A. Model Architecture Details

We base our model on the recently introduced DeepLabV3 [10] segmentation architecture. We use ResNet101 [20] as our base feature encoder, with dilated convolutions, resulting in a feature map which is downsampled by a factor of 8 compared with the original input image. We then append dilated (atrous) convolutional ASPP module [10]. This module is designed to improve the contextual reasoning of the network. We use an ASPP module comprised of four parallel convolutional layers, with 256 output channels and dilation rates (1, 12, 24, 36), with kernel sizes (1^2, 3^2, 3^2, 3^2). Additionally, we also apply global average pooling to the encoded features, and convolve them to 256 dimensions with a 1×1 kernel. We apply batch normalisation to each of these layers and concatenate the resulting 1280 features together. This produces the shared representation between each task.

We then split the network, to decode this representation to a given task output. For each task, we construct a decoder consisting of two layers. First, we apply a 1×1 convolution, outputting 256 features, followed by batch normalisation and a non-linear activation. Finally, we convolve this output to the required dimensions for a given task. For classification, this will be equal to the number of semantic classes, otherwise the output will be 1 or 2 channels for depth or instance segmentation respectively. Finally, we apply bilinear upsampling to scale the output to the same resolution as the input.

The majority of the model's parameters and depth is in the feature encoding, with very little flexibility in each task decoder. This illustrates the attraction of multitask learning; most of the compute can be shared between each task to learn a better shared representation.

A.1. Optimisation

For all experiments, we use an initial learning rate of 2.5×10^{-3} and polynomial learning rate decay $(1 - \frac{iter}{max\ iter})^{0.9}$. We train using stochastic gradient descent, with Nesterov updates and momentum 0.9 and weight decay 10^4. We conduct all experiments in this paper using PyTorch.

For the experiments on the Tiny CityScapes validation dataset (using a down-sampled resolution of 128×256) we train over $50,000$ iterations, using 256×256 crops with batch size of 8 on a single NVIDIA 1080Ti GPU. We apply random horizontal flipping to the data.

For the full-scale CityScapes benchmark experiment, we train over $100,000$ iterations with a batch size of 16. We apply random horizontal flipping (with probability 0.5) and random scaling (selected from 0.7 - 2.0) to the data during training, before making a 512×512 crop. The training data is sampled uniformly, and is randomly shuffled for each epoch. Training takes five days on a single computer with four NVIDIA 1080Ti GPUs.

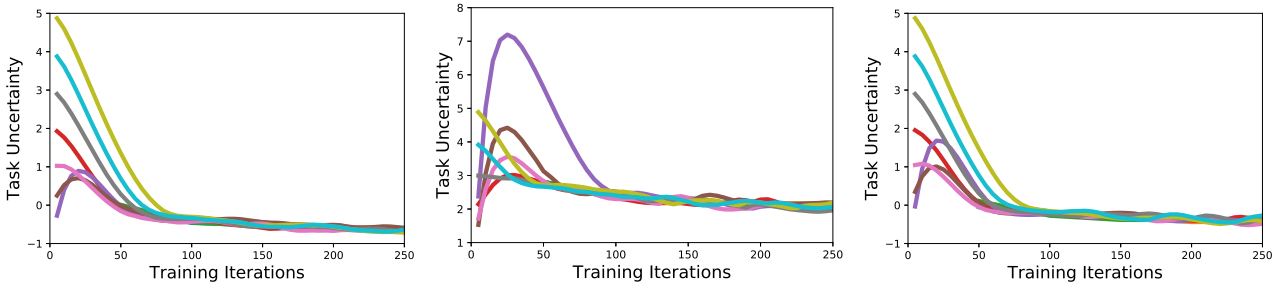
B. Further Analysis

This task uncertainty loss is also robust to the value we use to initialise the task uncertainty values. One of the attractive properties of our approach to weighting multi-task losses is that it is robust to the initialisation choice for the homoscedastic noise parameters. Figure 6 shows that for an array of initial choices of $\log \sigma^2$ from -2.0 to 5.0 the homoscedastic noise and task loss is able to converge to the same minima. Additionally, the homoscedastic noise terms converges after only 100 iterations, while the network requires $30,000+$ iterations to train. Therefore our model is robust to the choice of initial value for the weighting terms.

