PROJECT REPORT

Sai Kiran(Net ID: vc2118)
Gayatri Kalidindi (Net ID:gk2205)

May 2021

Title: Hierarchical Multi scale Attention for Semantic Segmentation
Github: here

Introduction and Problem Description:
In general, multiple scaled versions of the same image are used to improve semantic segmentation. The task of semantic segmentation is to label all pixels within an image as belonging to one of N classes. Some predictions(like large structures) are best handled at lower inference resolution and other tasks(like thin structures or edges of objects) are better handled at higher inference resolution. In this work (paper), these different scale images(like 1x, 0.5x, 2x) are passed and the results are combined using an attention-based method to predict how to combine multi-scale predictions, instead of traditional max/average pooling. We also use an hierarchical structure which is more efficient and takes 4x less memory to train. With the hierarchical method, instead of learning all attention masks for each of a fixed set of scales, we learn a relative attention mask between adjacent scales. When training the network, we only train with adjacent scale pairs. This hierarchical structure allows us to improve on the training efficiency as compared to other methods.

Motivation
We have found this topic interesting due to its immense potential including but not limited to use cases like Biomedical image diagnosis, Autonomous vehicles such as self-driving cars, Handwriting recognition, Portrait mode photography, Virtual Backgrounds, Virtual Make-up/try-on, etc.

Literature Survey:

1. Paper(Main) : Hierarchical Multi scale Attention for Semantic Segmentation.
   Authors: Andrew Tao - NVIDIA, Karan Sapra- NVIDIA, Bryan Catanzaro - NVIDIA
   Link : Multi scale Attention

2. Paper(Motivation) : Attention to Scale: Scale-aware Semantic Image Segmentation.
   Authors: Liang-Chieh Chen, Yi Yang, Jiang Wang, Wei Xu, Alan L. Yuille
   Link : Attention to Scale

3. Paper(Loss) : Region Mutual Information Loss for Semantic Segmentation.
   Authors: Shuai Zhao , Yang Wang , Zheng Yang , Deng Cai
   Link : Region Mutual Information Loss

4. Paper(Dataset) : The Mapillary Vistas Dataset.
   Authors : Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, Peter Kontschieder
   Link :Mapillary Vistas Dataset

5. Paper(Dataset) : The Cityscapes Dataset.
   Authors : Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld
   Link : The Cityscapes Dataset

Description of Dataset:

The paper used **Cityscapes** dataset which is a large dataset that labels 19 semantic classes across 25,000 high resolution images. This is a diverse dataset of stereo video sequences recorded in 50 different cities. The cityscapes dataset is intended for accessing the performance of computer vision algorithms for major tasks of semantic urban scene understanding.
Mapillary Vistas is a dataset containing 18,000 high resolution images annotated into 65 semantic object categories and 100 instance-specifically annotated categories. It covers 6 continents with a variety of weather, season, time of day, camera, and viewpoint. This dataset is designed to capture the broad range of outdoor scenes available around the world. Training and validation data comprise 18000 and 2000 images respectively and the remaining 5000 images from the test set. It is a larger dataset and we will be using this dataset for our evaluations. It is divided into training, validation and testing folders which each have images, instances, labels and panopic.

Description of Models:
Input: Images from Cityscapes/mapillary dataset of different scales (0.25x(for evaluation), 0.5x, 1x, 2x).
Output : A high resolution image with a label for each pixel of an image with its corresponding class.
Data augmentation: We employ gaussian blur, color augmentation, random horizontal flip and random scaling (0.5x - 2.0x) on the input images to augment the dataset of the training process. We use a crop size of 2048x1024 for Cityscapes and 1856x1024 for Mapillary. The main models which will be used

are ResNet-50 and Attention blocks.
ResNet 50 trunk with DeepLab V3+ architecture(For demo): It consists of 5 stages each with a convolution and identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. It has over 23 million trainable parameters.
HRNet-OCR (For state-of-art results): The HRNet maintains high-resolution representations through the whole process.It starts with a high-resolution convolution stream, gradually adds high-to-low resolution convolution streams one by one, and connects the multi-resolution streams in parallel. Object-contextual representation(OCR) is a weighted aggregation of all the object region representations. It has 76 million parameters.
Attention: Input processing technique for neural networks that allows the network to focus on specific aspects of a complex input, one at a time until the entire dataset is categorized.
Semantic and Attention head consists of (3x3 conv) $\rightarrow$ (BN) $\rightarrow$ (ReLU) $\rightarrow$ (3x3 conv) $\rightarrow$ (BN) $\rightarrow$ (ReLU) $\rightarrow$ (1x1 conv).
Semantic and Attention heads are the same when we use HRNet. At the end, predictions are upsampled to the target image size with bilinear upsampling.
Using hierarchical model, makes this method scale independent during inference which is a great advantage.

