AML Reproduction Challenge HT 2019

Oscar Key

Approximate word count: 4986

1 Introduction

We attempt to reproduce results from *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics; Kendall, Gal, Cipolla*. We successfully reproduce results analogous to those in the paper, on a Fashion MNIST task. We are partially successful at reproducing the original results from the Cityscapes task in the paper.

The core contribution of the paper is a principled method for weighting individual tasks in a multi-task loss function, based on the uncertainty of each task. The authors demonstrate the effectiveness of the new method by applying it to a computer vision task.

I was interested in this paper because the novel multi-task loss function it introduces is useful in many different applications of machine learning. For example semantic segmentation, which is part of the computer vision task that the paper describes, is immediately applicable in self driving cars (Treml et al. 2016). Additionally, the paper is convenient to reproduce. The authors use a publicly available dataset, and run a portion of their experiments on a similar amount of computing power to what we had available.

2 The paper

2.1 Motivation

In many applications of machine learning it is beneficial to have a single model achieve several objectives simultaneously. This can be more computationally efficient and produce more accurate results than training separate models for each task (Baxter 2000). However, this opens the question of how to combine the losses from each individual task into a single overall loss. Kendall et al. 2018 observe that the majority of prior work uses a simple sum of the individual loss functions:

$$L_{\text{total}} = \sum_i w_i L_i \tag{1}$$

where the weights, w_i, are fixed hyperparameters determined through a grid search. The authors note that the search for the weights may be very expensive, and may not find the optimal values because it is limited in resolution. The weights might also have varying optimal values at different epochs during training, and so any value of the fixed weights may not be optimal for the entire training procedure.

2.2 Hypothesis

The hypothesis that the paper presents and tests is that a multi-task model will achieve higher accuracy when using a loss which weights each task by a function of its learned uncertainty, rather than by fixed weights. Moreover, a multi-task model will achieve higher accuracy than several single task models learning each task individually. Specifically the paper proposes the following loss function, here stated for a two task model but extensible to any number of tasks:

$$\mathcal{L}(\boldsymbol{W}, \sigma_1, \sigma_2) = \frac{1}{\sigma_1}\mathcal{L}_1(\boldsymbol{W}) + \frac{1}{\sigma_1^2}\mathcal{L}_2(\boldsymbol{W}) + \log\sigma_1 + \log\sigma_2 \tag{2}$$

where $\mathcal{L}_1$ is the L1 loss of a regression task and $\mathcal{L}_2$ the cross entropy loss of a classification task. σ_1 and σ_2 are parameters of the model, which correspond to the uncertainty of each task, as discussed below.

2.3 Theoretical justification

The derivation in this section closely follows Kendall et al. 2018.

The authors justify the hypothesis loss function by showing that it weights each task by a function of its *homoscedastic aleatoric* or *task-dependent* uncertainty. They derive the loss function by applying maximum likelihood estimation to a probabilistic model of a multi-task model.

First let us review the different types of uncertainty that Bayesian techniques model (Kendall et al. 2018):

Epistemic uncertainty is uncertainty about what model generated the data, because there are many possible models which can fit the limited training data available.

Aleatoric uncertainty is noise in the observations, because we cannot observe all explanatory variables with infinite precision e.g. noise in a sensor reading.

Aleatoric uncertainty is then further divided into:

Heteroscedastic uncertainty (data dependent) which is a function of the input data e.g. predictions of arrival time of a bus based on departure time might have more variance during rush hour.

Homoscedastic uncertainty (task dependent) which is fixed for a particular task, and so is not a function of the input data e.g. predicting a sequence of dice rolls, the uncertainty about each roll is the same no matter where it is in the sequence.

We now show that the weight of a task in the overall loss function should be a function of its homoscedastic aleatoric uncertainty, σ. The multi-task model we will consider has n outputs, each of which may be either a regression or a classification output.

