Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics [8]

Raghav Mehta
McGill University
raghav.mehta@mail.mcgill.ca

Abstract

Recently, Deep Learning has gained a high amount of attention Computer Vision community. Deep Learning based methods achieve state-of-the-art results in many problems like classification, segmentation, object detection etc. Many recent papers show that these applications benefits from Multi-Task learning, although at an expense of tuning one more hyper-parameter, i.e. relative weights of these tasks, as the final performance highly correlates with these task specific weights. In a recent paper by Kendall et al. [8], authors show that these task specific weights can be learned directly by an optimization. They correlate this weight learning process in a multiple task learning problem with the homoscedastic uncertainty of each task, and show that the weights learned using this methods gives better performance than manually-tuned task weights. They experiment on CityScape Dataset for tasks of per-pixel depth regression, semantic and instance segmentation from a monocular input image. In this project, we will try to re-produce some results of this paper.

1. Introduction

Multi-task learning for different, but somehow related, tasks can improves performance on these task. We can view this multi-task learning as a regularization process, where each task acts as a regularization for other tasks. This allows the networks to learn shared representation which is common to all tasks, which in-turn helps it to generalize well. Multi-task learning has been shown to improve performance in many different machine learning areas like computer vision, natural language processing, speech recognition etc.

In this project, we focus on a recent paper by Kendall et al. [8]. In that paper, authors explore Multi-Task learning in the setting of visual scene understanding. Specifically, they focus on three tasks from a monocular camera image: semantic segmentation, instance segmentation, and depth regression. In semantic segmentation, each pixel of an image is classified into pre-defined object classes, for example, car, human, sky etc. In Instance Segmentation, all pixel are classified into instances of different objects. This task is more difficult than semantic segmentation, as the system needs to differentiate between human1 and human2. In depth regression, the system tries to estimate depth of each pixel in comparison to camera. If the system learns all these task in a multi-task setting than these tasks can benefit from each other. For example, it is easier to separate human1 from human2 after "human" class is segmented from other classes with semantic segmentation. Similarly, depth regression can also benefit from semantic segmentation. Overview of the proposed method in [8] can be seen in Fig:1

Prior approaches to [8] for multi-task learning used a naive weighted sum of losses, where loss weights are either kept same for all tasks, or they are manually tuned. Manual tuning of these weights can be time and resource consuming. Relative weights of these tasks can play a really important part in the final performance, as shown in [8]. In this case, it is desirable if there is a method by which these relative weights can be learned. In [8], authors proposed a principled way to learn these task specific weights. They showed that this multi-task weight learning can improve performance compare to manual tuning of weights. We will look into details of the proposed method in the next section.

2. Method

Deep Learning has become omni-present in almost all of the applied machined learning fields due to recent advances. For example, in the field of Computer Vision, Deep Learning tools achieve state-of-the-art results in the varied range of problems like classification, segmentation, object detection, depth regression, etc. Though, these results are promising and gives better performance, they don't provide us the associated uncertainty with the output. This is due to the fact that Deep Learning gives point-wise determinis-