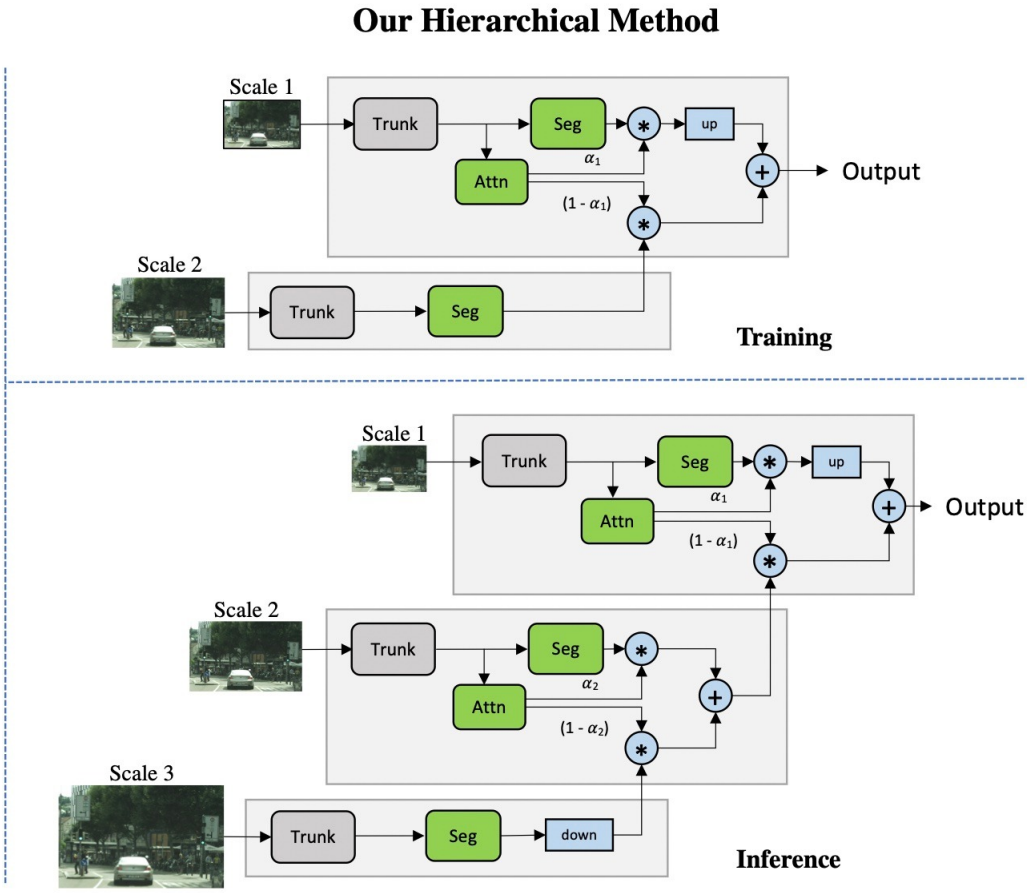


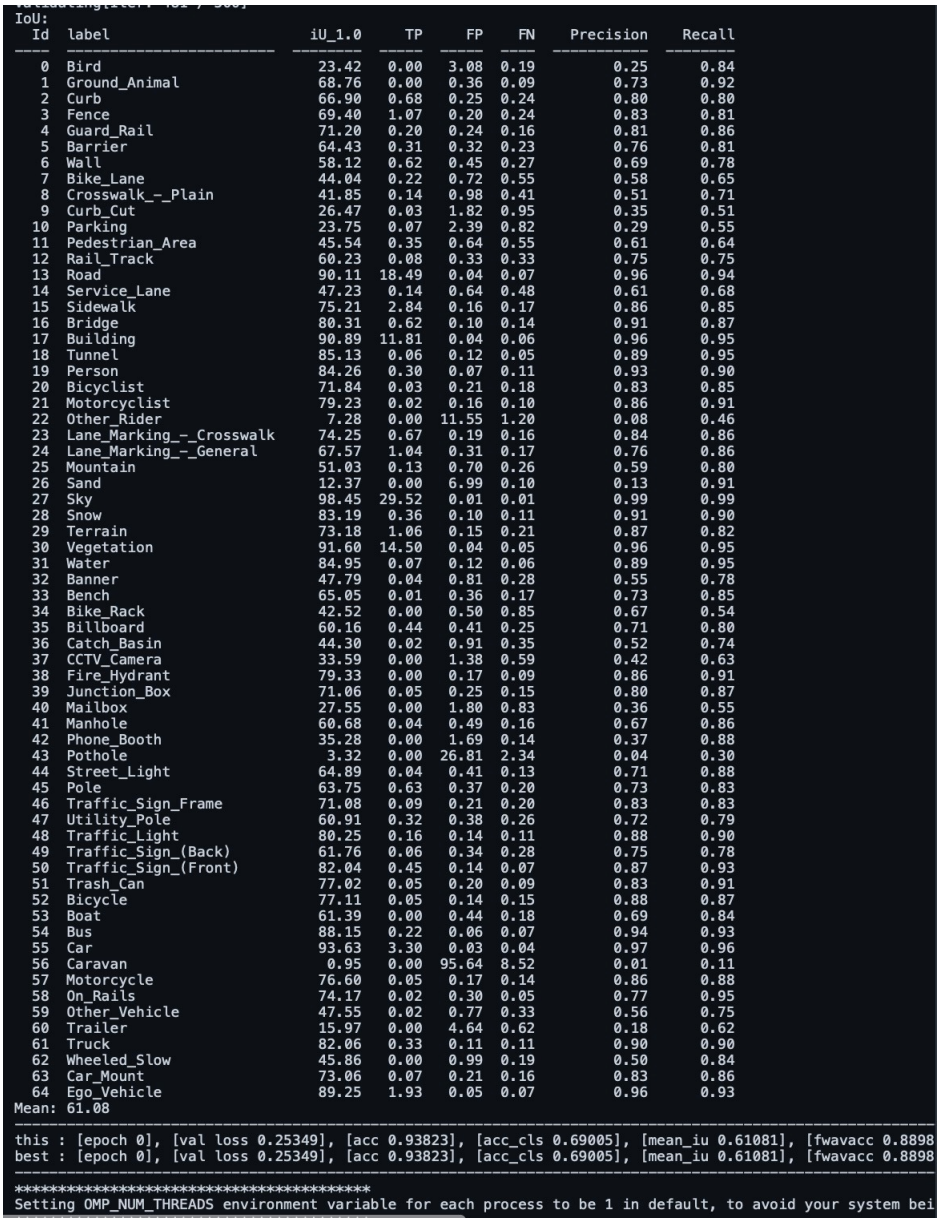
Figure1: Hierarchical Model

Description of the Loss Function:
We are using RMI as the primary Loss function(and Cross Entropy for evaluation) because RMI loss is used to model the dependencies among pixels more simply and efficiently. Pixel-wise loss treats the pixels as independent samples, whereas RMI uses one pixel and its neighbour pixels to represent this pixel. Then for each pixel in an image, we get a multi-dimensional point that encodes the relationship between pixels, and the image is cast into a multi-dimensional distribution of these high-dimensional points. The prediction and ground truth thus can achieve high order consistency through maximizing the mutual information (MI) between their multi-dimensional distributions. But the actual value of the MI is hard to calculate,so we derive a lower bound of the MI and maximize the lower bound to maximize the real value of the MI. We also used cross entropy for the auxiliary loss function. It measures the performance of a

classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Description of Training and hyperparameters:
The model is trained on NYU HPC using 2 GPU's with synchronous batch normalization, distributed data parallel training(apex) and mixed precision(fp16). Stochastic gradient descent (SGD) is used as an optimizer with a batch size of 1 per GPU, momentum 0.9 and weight decay of 5e-4 in training. We are using polynomial learning rate, as it is proved to converge faster than normal step policy, and thus can achieve 1.5 percent better performance with the same iterations. For Mapillary, we are using a poly exponent of 1, an initial learning rate of 0.02, and train for 44 epochs across 1 nodes(32 v100s or RTX 8000s) - We will be using ocrnet.HRNetMscale model network to work with. We will use 0.5,1.0,2.0 as different scales
We will also use 2 batches to train per GPU. Guassian Blur Augmentation is set and uses a crop size of 1024,1024. Rest of the parameters can be found **here**.
For validation we used 4 GPUs with distributed data parallel processing(apex), synchronous batch normalization, mixed precision(fp16), using 1 batch per GPU in evaluation mode with 2177 as the longest size for eval, with amp optimization level of 3. Scales of 0.25,0.5,1.0,2.0, with flipping enabled and use the snapshot from assets folder for pretrained model. Rest of the parameters can be found **here**.
Apex : Nvidia Apex Distributed Data Parallel, which is a module wrapper that enables easy multiprocess distributed data parallel training.
fp16 : Nvidia Apex AMP (Automatic mixed precision), generally GPUs are good at 32 bit, by switching to 16 bit, we will be using half the memory and theoretically less computation at the expense of the available number range and precision. But 16 bit training creates problems with weight updates, gradient underflow and overflow. Mixed precision training helps to avoid these problems.

Evaluation results:
Mapillary Vistas is a large dataset containing 18,000 high resolution images annotated into 66 object categories. We submitted a SBATCH job with 4 GPUs, 6 CPUs and 64GB RAM, and evaluation ran for around 2 hrs, which can be found **here**. We got an IOU score of 61.08, which is close to what the paper has achieved(61.05), which is not surprising since we used the pretrained weights. We will get output for the job we submitted in slrum file (**here**).



IoU:

Id	label	iU_1.0	TP	FP	FN	Precision	Recall
0	Bird	23.42	0.00	3.08	0.19	0.25	0.84
1	Ground_Animal	68.76	0.00	0.36	0.09	0.73	0.92
2	Curb	66.90	0.68	0.25	0.24	0.80	0.80
3	Fence	69.40	1.07	0.20	0.24	0.83	0.81
4	Guard_Rail	71.20	0.20	0.24	0.16	0.81	0.86
5	Barrier	64.43	0.31	0.32	0.23	0.76	0.81
6	Wall	58.12	0.62	0.45	0.27	0.69	0.78
7	Bike_Lane	44.04	0.22	0.72	0.55	0.58	0.65
8	Crosswalk_-_Plain	41.85	0.14	0.98	0.41	0.51	0.71
9	Curb_Cut	26.47	0.03	1.82	0.95	0.35	0.51
10	Parking	23.75	0.07	2.39	0.82	0.29	0.55
11	Pedestrian_Area	45.54	0.35	0.64	0.55	0.61	0.64
12	Rail_Track	60.23	0.08	0.33	0.33	0.75	0.75
13	Road	90.11	18.49	0.04	0.07	0.96	0.94
14	Service_Lane	47.23	0.14	0.64	0.48	0.61	0.68
15	Sidewalk	75.21	2.84	0.16	0.17	0.86	0.85
16	Bridge	80.31	0.62	0.10	0.14	0.91	0.87
17	Building	90.89	11.81	0.04	0.06	0.96	0.95
18	Tunnel	85.13	0.06	0.12	0.05	0.89	0.95
19	Person	84.26	0.30	0.07	0.11	0.93	0.90
20	Bicyclist	71.84	0.03	0.21	0.18	0.83	0.85
21	Motorcyclist	79.23	0.02	0.16	0.10	0.86	0.91
22	Other_Rider	7.28	0.00	11.55	1.20	0.08	0.46
23	Lane_Marking_-_Crosswalk	74.25	0.67	0.19	0.16	0.84	0.86
24	Lane_Marking_-_General	67.57	1.04	0.31	0.17	0.76	0.86
25	Mountain	51.03	0.13	0.70	0.26	0.59	0.80
26	Sand	12.37	0.00	6.99	0.10	0.13	0.91
27	Sky	98.45	29.52	0.01	0.01	0.99	0.99
28	Snow	83.19	0.36	0.10	0.11	0.91	0.90
29	Terrain	73.18	1.06	0.15	0.21	0.87	0.82
30	Vegetation	91.60	14.50	0.04	0.05	0.96	0.95
31	Water	84.95	0.07	0.12	0.06	0.89	0.95
32	Banner	47.79	0.04	0.81	0.28	0.55	0.78
33	Bench	65.05	0.01	0.36	0.17	0.73	0.85
34	Bike_Rack	42.52	0.00	0.50	0.85	0.67	0.54
35	Billboard	60.16	0.44	0.41	0.25	0.71	0.80
36	Catch_Basin	44.30	0.02	0.91	0.35	0.52	0.74
37	CCTV_Camera	33.59	0.00	1.38	0.59	0.42	0.63
38	Fire_Hydrant	79.33	0.00	0.17	0.09	0.86	0.91
39	Junction_Box	71.06	0.05	0.25	0.15	0.80	0.87
40	Mailbox	27.55	0.00	1.80	0.83	0.36	0.55
41	Manhole	60.68	0.04	0.49	0.16	0.67	0.86
42	Phone_Booth	35.28	0.00	1.69	0.14	0.37	0.88
43	Pothole	3.32	0.00	26.81	2.34	0.04	0.30
44	Street_Light	64.89	0.04	0.41	0.13	0.71	0.88
45	Pole	63.75	0.63	0.37	0.20	0.73	0.83
46	Traffic_Sign_Frame	71.08	0.09	0.21	0.20	0.83	0.83
47	Utility_Pole	60.91	0.32	0.38	0.26	0.72	0.79
48	Traffic_Light	80.25	0.16	0.14	0.11	0.88	0.90
49	Traffic_Sign_(Back)	61.76	0.06	0.34	0.28	0.75	0.78
50	Traffic_Sign_(Front)	82.04	0.45	0.14	0.07	0.87	0.93
51	Trash_Can	77.02	0.05	0.20	0.09	0.83	0.91
52	Bicycle	77.11	0.05	0.14	0.15	0.88	0.87
53	Boat	61.39	0.00	0.44	0.18	0.69	0.84
54	Bus	88.15	0.22	0.06	0.07	0.94	0.93
55	Car	93.63	3.30	0.03	0.04	0.97	0.96
56	Caravan	0.95	0.00	95.64	8.52	0.01	0.11
57	Motorcycle	76.60	0.05	0.17	0.14	0.86	0.88
58	On_Rails	74.17	0.02	0.30	0.05	0.77	0.95
59	Other_Vehicle	47.55	0.02	0.77	0.33	0.56	0.75
60	Trailer	15.97	0.00	4.64	0.62	0.18	0.62
61	Truck	82.06	0.33	0.11	0.11	0.90	0.90
62	Wheeled_Slow	45.86	0.00	0.99	0.19	0.50	0.84
63	Car_Mount	73.06	0.07	0.21	0.16	0.83	0.86
64	Ego_Vehicle	89.25	1.93	0.05	0.07	0.96	0.93

Mean: 61.08

this : [epoch 0], [val loss 0.25349], [acc 0.93823], [acc_cls 0.69005], [mean_iu 0.61081], [fwavacc 0.8898

best : [epoch 0], [val loss 0.25349], [acc 0.93823], [acc_cls 0.69005], [mean_iu 0.61081], [fwavacc 0.8898

****

Setting OMP_NUM_THREADS environment variable for each process to be 1 in default, to avoid your system bei

Figure2: Evaluation Output(Image trimmed for Conciseness)

Training results:

For training, we experimented with 2 different scenarios. Initially we tried to train using 4 GPUs, 256GB RAM, 64 Cores CPU on the whole dataset which

is of 18,000 training images, 7000 Validation and 2000 for testing, we got an CUDA out of memory error, which you can see in **this** slrum output. Thereafter we have iteratively reduced the dataset simultaneously to train and see to get acceptable results. At the end, we found out that using 30 percent of data 5479 (down from 18,000) images for training, and 604 validation images(down from 2000) worked okay for us. We trained it by submitting a job of 2 GPUs, 64GB RAM, 6 Core CPU for 60 hrs. This resulted in an mean IOU of 46.04(found **here**, at the end at 44 epoch- view raw since its a very big file), which rose from 20 in 44 epochs. Considering the fact that we only used 30 percent of the data set and only trained for 44 epoch, where in paper they trained for 175 epochs, we can be certain that we could have achieved the training accuracy of 85 if we would have trained in whole dataset for 175 epochs. One of the result images can be seen below. The sbatch jobs initially took lot of time to be executed from the stack, so we had to use 2 GPUs instead of 4, which ultimately saved us 2 days of trail and error time.