Figure 7 shows losses and uncertainty estimates for each task during training of the final model on the full-size CityScapes dataset. At a point 500 iterations into training, the model estimates task variance of 0.60, 62.5 and 13.5 for semantic segmentation, instance segmentation and depth regression, respectively. Becuase the losses are weighted by the inverse of the uncertainty estimates, this results in a task weighting ratio of approximately $23 : 0.22 : 1$ between semantics, instance and depth, respectively. At the conclusion of training, the three tasks have uncertainty estimates of 0.075, 3.25 and 20.4, which results in effective weighting between the tasks of $43: 0.16 : 1$. This shows how the task uncertainty estimates evolve over time, and the approximate final weightings the network learns. We observe they are far from uniform, as is often assumed in previous literature.

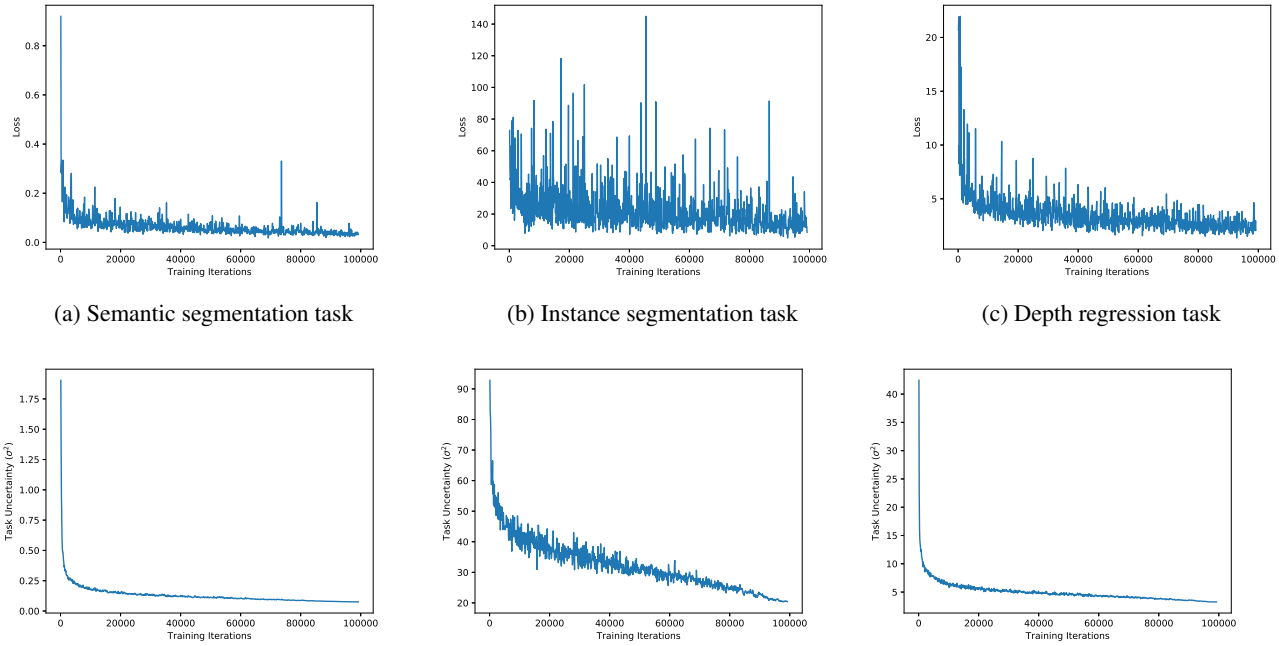
Interestingly, we observe that this loss allows the network to dynamically tune the weighting. Typically, the homoscedastic noise terms decrease in magnitude as training progresses. This makes sense, as during training the model becomes more effective at a task. Therefore the error, and uncertainty, will decrease. This has a side-effect of increasing the effective learning rate – because the overall uncertainty decreases, the weight for each task's loss increases. In our experiments we compensate for this by annealing the learning rate with a power law.

Finally, a comment on the model's failure modes. The model exhibits similar failure modes to state-of-the-art single-task models. For example, failure with objects out of the training distribution, occlusion or visually challenging situations. However, we also observe our multi-task model tends to fail with similar effect in all three modalities. Ie. an erroneous pixel's prediction in one task will often be highly correlated with error in another modality. Some examples can be seen in Figure 8.



(a) Semantic segmentation task (b) Instance segmentation task (c) Depth regression task

Figure 6: **Training plots showing convergence of homoscedastic noise and task loss** for an array of initialisation choices for the homoscedastic uncertainty terms for all three tasks. Each plot shows the the homoscedastic noise value optimises to the same solution from a variety of initialisations. Despite the network taking $10,000+$ iterations for the training loss to converge, the task uncertainty converges very rapidly after only 100 iterations.