2.3.1 Regression tasks

First let us look at a single regression task. Kendall et al. 2018 derive the loss function for Gaussian noise which corresponds to L2 loss. However, in the experiments they use L1 loss, which we can instead derive by modelling Laplace noise:

$$p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) = \text{Lap}(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) \tag{3}$$

Here $\boldsymbol{f}^{\boldsymbol{W}}$ represents a neural network with parameters $\boldsymbol{W}$, $\boldsymbol{x}$ is the model input, $\boldsymbol{y}$ is the model output and σ is the homoscedastic aleatoric uncertainty in the observations. Note how σ is fixed for this particular task, it does not depend on the input data nor the uncertainty in the model.

To fit the model to some training data, $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, we often use maximum likelihood estimation. We maximise the probability of observing the output data given the input data and model parameters, $p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}))$ (Murphy 2012). By applying this to our Laplace model of $\boldsymbol{y}$ above, and then taking logs, we get the log likelihood:

$$\begin{aligned}
p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) &= \text{Lap}(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) && (4)\\
&= \frac{1}{2\sigma} \exp\left(-\frac{|\boldsymbol{y} - \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})|}{\sigma} \right) && (5)\\
\log p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) &= -\frac{1}{\sigma} \left| \boldsymbol{y} - \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}) \right| - \log 2\sigma && (6)\\
-\log p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) &= \frac{1}{\sigma} \left| \boldsymbol{y} - \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}) \right| + \log 2\sigma && (7)\\
&\propto \frac{1}{\sigma} \left| \boldsymbol{y} - \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}) \right| + \log \sigma && (8)
\end{aligned}$$

By minimising equation 8 (the negative log likelihood, NLL) w.r.t. $\boldsymbol{W}$, summed over all training data, we fit a model with a single output to the training data.

Now let us examine the multi-task case where we have several regression outputs $\hat{\boldsymbol{y}}_1$, $\hat{\boldsymbol{y}}_2$, ... We factorise them and apply the same maximum likelihood technique:

$$\begin{aligned}
p(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots \mid \boldsymbol{f}^{\boldsymbol{W}}(x), \sigma_1, \sigma_2) &= p(\boldsymbol{y}_1 \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_1) \cdot p(\boldsymbol{y}_2 \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_2) \cdot \dots && (9)\\
&= \text{Lap}(\boldsymbol{y}_1; \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_1) \cdot \text{Lap}(\boldsymbol{y}_2; \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_2) \cdot \dots && (10)\\
-\log p(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots \mid \boldsymbol{f}^{\boldsymbol{W}}(x), \sigma_1, \sigma_2) &\propto \frac{1}{\sigma_1} \left| \boldsymbol{y}_1 - \boldsymbol{f}_1^{\boldsymbol{W}}(\boldsymbol{x}) \right| + \frac{1}{\sigma_2} \left| \boldsymbol{y}_2 - \boldsymbol{f}_2^{\boldsymbol{W}}(\boldsymbol{x}) \right| + \log \sigma_1 + \log \sigma_2 + \dots && (11)
\end{aligned}$$

Thus we get the overall loss function as in the hypothesis:

$$\mathcal{L}(\boldsymbol{W}, \sigma_1, \sigma_2) \propto \frac{1}{\sigma_1} \mathcal{L}_1(\boldsymbol{W}) + \frac{1}{\sigma_2} \mathcal{L}_2(\boldsymbol{W}) + \log \sigma_1 + \log \sigma_2 + \dots \tag{12}$$

where $\mathcal{L}_i$ is the L1 loss of each individual task. We treat each σ_i as a learnable parameter of the model.

2.3.2 Classification tasks

We can extend the loss function to additionally support classification tasks. We model the classification task using a Softmax function:

$$p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) = \text{Softmax}\left(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})\right) \tag{13}$$

where we treat the output as a probability vector from which we sample to get the predicted class. In order to incorporate heteroscedastic uncertainty in this model we first scale the output of the network as follows:

$$p(\boldsymbol{y} \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) = \text{Softmax}\left(\frac{1}{\sigma^2}\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})\right) \tag{14}$$

Thus the softmax has the form:

$$\text{Softmax}_c\left(\frac{1}{\sigma^2}\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})\right) = \frac{\exp(f_c^W(\boldsymbol{x})/\sigma^2)}{\sum_{c'}\exp(f_{c'}^W(\boldsymbol{x})/\sigma^2)} \tag{15}$$