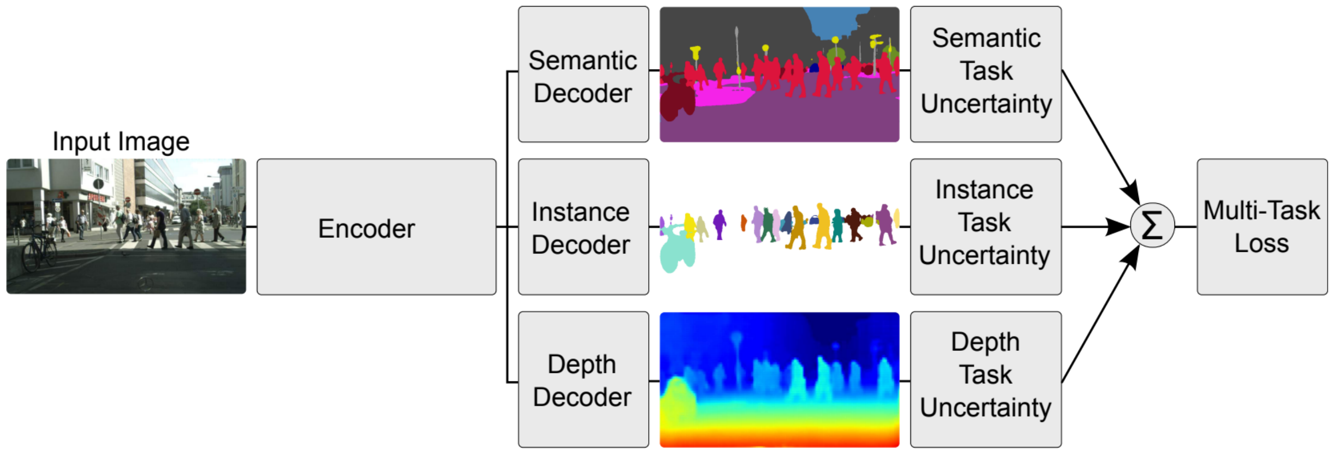


Figure 1. Overview of the proposed method in [8]. Here, for three different task (semantic segmentation, instance segmentation, and depth regression) are learnt in a multi-task setting. Image Courtesy: [8]

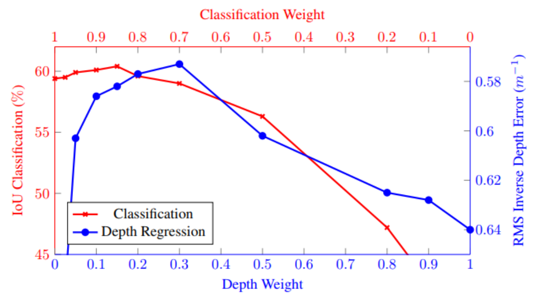
tic prediction for a single input instead of the distribution of prediction. In many crucial applications like Medical Image Processing, self-driving cars, etc., it is of paramount importance to get associated uncertainty with the output. This is due to the fact that a wrong decision in the above mentioned field can be a difference between life-and-death. Take for example a case reported in May 2016 [1], where a fatality was reported as a self-driving car of Telsa was not able to successfully differentiate between sky and a white side of trailer. Similarly, in case of automatic medical diagnosis, we don't want a patient to be identified as a healthy patient when the automatic Diagnosis system is not confident in its prediction. Instead of that it is preferable if such a case is referred to a senior doctor for better diagnosis.

In a seminal paper by Gal and Ghahramani [4], they close the gap between Bayesian methods and Deep Learning based methods. They mathematically prove that the use of dropout, a widely popular regularization technique, after every weight layer a in deep network can be interpreted as a Bayesian approximation of deep Gaussian Process (GP) model. Specifically, they shows that the dropout minimizes the KL divergence between an approximate distribution and the posterior of a deep Gaussian Process. At test time to get an associated uncertainty with the output, same input is passed through the network with dropout, N times which results in N Monte-Carlo (MC) samples of the output. Now, uncertainty is calculated as a variance of this MC samples.

Although, MC-Dropout based uncertainty estimate allows us to model uncertainty it doesn't say anything about the data uncertainty. In the following paper [7], authors divide the uncertainty in mainly two parts:

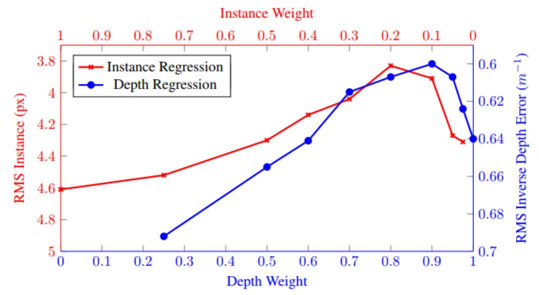
- Epistemic Uncertainty: which captures the uncertainty associated with the model parameters. This uncertainty arises as we our model parameters are not able to capture the true distribution of the model which generated our data. This uncertainty can become zero if we have infinite amount of data as it will allow our model parameter to learn the true distribution of data generation model. MC-Dropout based uncertainty estimate captures Epistemic uncertainty. This uncertainty is useful for capturing out-of-data examples. For example, if we train our model on MRI scans from Phillip and Siemens scanners but during test time we use MRI scans from GE scanners then Epistemic uncertainty for MRI scan from GE scanners will be higher than Phillips or Siemens scanners.

- Aleatoric Uncertainty: which captures the uncertainty associated with the input data. This is further divided it into two parts: Homoscedastic and Heteroscedastic uncertainty. Homoscedastic uncertainty remains constant with the input but instead it is dependent on the task at hand, while Heteroscedastic uncertainty changes with change in input. Take for example the task of MRI segmentation. In this case if we use MRI scan of same patient with different noise than the Heteroscedastic uncertainty will change, but Homoscedastic uncertainty will remain constant. But if we change the task from segmentation to regression than for the same MRI scan of patient with same noise will results in changes of Homoscedastic uncertainty, but Heteroscedastic uncertainty will remain constant.