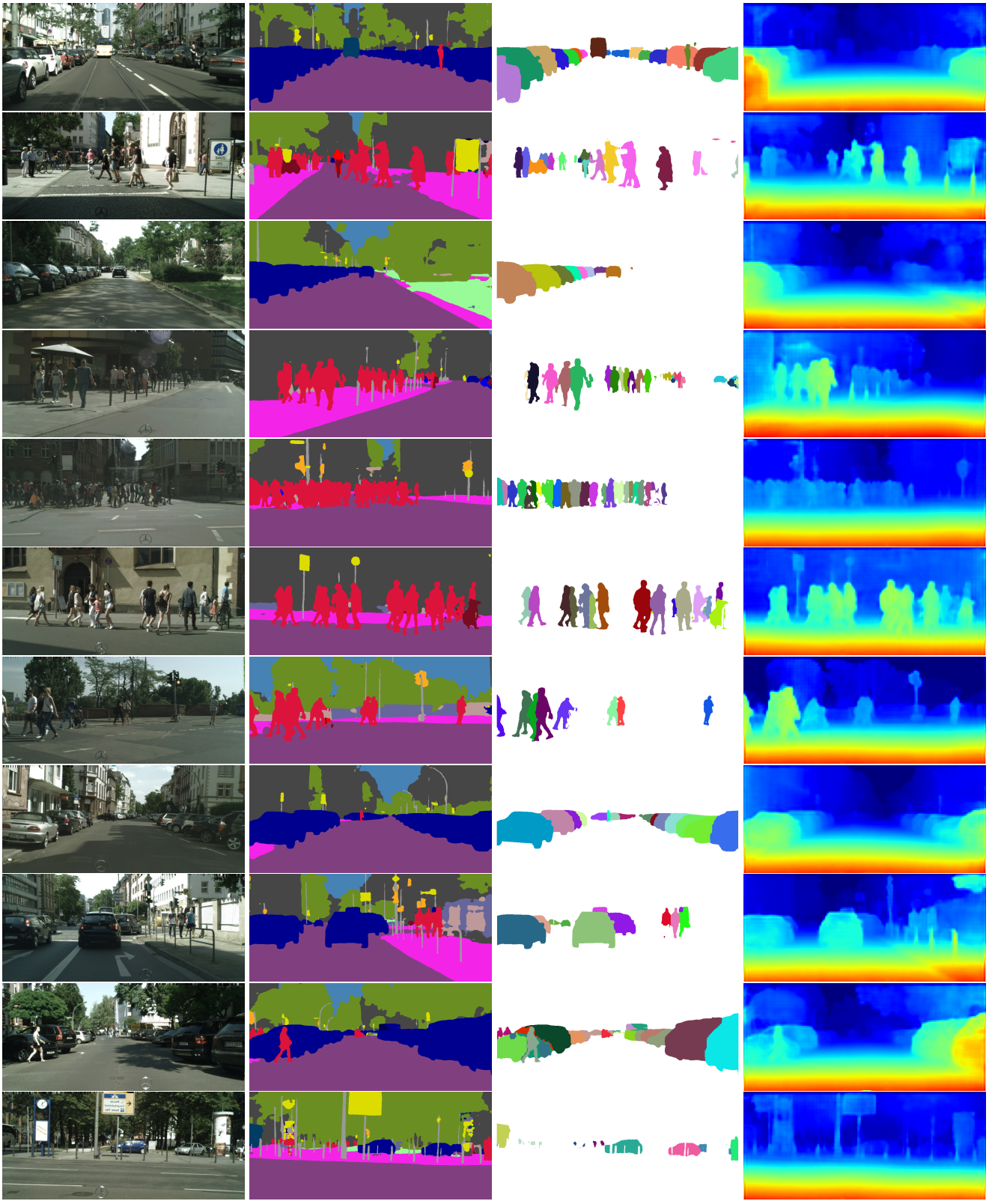


(a) Semantic segmentation task (b) Instance segmentation task (c) Depth regression task

(d) Semantic segmentation task (e) Instance segmentation task (f) Depth regression task

Figure 7: **Learning task uncertainty.** These training plots show the losses and task uncertainty estimates for each task during training. Results are shown for the final model, trained on the fullsize CityScapes dataset.

C. Further Qualitative Results



(a) Input image (b) Semantic segmentation (c) Instance segmentation (d) Depth regression

Figure 8: **More qualitative results on test images from the CityScapes dataset.**