We can see that the softmax now has the same functional form as the Boltzmann distribution, which justifies why we scale the network output by $\frac{1}{\sigma^2}$. The Boltzmann distribution, which gives the probability that a system is in a given state c, is defined as:

$$\text{Boltz}(c) = \frac{\exp(-\epsilon_c/kT)}{\sum_{c'}\exp(-\epsilon_{c'}/kT)} \tag{16}$$

Here ϵ_c is the energy of the state c, and T is the temperature of the state. By comparing the equations 15 and 16 we can see that $f_c^W(\boldsymbol{x})$ corresponds to ϵ_c, and σ^2 corresponds to kT. In the Boltzmann distribution T is related to the entropy of the system, the higher the temperature the higher the entropy and uncertainty. Thus larger values of σ^2 are associated with higher uncertainty.

Again to fit the model we use maximum likelihood estimation, so we compute the log likelihood:

$$p(\boldsymbol{y} = c \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) = \frac{\exp(f_c^W(\boldsymbol{x})/\sigma^2)}{\sum_{c'}\exp(f_{c'}^W(\boldsymbol{x})/\sigma^2)} \tag{17}$$

$$\log p(\boldsymbol{y} = c \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) = \frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x}) - \log\sum_{c'}\exp\left(\frac{1}{\sigma^2}\boldsymbol{f}_{c'}^{\boldsymbol{W}}(\boldsymbol{x})\right) \tag{18}$$

For classification tasks we often use the cross entropy loss function:

$$l_{\text{CE}}(\boldsymbol{y} = c, \boldsymbol{W}) = -\log(\text{Softmax}(\boldsymbol{y} = c, \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}))) \tag{19}$$

$$\log l_{\text{CE}}(\boldsymbol{y} = c, \boldsymbol{W}) = \log\left(\sum_{c'}\exp\left(\boldsymbol{f}_{c'}^{\boldsymbol{W}}(\boldsymbol{x})\right)\right) - \boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x}) \tag{20}$$

We can rewrite equation 18 in terms of l_{CE}:

$$\begin{aligned}\log p(\boldsymbol{y} = c \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) &= \frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x}) - \log\sum_{c'}\exp\left(\frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x})\right) \\ &\quad + \frac{1}{\sigma^2}\log\sum_c\exp(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) - \frac{1}{\sigma^2}\log\sum_c\exp(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) \end{aligned} \tag{21}$$

$$\begin{aligned}&= \frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x}) - \frac{1}{\sigma^2}\log\sum_c\exp(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) \\ &\quad + \log\left(\sum_c\exp(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}))\right)^{\frac{1}{\sigma^2}} - \log\sum_{c'}\exp\left(\frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x})\right)\end{aligned} \tag{22}$$

$$= -\frac{1}{\sigma^2}l_{\text{CE}}(\boldsymbol{y} = c, \boldsymbol{W}) - \log\frac{\sum_{c'}\exp\left(\frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x})\right)}{(\sum_c\exp(\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})))^{\frac{1}{\sigma^2}}} \tag{23}$$

Kendall et al. 2018 note that:

$$\left(\sum_{c'}\exp\left(\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x})\right)\right)^{\frac{1}{\sigma^2}} \approx \frac{1}{\sigma}\sum_{c'}\exp\left(\frac{1}{\sigma^2}\boldsymbol{f}_c^{\boldsymbol{W}}(\boldsymbol{x})\right) \tag{24}$$

An assumption that allows us to simplify equation 23 to the NLL:

$$-\log p(\boldsymbol{y} = c \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma) \approx \frac{1}{\sigma^2}l_{\text{CE}}(\boldsymbol{y} = c, \boldsymbol{W}) + \log\sigma \tag{25}$$

