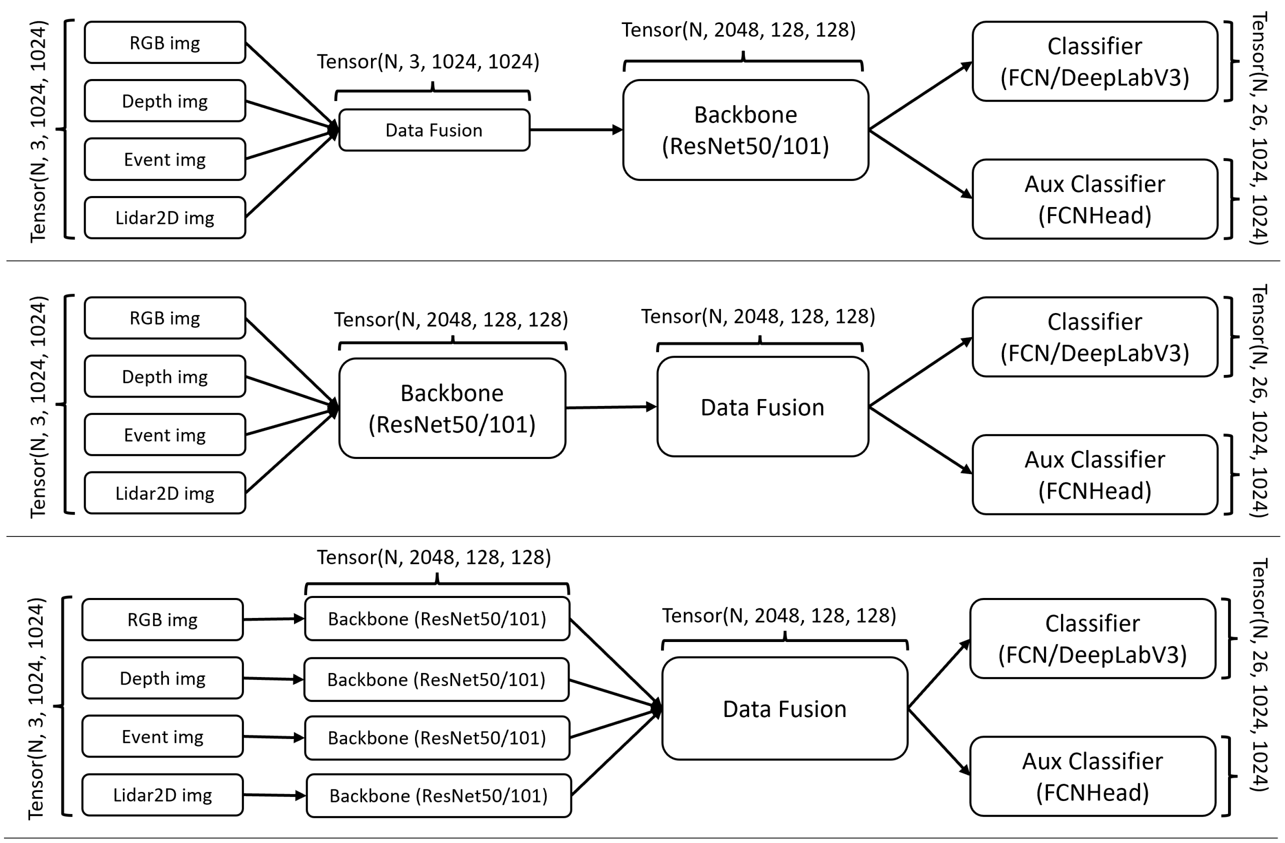

Figure 4: Graphical representation of proposed architectures: Sensor-level abstraction (Top), Single-backbone Feature-level abstraction (Middle), and Multiple-backbone Feature-level abstraction (Bottom)

4 Results

Based on considerations in two previous chapters, we could have a total of 72 architectures, which are resumed in table 1.

Backbone	**Classifier**	**Aux**	**Criterion**	**Decomposition**	**Fusion**
ResNet50/ ResNet101	FCN/ DeepLabV3	No	CrossEntropy	PCA/ Isomap	Early/ Middle-Single/ Middle-Multi
ResNet50/ ResNet101	FCN/ DeepLabV3	Yes	CrossEntropy + CrossEntropy Boundary + CrossEntropy	PCA/ Isomap	Early/ Middle-Single/ Middle-Multi

Table 1: All possible architectures that can be tried, where '/' denotes that all combinations must be taken into consideration

In our case, we tried to train:

- DeepLabV3 + ResNet101 (RGB only): we trained this architecture as a baseline model, in order to be able to make a performance comparison with data-fusion results;