Task Weights		Class	Depth
Class	Depth	IoU [%]	Err. [px]
1.0	0.0	59.4	-
0.975	0.025	59.5	0.664
0.95	0.05	59.9	0.603
0.9	0.1	60.1	0.586
0.85	0.15	60.4	0.582
0.8	0.2	59.6	0.577
0.7	0.3	59.0	0.573
0.5	0.5	56.3	0.602
0.2	0.8	47.2	0.625
0.1	0.9	42.7	0.628
0.0	1.0	-	0.640
Learned weights with task uncertainty (this work, Section 3.2)		**62.7**	**0.533**

(a) Comparing loss weightings when learning **semantic classification and depth regression**



Task Weights		Instance	Depth
Instance	Depth	Err. [px]	Err. [px]
1.0	0.0	4.61	
0.75	0.25	4.52	0.692
0.5	0.5	4.30	0.655
0.4	0.6	4.14	0.641
0.3	0.7	4.04	0.615
0.2	0.8	3.83	0.607
0.1	0.9	3.91	0.600
0.05	0.95	4.27	0.607
0.025	0.975	4.31	0.624
0.0	1.0		0.640
Learned weights with task uncertainty (this work, Section 3.2)		**3.54**	**0.539**

(b) Comparing loss weightings when learning **instance regression and depth regression**

Figure 2. Effect of changing relative weights of tasks in a multi-task learning system. Image Courtesy: [8]

2.1. Multi-Task Learning with Homoscedastic Uncertainty

In a multi-task setting, the total loss which a system tries to optimize is given as:

$$L_{total} = \sum_i w_i L_i$$

As mentioned before, generally these task specific weights w_i are manually tuned. These weights play an important part in deciding final performance of the system. An example of this is given in [8]. In Fig:2, effect of changing relative weights in a multi-task setting in two different scenario: semantic segmentation - depth regression and instance segmentation - depth regression is given. Performance for semantic segmentation is measured by calculating mean Intersection over Union (IoU) across classes, while for instance segmentation and depth regression it is measured as mean absolute error in a pixel space.

From Fig:2, we can see that when relative weights for different task changes, the performance for the task changes. The effect of weight on final performance is not linear. We can also observe that the weight for a particular task also depends on other task associated with it in the multi-task setting. Take for an example task of depth regression. we can observe that when system is performing multi-task learning for two tasks of semantic segmentation and depth regression, when weight for depth is kept 0.3 it gives best performance, while when the task is of Instance segmentation and depth regression, depth weight $= 0.9$ gives best performance. This shows that keeping relative weights for each task to optimal values is of critical importance, but finding this optimal weight can be expensive, which can grow exponentially with increase in number of tasks. We can also observe that multi-task learning helps for each task, as we can see improvement in performance compare to when weights are 1.

In the next, we will look at how these relative weights can be learnt as homoscedastic uncertainty.

2.2. Multi-task Likelihood

Let $f^W(x)$ be an output of a neural network with weights W and input x. We will follow probabilistic model and assume that true output value y is corrupted by a gaussian noise with standard deviation σ. In this case, likelihood of our model for regression task is defined as:

$$\begin{aligned} p(y|f^W(x)) &= \mathcal{N}(y; f^W(x), \sigma^2) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y - f^W(x))^2}{2\sigma^2}) \end{aligned} \tag{1}$$

In maximum likelihood inference, we maximize the log likelihood of model.

$$\therefore \log p(y|f^W(x)) \propto -\frac{1}{2\sigma^2}||y - f^W(x)||^2 - \log \sigma^2$$