If our model has multiple classification outputs then we can combine them in the same fashion as regression outputs:

$$p(\boldsymbol{y}_1, \boldsymbol{y}_2 \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_1, \sigma_2) = \text{Softmax}(\boldsymbol{y}_1; \frac{1}{\sigma_1}\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) \cdot \text{Softmax}(\boldsymbol{y}_2; \frac{1}{\sigma_2}\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) \tag{26}$$

$$\log p(\boldsymbol{y}_1, \boldsymbol{y}_2 \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_1, \sigma_2) = \log \text{Softmax}(\boldsymbol{y}_1; \frac{1}{\sigma_1}\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) + \log \text{Softmax}(\boldsymbol{y}_2; \frac{1}{\sigma_2}\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) \tag{27}$$

And the total loss is:

$$\mathcal{L}(\boldsymbol{W}, \sigma_1, \sigma_2) = \frac{1}{\sigma_1^2}\mathcal{L}_1(\boldsymbol{W}) + \frac{1}{\sigma_2^2}\mathcal{L}_2(\boldsymbol{W}) + \log \sigma_1 + \log \sigma_2 \tag{28}$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ are the cross entropy losses of each individual task, and σ_i are learned parameters.

2.3.3 Joint tasks

We can combine the results of the previous two sections for any arbitrary mixture of classification and regression tasks. For example, if the model has a regression output $\boldsymbol{y}_1$ and a classification output $\boldsymbol{y}_2$ then the loss function is:

$$p(\boldsymbol{y}_1, \boldsymbol{y}_2 \mid \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_1, \sigma_2) = \text{Lap}(\boldsymbol{y}_1; \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}), \sigma_1) \cdot \text{Softmax}(\boldsymbol{y}_2; \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})) \tag{29}$$

$$\mathcal{L}(\boldsymbol{W}, \sigma_1, \sigma_2) = \frac{1}{\sigma_1}\mathcal{L}_1(\boldsymbol{W}) + \frac{1}{\sigma_2^2}\mathcal{L}_2(\boldsymbol{W}) + \log \sigma_1 + \log \sigma_2 \tag{30}$$

where $\mathcal{L}_1$ is the L1 loss for the first task, and $\mathcal{L}_2$ is the cross entropy loss for the second task.

We will use this approach later to build the loss functions for our experiments.

2.4 Novel architecture

The second contribution of the paper is a novel computer vision architecture which performs semantic segmentation, instance segmentation and depth regression simultaneously (see section 4.1). The model has a large convolutional encoder which accepts the raw input image and outputs a feature map. There are then decoders for each individual task. The idea is that the encoder learns a shared representation that is useful for all of the tasks, allowing the decoders to be smaller. Additionally the multiple decoders might regularise the shared representation, so performing several tasks at once may have better performance than performing each individually. I describe the model from the paper in more detail in section 4.2.1.

3 Fashion MNIST task

First we demonstrate the hypothesis on a small task using the Fashion-MNIST dataset, where we could iterate more quickly.

3.1 Experimental methods

Fashion-MNIST contains a large number of 28x28 greyscale images of clothing, taken from a clothing catalogue (Xiao et al. 2017). Figure 5 shows some example images. The images are divided into 10 categories e.g. shirt, shoe. The two tasks we implement are:

Classification

Task	Classification as one of the 10 classes
Loss function	Cross entropy loss
Accuracy measure	Percentage of images in test set classified correctly

Reconstruction

Task	Compress the image down to a low dimensional representation, then reconstruct the original image (i.e. an autoencoder)
Loss function	L1 loss against the original image
Accuracy measure	Loss as above

The structure of the model is shown in figure 1. It is a convolutional autoencoder, with a second decoder added for the classification task containing two fully connected layers.

For the fixed weight experiments we use the following loss function:

$$L_{\text{total}}(\boldsymbol{W}) = w_1 \mathcal{L}_1(\boldsymbol{W}) + w_2 \mathcal{L}_2(\boldsymbol{W}) \tag{31}$$

where $\mathcal{L}_1$ is the loss for the classification task and $\mathcal{L}_2$ the loss for the reconstruction task. For experiments involving only one of the tasks, we set $w_i = 0$ for the disabled task.

For the learned uncertainty weights experiments the loss function is:

$$\mathcal{L}_{\text{total}}(\boldsymbol{W}, s_1, s_2) = e^{-s_1}\mathcal{L}_1(\boldsymbol{W}) + \frac{1}{2}e^{-s_2}\mathcal{L}_2(\boldsymbol{W}) + \frac{1}{2}(s_1 + s_2) \tag{32}$$

where $s_i = \log \sigma_i^2$. Here we follow Kendall et al. 2018 by learning the log uncertainty s_i, instead of σ_i, to improve numerical stability.