General Setup/Process:

At first we need to setup pytorch environment using Singularity in HPC and, create an folder called "smalldata" which consists of the downloaded dataset, pretrained and training models along with their weights, and an "uniform centroid" folder where in centroids of all the training images are extracted.
The process for training, which is similar in evaluation followed the following way. Initially, we had to download the dataset which was approximately 30GB into our /scratch/ directory, since the normal /home/ directory of HPC lacked the required inodes. After that, we had to run "python inference/training" in which all the files will be copied into "logs" folder and, a '.sh' file is created with the required hyper parameters as mentioned in yml file. Now, we have to create an sbatch job to run the script. Now, once the job is started, the output starts accumulating in "slrum-jobid.out" files.

IoU:

Id	label	iU_1.0	TP	FP	FN	Precision	Recall
0	Bird	7.18	0.00	11.58	1.35	0.08	0.43
1	Ground_Animal	2.35	0.00	37.84	3.65	0.03	0.22
2	Curb	62.02	0.67	0.30	0.32	0.77	0.76
3	Fence	59.53	0.80	0.39	0.29	0.72	0.78
4	Guard_Rail	62.35	0.14	0.25	0.35	0.80	0.74
5	Barrier	59.62	0.25	0.53	0.15	0.65	0.87
6	Wall	41.18	0.66	0.25	1.18	0.80	0.46
7	Bike_Lane	22.89	0.20	0.75	2.62	0.57	0.28
8	Crosswalk_-_Plain	28.81	0.09	1.36	1.11	0.42	0.47
9	Curb_Cut	18.16	0.02	3.17	1.34	0.24	0.43
10	Parking	23.67	0.07	1.40	1.82	0.42	0.35
11	Pedestrian_Area	52.15	0.74	0.22	0.69	0.82	0.59
12	Rail_Track	52.19	0.05	0.44	0.47	0.69	0.68
13	Road	86.52	17.44	0.08	0.07	0.92	0.93
14	Service_Lane	29.39	0.06	1.61	0.80	0.38	0.56
15	Sidewalk	65.82	2.77	0.31	0.21	0.76	0.83
16	Bridge	69.26	0.40	0.14	0.30	0.88	0.77
17	Building	87.83	12.01	0.08	0.06	0.93	0.94
18	Tunnel	8.84	0.02	10.07	0.23	0.09	0.81
19	Person	78.24	0.37	0.11	0.17	0.90	0.86
20	Bicyclist	44.45	0.02	1.05	0.20	0.49	0.83
21	Motorcyclist	72.29	0.04	0.16	0.22	0.86	0.82
22	Other_Rider	0.00	0.00	inf	inf	0.00	0.00
23	Lane_Marking_-_Crosswalk	63.46	0.63	0.26	0.32	0.79	0.76
24	Lane_Marking_-_General	60.47	0.88	0.43	0.22	0.70	0.82
25	Mountain	54.10	0.12	0.69	0.16	0.59	0.86
26	Sand	0.00	0.00	inf	inf	0.00	0.00
27	Sky	98.19	29.65	0.01	0.01	0.99	0.99
28	Snow	66.04	0.50	0.22	0.30	0.82	0.77
29	Terrain	68.08	1.00	0.22	0.25	0.82	0.80
30	Vegetation	88.54	13.67	0.07	0.06	0.94	0.94
31	Water	63.09	0.02	0.39	0.19	0.72	0.84
32	Banner	20.71	0.02	2.66	1.17	0.27	0.46
33	Bench	39.06	0.01	0.22	1.34	0.82	0.43
34	Bike_Rack	8.93	0.00	4.44	5.76	0.18	0.15
35	Billboard	51.63	0.44	0.51	0.43	0.66	0.70
36	Catch_Basin	27.17	0.01	1.93	0.76	0.34	0.57
37	CCTV_Camera	20.13	0.00	2.73	1.24	0.27	0.45
38	Fire_Hydrant	59.11	0.00	0.21	0.48	0.82	0.68
39	Junction_Box	41.77	0.03	0.82	0.57	0.55	0.64
40	Mailbox	8.36	0.00	9.17	1.79	0.10	0.36
41	Manhole	49.41	0.03	0.67	0.36	0.60	0.74
42	Phone_Booth	3.05	0.00	4.31	27.44	0.19	0.04
43	Pothole	0.33	0.00	304.04	1.78	0.00	0.36
44	Street_Light	54.46	0.04	0.67	0.17	0.60	0.86
45	Pole	51.74	0.52	0.59	0.34	0.63	0.74
46	Traffic_Sign_Frame	60.22	0.07	0.39	0.27	0.72	0.79
47	Utility_Pole	47.72	0.26	0.62	0.48	0.62	0.68
48	Traffic_Light	73.35	0.15	0.16	0.21	0.86	0.83
49	Traffic_Sign_(Back)	51.97	0.05	0.56	0.37	0.64	0.73
50	Traffic_Sign_(Front)	76.88	0.42	0.17	0.13	0.85	0.89
51	Trash_Can	55.40	0.05	0.31	0.49	0.76	0.67
52	Bicycle	68.23	0.04	0.18	0.29	0.85	0.78
53	Boat	2.26	0.00	0.86	42.32	0.54	0.02
54	Bus	83.33	0.26	0.10	0.10	0.91	0.91
55	Car	92.60	3.22	0.05	0.03	0.95	0.97
56	Caravan	0.00	0.00	inf	inf	0.00	0.00
57	Motorcycle	62.72	0.05	0.39	0.20	0.72	0.83
58	On_Rails	10.20	0.00	8.02	0.78	0.11	0.56
59	Other_Vehicle	20.46	0.01	1.14	2.75	0.47	0.27
60	Trailer	0.04	0.00	2304.75	59.93	0.00	0.02
61	Truck	72.74	0.27	0.14	0.24	0.88	0.81
62	Wheeled_Slow	31.07	0.00	1.08	1.13	0.48	0.47
63	Car_Mount	64.36	0.08	0.46	0.09	0.68	0.91
64	Ego_Vehicle	86.34	1.94	0.05	0.11	0.95	0.90