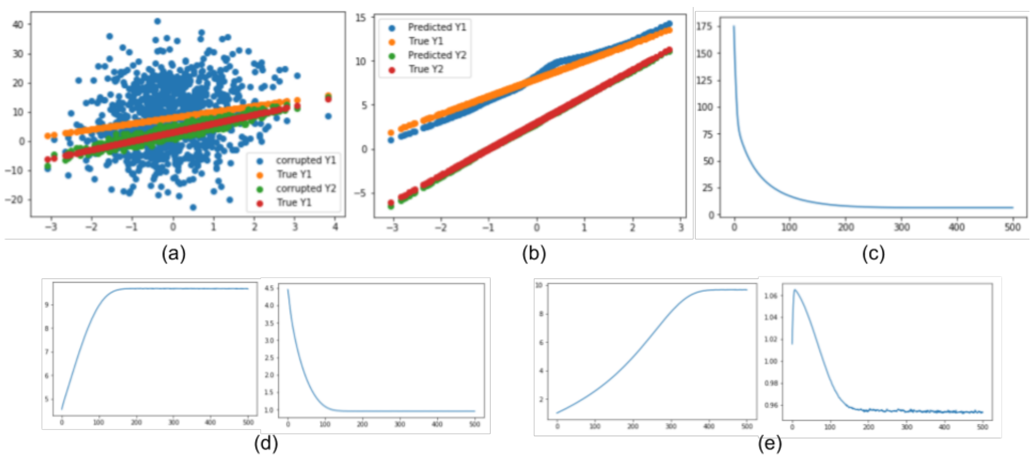


Figure 3. Example of Homoscedastic Uncertainty as task-dependent uncertainty. (a) Output y_1 and y_2 are corrupted with $\sigma_1 = 10$ and $\sigma_2 = 1$. (b) prediction by a single hideen layer neural network which minimizes the loss defined in Eq:(3). (c) Loss history as a function of epoch. Progression of predicted σ_1 and σ_2 with epoch when they are initialized as (d) 1 and (e) 5.

Now, assuming that our model produces two different outputs y_1 and y_2, and these outputs are corrupted with gaussian noise with standard deviations σ_1 and σ_2.

$$\begin{aligned} p(y_1, y_2|f^W(x)) &= p(y_1|f^W(x)) \cdot p(y_2|f^W(x)) \\ &= \mathcal{N}(y_1; f^W(x), \sigma_1^2) \cdot \mathcal{N}(y_2; f^W(x), \sigma_2^2) \end{aligned} \tag{2}$$

In the above equation, we assumed that weight of neural network w and input to the neural network x are sufficient condition to consider y_1 and y_2 to be independent.

Now, in maximum likelihood inference we minimize negative log-likelihood. Therefore, total loss of the system Ł(W, σ_1, σ_2) can be calculated as follows:

$$\begin{aligned} Ł(W, \sigma_1, \sigma_2) &= -\log p(y_1, y_2|f^W(x)) \\ &= -\log p(y_1|f^W(x)) - \log p(y_2|f^W(x)) \\ &\propto \frac{1}{2\sigma_1^2}||y_1 - f^W(x)||^2 + \log \sigma_1^2 + \\ &\quad \frac{1}{2\sigma_2^2}||y_2 - f^W(x)||^2 + \log \sigma_2^2 \\ &= \frac{1}{2\sigma_1^2}Ł_1(W) + \frac{1}{2\sigma_2^2}Ł_2(W) + \log \sigma_1^2 + \log \sigma_2^2 \end{aligned} \tag{3}$$

We can view the above equation as a weighted summation of Ł$_1(W)$ and Ł$_2(W)$, where weights are $\frac{1}{2\sigma_1^2}$ and $\frac{1}{2\sigma_2^2}$. With $\log \sigma_1^2$ and $\log \sigma_2^2$ acting as a regularization on weights so that they don't become arbitrary large or small.

For classification, we pass the output of neural network through SoftMax function. In case of output being corrupted by gaussian distribution with zero mean and σ standard deviation, we can view likelihood as passing scaled version of network output through SoftMax, where scale is defined by $\frac{1}{\sigma^2}$. Therefore, likelihood of the model:

$$p(y|f^W(x), \sigma) = \text{SoftMax}(\frac{1}{\sigma^2} f^W(x))$$

Above equation can be interpreted as a Gibbs distribution where the input is scaled by σ^2. We can learn this scale. This relate to its uncertainty, as measured in entropy. The log-likelihood for this output can be written as

$$\begin{aligned} \log p(y = c|f^W(x), \sigma) &= \frac{1}{\sigma^2} f_c^W(x) \\ &\quad - \log \sum_{c'} \exp(\frac{1}{\sigma^2} f_{c'}^W(x)) \end{aligned} \tag{4}$$

Similar to the case for regression output, we can view this as a weighted loss for classification task.