Note that this is not actually the correct loss function as it assumes that $\mathcal{L}_2$ is the L2 loss (following the derivation by Kendall et al. 2018), when it is the L1 loss. The correct loss function is:

$$\mathcal{L}_{\text{total}}(\boldsymbol{W}, s_1, s_2, s_3) = e^{-s_1}\mathcal{L}_1(\boldsymbol{W}) + e^{-\frac{1}{2}s_2}\mathcal{L}_2(\boldsymbol{W}) + \frac{1}{2}(s_1 + s_2) \tag{33}$$

However we only discovered this after running the experiments. I discuss the implications in the evaluation.

We preprocessed both datasets by normalising them to have mean 0, standard deviation 1. We used the Adam optimiser with a learning rate of 0.0001.

The code for the Fashion MNIST experiments is in `multitask-learning/mnist`.

<table>
<tr><td colspan="2">input image (28x28)</td></tr>
<tr><td colspan="2">conv2d kernel=3x3 filters=32 padding=1 stride=1 (ReLU)</td></tr>
<tr><td colspan="2">conv2d kernel=3x3 filters=32 padding=1 stride=1 (ReLU)</td></tr>
<tr><td colspan="2">maxpool2d kernel=2x2 stride=2</td></tr>
<tr><td colspan="2">conv2d kernel=2x2 filters=32 padding=0 stride=1 (ReLU)</td></tr>
<tr><td colspan="2">conv2d kernel=2x2 filters=16 padding=0 stride=1 (ReLU)</td></tr>
<tr><td colspan="2">maxpool2d kernel=2x2 stride=2</td></tr>
<tr><td colspan="2">conv2d kernel=2x2 filters=16 padding=1 stride=2 (ReLU)</td></tr>
<tr><td colspan="2">maxpool2d kernel=2x2 stride=2</td></tr>
<tr><td colspan="2">encoder output (2x2x16)</td></tr>
<tr><td>fully connected in=64 out=128 (ReLU)</td><td>convtranspose2d kernel=2x2 filters=2 padding=0 stride=2 (ReLU)</td></tr>
<tr><td>fully connected h=128 out=10 (softmax)</td><td>convtranspose2d kernel=2x2 filters=2 padding=0 stride=2 (ReLU)</td></tr>
<tr><td rowspan="2"></td><td>convtranspose2d kernel=2x2 filters=2 padding=0 stride=2 (ReLU)</td></tr>
<tr><td>convtranspose2d kernel=2x2 filters=2 padding=0 stride=2 (Tanh)</td></tr>
<tr><td>class likelihoods (10x1)</td><td>reconstructed output (28x28)</td></tr>
</table>

Figure 1: The encoder and decoders of the Fashion-MNIST model. The top section is the encoder, and the two bottom halves are each a decoder.

3.2 Experiments and results

We perform experiments analogous to table 1 in Kendall et al. 2018. Specifically we compare running each task individually, with equal weighting, with near optimal weights and with learned uncertainty weights. To find the near optimal weights we performed a grid search, see figure 3.

The results, shown in table 1 and figure 2, match those found by Kendall et al. 2018 and support the hypothesis. For the reconstruction task both multi-task experiments achieve lower loss than the single task experiment, and the learned uncertainty weights outperform the optimal fixed weights. For the classification task, all experiments achieve similar accuracy, but the learned uncertainty weights converge faster.

Figure 4 shows that the task uncertainties that the model learns. We expect that the model would always learn the same uncertainty for each task because the uncertainty for a particular task is constant, independent of initialisation or experiment configuration. We see that for the classification task the model converges to the same uncertainty. However, for the reconstruction task the learned uncertainty varies. We also note that in both cases the uncertainty values converge more slowly than the authors of the original paper claim. They find that the uncertainties converge in 100s of iterations, i.e. within the first epoch, whereas in our experiments it takes 10s of epochs.