Mean: 46.04

---

this : [epoch 44], [val loss 0.35988], [acc 0.91293], [acc_cls 0.57916], [mean_iu 0.46038], [fwavacc 0.85540]
best : [epoch 44], [val loss 0.35988], [acc 0.91293], [acc_cls 0.57916], [mean_iu 0.46038], [fwavacc 0.85540]

Figure3: Training Output(Image trimmed for Conciseness)

Figure4: Training Output for an instance image

Outcomes and Challenges:

The main challenge we faced was in the resource allocation wait time once a job is submitted in HPC. We had to wait 4hrs, and 6hrs the 2 times we have requested 4 GPUs, and we were lucky to figure out the problems we had in the second attempt itself. To overcome this, we tried to run everything in one GPU unless we get an "CUDA out of memory error", but after that every error we

face, will take a lot of time to fix. We tried to trim down the Cityscapes dataset and, evaluate as we initially planned, but upon numerous trail runs we figured that the dataset is correlated and is not easy to trim down, so we went with mapillary. There is still a lot of learning curve in HPC Singularity, for which we will be facing a lot of issues in the future, but thankfully hpc@nyu.edu is quick to respond. .

Conclusion:

Our aim from first was to try to implement the paper, and little to extend it, and we have almost achieved it. For the evaluation part, we were able to perfectly emulate the paper results, but for training we had to reduce the dataset size to 30 percent of original and, we were able to train for 44 epochs instead of 175 as they were produced in the paper. We had run the training job for 60hrs only and later realised that it was able to only train the model for 44 epoch, so after the 45th epoch the job has timed out, and we felt like running the job again with 4 GPUs for 120hrs doesn't make sense, since the model has already showed its intention to increase from 20 to 46 in 44 epochs. Even though our aim was not to extend the paper, it hurts that we were not able to extend it even if its an NVIDIA paper. Ultimately its all about learning, and we had gained a lot from this project experience, from understanding state-of-thee-art paper expectation, standard to using HPCs.