2.3. Example of Homoscedastic Uncertainty in case of multiple regression outputs

Let us consider a simple problem where the outputs y_1 and y_2 are related to input x in a following manner: $y_1 = 2x + 8$ and $y_2 = 3x + 3$. Let us consider a case where y_1 and y_2 are corrupted with additive gaussian noise with zero mean and σ_1 and σ_2 standard deviations. Note that the value of σ is not dependent on the input but is dependent on the task (i.e. σ_1 or σ_2 for y_1 or y_2). In this example, we will take $\sigma_1 = 10$ and $\sigma_2 = 1$. Now, we train a 2 layer neural network on 1000 i.i.d. instances of x for 500 epochs. We minimize the multi-task loss function defined in Eq:(3). Note that this equation allows us to estimate σ_1 and σ_2 (uncertainties) associated with y_1 and y_2. True Data distribution of y_1 and y_2 and predicted distribution for the same can be seen in Fig:3(a) and (b). From this

Loss	Task Weights			Segmentation	Instance	Inverse Depth
	Seg.	Inst.	Depth	IoU [%]	Mean Error [px]	Mean Error [px]
Segmentation only	1	0	0	59.4%	-	-
Instance only	0	1	0	-	4.61	-
Depth only	0	0	1	-	-	0.640
Unweighted sum of losses	0.333	0.333	0.333	50.1%	3.79	0.592
Approx. optimal weights	0.89	0.01	0.1	62.8%	3.61	0.549
2 task uncertainty weighting	✓	✓		61.0%	**3.42**	-
2 task uncertainty weighting	✓		✓	62.7%	-	0.533
2 task uncertainty weighting		✓	✓	-	3.54	0.539
3 task uncertainty weighting	✓	✓	✓	**63.4%**	3.50	**0.522**

Figure 4. Effect of multi-task learning on semantic segmentation, instance segmentation, and depth regression task on cityscapes dataset. Image Courtesy: [8]

we can see that we are able to recover true distributions of y_1 and y_2 despite training the neural network on corrupted values. Similarly, we are able to recover true values of σ_1 and σ_2. As both sigmas represents the uncertainty associated with tasks, the predicted value of the same should not be dependent on their initialization. We can observe that if we initialize them to either (d)1 or (e)5, we are able to reach the same values (9.95 and 0.95) for them. This shows that we are indeed able to get true task dependent uncertainty associated with the tasks using Eq:(3).

3. Scene Understanding Model

As mentioned before, in [8] authors experiment on task of semantic segmentation, instance segmentation, and depth regression tasks for multi-task learning. They use deep convolutional encoder decoder network, where same encoder is used to learn a task independent common representation. This common representation is given as an input to task specific decoder. Specifically, they use DeepLabV3 [2] with ResNet101 [5] as the base feature encoder followed by Atrous Spatial Pyramid Pooling [2] module to increase contextual awareness. Output of ASPP module is treated as task-independent features, which are then processed using task specific decoders which consists of a 3x3 convolutional layer with 256 features, followed by 1x1 convolution layer which has total task dependent features, i.e. 1 for depth regression or instance segmentation and c for semantic segmentation.

They use categorical cross-entropy loss for semantic segmentation, while for depth regression and instance segmentation they use pixel-wise mean absolute error. Performance of the system is measure with mean Intersection over Union (IoU) for semantic segmentation and mean absolute error for regression and instance segmentation.

3.1. Experiments

Cityscapers [3] dataset was used for experimentation. This dataset is a large dataset for road scene understanding. It provides semantic segmentation of 20 classes (Road, Sidewalk, building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person, rider, car, truck, bus, train, motorcycle, bicycle, background) and its corresponding instance segmentation. Ground for depth regression was acquired using SGM [6]. The dataset was collected from a number of cities in fine weather and consists of 2975 training and 500 validation images at 2048x1028 resolution.

Although, in [8] authors experimented on full resolution dataset once (Training Time: 5 days on 4 GPUs), for majority of the paper they used downsampled version of the dataset with downsampling factor of 8. This results in images of size 256x128. Results for this downsampled version of dataset, reported in [8] are given in Fig:4. From this table we can observe that when network is trained using uncertainty weighted loss, performance improves in comparison to either single task or manually tuned weights as well as naive uniform weights for multi-task. In addition, we can also observe that when more related task are added in multi-task learning performance improves further.

3.1.1 Reproduced Results

We tried to reproduced all the rows of Fig:4 for this project. Each row requires us to train a separate neural network which takes 2 days to train. Keeping time and resource constraints in mind, we were only able to reproduce experiments related to semantic segmentation and depth regression. Specifically, we reproduced rows-1,3,4, and 7. Although, this results in lower experiments, we should be able to see if uncertainty weighing helps or not. These results are tabulated in Table:3.1. From this table we can see that,

Loss	Segmentation Weight	Depth Weight	Mean IoU (%)	Depth MAE (px)
Segmentation Only	1	0	55.40	-
Depth Only	0	1	-	0.681
Unweighted (Seg + Depth)	1	1	55.26	0.680
Uncertainty (Seg + Depth)	0.2 (6.621)	1 (33.04)	56.44	0.572

Table 1. Effect of Uncertainty weighing on Multi-task learning for the task of semantic segmentation and depth regression on cityscapes dataset.