Finally, qualitative results from the reconstruction task are visible in figure 5. While the results are not state of the art, they confirm that the network is learning the task correctly.

Loss	Task weights		Classification accuracy (%) (higher better)	Reconstruction error (lower better)
	Class.	Recon.		
Classification only	1	0	89.7	-
Reconstruction only	0	1	-	0.0873
Unweighted sum of losses	0.5	0.5	89.3	0.0847
Approx. optimal weights	0.4	0.6	90.0	0.0785
Learned uncertainty weights	-	-	**90.2**	**0.0749**

Table 1: Results on the Fashion-MNIST dataset.

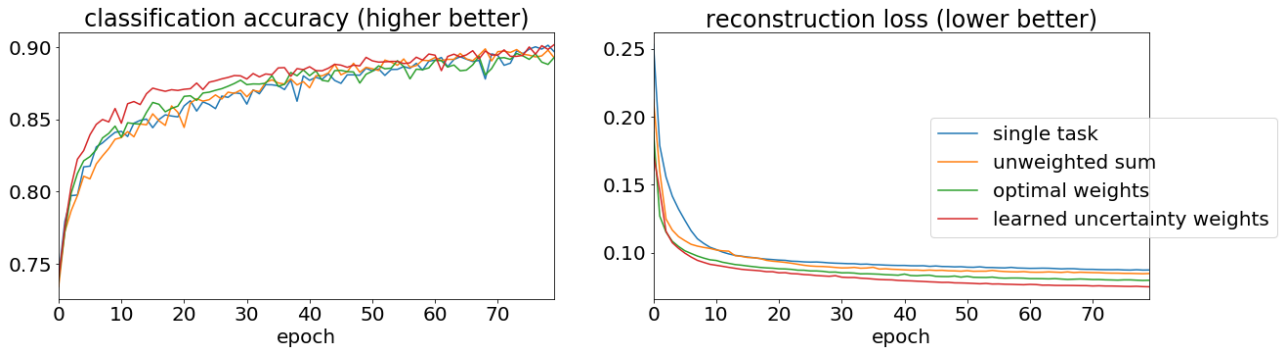

Figure 2: Accuracy and error on the validation set for the tasks in table 1.

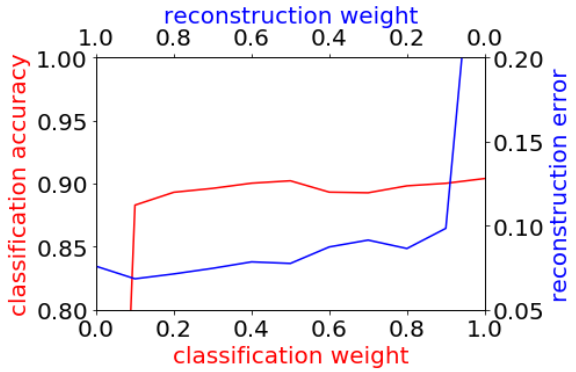

Figure 3: Grid search of fixed loss weights for the classification and reconstruction tasks. Higher classification error and lower reconstruction error is better. From this we choose optimal weights: classification weight=0.4, reconstruction weight=0.6.