- FCN + ResNet50/101 + NoAux + PCA + Middle-Single: we trained this model but we stopped after few epochs, since it wasn't able to learn (it went good in training, but performance in validation were close to 0), and we still don't know why;

- DeepLabV3 + ResNet50/101 + NoAux + PCA + Early: the combination of PCA with early fusion has been the easiest one to train, simply because it was the fastest one. We fully trained the two models in this settings;

- DeepLabV3 + ResNet50/101 + NoAux + Isomap + Early: after getting promising results from the previous experiment, we tried to train the same configurations, but with Isomap instead of PCA. Unfortunately, we weren't able to end the train due to time limits ($\approx$ 4 hours for few training iterations using 6 CPU cores).
  One possibility, which has not been explored in this project, is to pre-compute fused features with Isomap on a CPU node (which has power-full CPUs, offering more than 60 cores simultaneously, i.e. a big speed-up for Isomap), and we can do this because we are considering early fusion;

- DeepLabV3 + ResNet50/101 + NoAux + PCA + Middle-Single: we trained models in these settings, with performance similar to the second group of settings (a little higher). Although, performing data fusion on $N \times 2048 \times 128 \times 128$ tensors is computationally more expensive than performing it on $N \times 3 \times 1024 \times 1024$ tensors, so we decided to continue with second group of settings instead of this.

- DeepLabV3 + ResNet50/101 + YesAux (Boundary + CE) + PCA + Early: we explored this configuration because we thought that, using an auxiliary classifier, and changing the main classifier's loss to a Boundary loss, we could increase performance. However we obtained results that are a little lower than the one obtained without the auxiliary classifier.

Specifically, *macro* and *weighted* average results for baseline and *DeepLabV3 + ResNet50/101 + NoAux + PCA + Early* are shown in table 2, while results for each class are shown in table 3.

Model	**mIoU (macro)**	**mIoU (weighted)**	**Accuracy (macro)**	**Accuracy (weighted)**
DeepLabV3+ResNet101 NoAux RGB-Only	0.3	0.8	0.35	0.85
DeepLabV3+ResNet50 NoAux PCA (Early-Fusion)	0.328	0.811	0.391	0.871
DeepLabV3+ResNet101 NoAux PCA (Early-Fusion)	0.34	0.85	0.403	0.905

Table 2: Obtained global results for baseline and two fusion-based models.

Model	**Road**		**Sky**		**Building**		**Vegetation**		**RoadLine**		**Cars**		**SideWalk**		**Pedestrian**		**Pole**		**Truck**	
	mIoU	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**	**mIoU**	**Acc**
DeepLabV3 ResNet101 NoAux RGB-Only	0.93	0.961	0.867	0.944	0.601	0.725	0.556	0.66	0.628	0.68	0.538	0.694	0.522	0.637	0.3	0.411	0.25	0.323	0.25	0.3
DeepLabV3 ResNet50 NoAux PCA Early-Fusion	0.8997	0.9362	0.8992	0.965	0.6114	0.7297	0.4972	0.551	0.5955	0.6864	0.5424	0.616	0.5202	0.6757	0.3241	0.4152	0.2929	0.4485	0.2113	0.2754
DeepLabV3 ResNet101 NoAux PCA Early-Fusion	0.9389	0.9704	0.9259	0.9668	0.6386	0.7364	0.6159	0.7083	0.6097	0.6801	0.5688	0.7277	0.5518	0.7099	0.3434	0.4614	0.3018	0.4239	0.2431	0.2851

Table 3: Obtained single-class results for baseline and two fusion-based models. Since we have 25 classes, only 10 of them are shown.

Unfortunately, we weren't able to train any model with multiple backbones, basically for two reasons: (1) We had to concurrently train 4 backbones and a classifier, which requires a lot of time, and we didn't had it; (2) In order to correctly train multiple backbones, we should have more data available.
In general, we expect that we can further increase performance, because each backbone can learn how to extract features from that specific sensor modality, and not from all modalities, even reaching 0.5-0.6 as mean IoU if additional training data and more sophisticated fusion algorithm (e.g. Isomap) are used.
Moreover, we think that our 0.35 value of mean IoU is not a bad result compared to the 0.6 of the state-of-the-art model in this dataset. In fact: (1) it was not our best possible model; (2) it uses only 5988 training samples (which are too few in order to understand all critical conditions in the dataset, plus every 25 classes, considering that some of them are really poor represented), compared to their 36000 training samples; (3) we used a simple traditional algorithm (i.e. PCA) for fusing features, while they used most complex attention mechanisms for fusion, and transformers for segmentation.