	Segmentation Weight	Regression Weight	Dice Tumour	Dice Enhance	Dice Core	MSE on Regression
Manually Tuned Weight	0.1	1.0	0.8705	0.5981	0.2725	0.0035
Task specific uncertainty weight	0.19 (42.48)	1.0 (223.27)	0.8812	0.6062	0.1502	0.0036

Table 2. Effect of Uncertainty weighing on Multi-task learning for the task of semantic segmentation and depth regression on Medical Imaging Brain Tumour dataset [9]

though we don't get as much improvement as reported in Fig:4, uncertainty weighted loss gives better performance than uniformly weighted loss and multi-task learning gives better performance than individual tasks. One of the reason because of which we may not have been able to get performance comparable to the one reported in the paper, is that the authors might be using some type of data augmentation but as it was not reported in the paper, we didn't include any data augmentation.

We used PyTorch Library for implementing this paper. We used publicly available code of DeepLab architecture [1] and modified it according to our need. All code related to multi-task uncertainty weight learning was coded by us. Our implementation is publicly available here [2]

3.1.2 Application to other domain

We applied the multi-task learning with uncertainty weighing to the domain of Modality synthesis in Medical Domain. Specifically, we took the architecture proposed in [9] which reported that multi-task learning helps in getting better synthesis of the missing MR modality in the presence of Tumours. We request readers to read [9] for further details about the network architecture, task specific loss, evaluation metrics, dataset, and pre-processing. In the paper, weights for two tasks, namely, Tumour segmentation and modality regression where finalized empherically and were fixed to 0.1 and 1 respectively. We ran the same experiment but instead of using manully defined task weights, we used uncertainty weighing covered in this report to weigh the losses. We report the performance in both cases in Table:2. From this table we can see that using uncertainty weighing doesn't result in a significant increase in performance. A closed look at the weights for each task in both cases reveals that this is due to the fact that task specific weights recovered using uncertainty weightage are almost identical to the manually tuned weights. So we can conclude that when manually tuned weights are optimal than uncertainty weighing doesn't result in a better performance.

4. Conclusion

We show how performance of a multi-task system changes with change in relative weights of the tasks. We looked into the different types of uncertainty and looked into the theory of task-dependent (Homoscedastic) uncertainty to weight multi-task learning system proposed in [8]. We show how this task-dependent uncertainty can be useful for learning relative weights and it can help in improving overall performance by experimenting in cityscapes dataset. We also applied it to the different domain of medical imaging show its effect.

Initially, value of $\log \sigma^2$ decrease rapidly as we add negation of that to the overall loss. This corresponds to increase in $\frac{1}{\sigma^2}$, which is a relative weight of the task. This results in unstable training in later epoch which can hinder overall performance. From the results we observed that though individual weights of this tasks can become large at the end of training their relative weights are not changing. In future, some of way of regularizing these relative weights should be explored so that $\sum_i \frac{1}{\sigma_i^2} = 1$. This way individual weights of the task won't rapidly increase and should help the training procedure.

One big assumption in the paper [8] was that task dependent uncertainty follows gaussian distribution. Thought this assumption results in a simple solution, which allows us to optimize the loss, this may not be true for many practical scenarios. In future, applying this method with different

[1]https://github.com/jfzhang95/pytorch-deeplab-xception
[2]https://github.com/RagMeh11/ECSE626_2019/tree/master/Project

distribution assumption should be explored.

References

[1] NHTSA. PE 16-007. 2017. Tesla crash preliminary evaluation report. Technical report, U.S. Department of Transportation, National Highway Traffic Safety Administration. 2

[2] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 5

[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. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 5

[4] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[6] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008. 5

[7] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584, 2017. 2

[8] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018. 1, 2, 3, 5, 6

[9] Raghav Mehta and Tal Arbel. Rs-net: Regression-segmentation 3d cnn for synthesis of full resolution missing brain mri in the presence of tumours. In *International Workshop on Simulation and Synthesis in Medical Imaging*, pages 119–129. Springer, 2018. 6