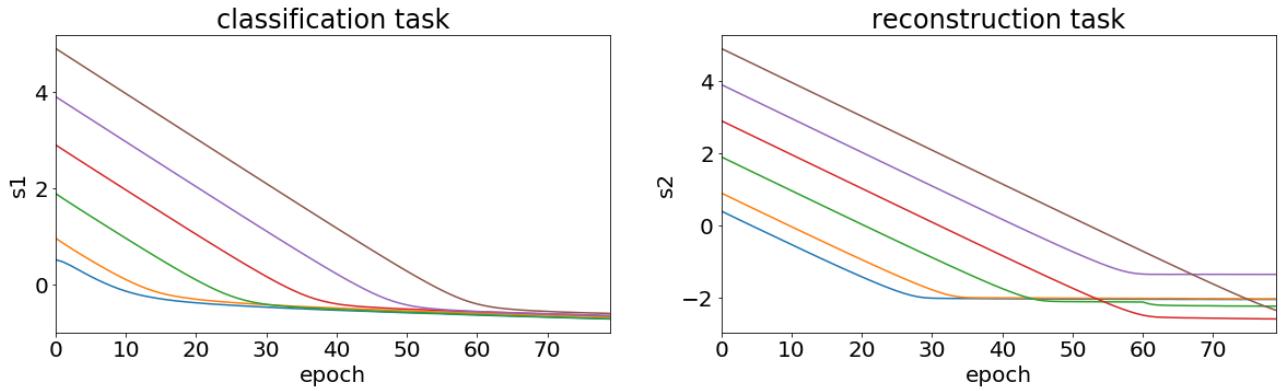

Figure 4: Uncertainty of each task over time, showing that the model converges to roughly the same uncertainty value no matter the initial value. We plot $s = log(\sigma^2)$.

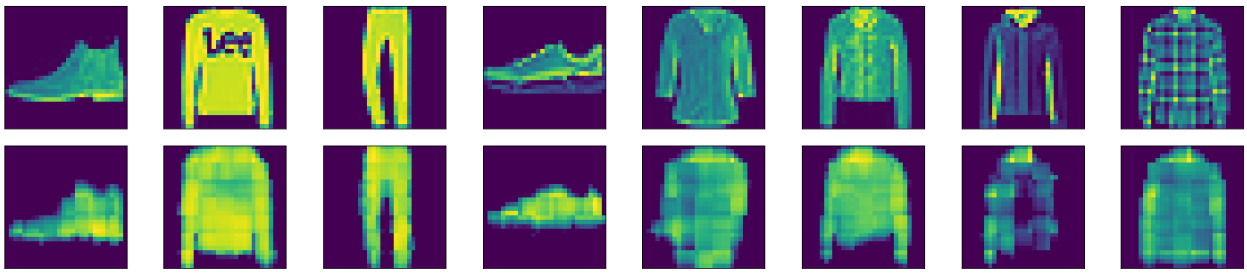

Figure 5: Qualitative outputs of the reconstruction decoder. The top line is the original image, and the bottom line is the reconstruction.

4 Cityscapes task

We attempt to reproduce the original experiments that Kendall et al. 2018 describe.

4.1 Tasks and dataset

Kendall et al. 2018 choose to demonstrate the hypothesis loss function by applying it to three computer vision tasks. The tasks they choose are:

Semantic segmentation Label each pixel in the image with the class it belongs to e.g. car, person.

Instance segmentation Identify all the individual instances of each class in the image, and label each pixel with the particular instance it belongs to e.g. person 1, car 3.

Depth regression Compute the distance from the camera of the point contained in each pixel in the image.

The authors use the Cityscapes dataset (Cordts et al. 2016) which contains labelled data for each of the three tasks above. Figure 6 shows an example image and its labelings. Additionally the authors create a copy of the dataset downsampled 8 times to 128x256, which they name "Tiny Cityscapes". We use this downsampled dataset for our experiments.

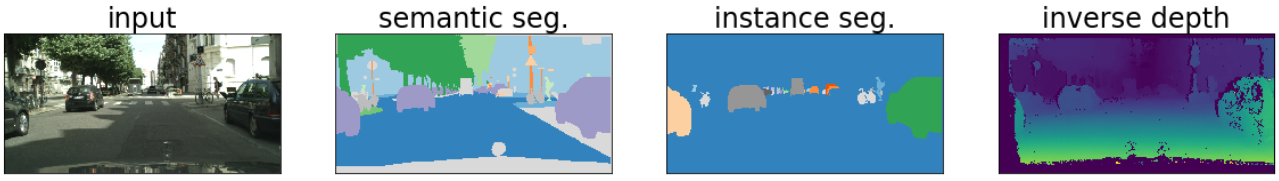

Figure 6: An example frame from the Tiny Cityscapes dataset. From left to right: the input to the model, semantic segmentation, instance segmentation, inverse depth.

4.2 Implementation details

4.2.1 Network architecture

The following architecture is as close as possible to what the Kendall et al. 2018 describe. Where the paper was ambiguous we contacted the original authors for clarification.

Overall the model has two parts. First an encoder network accepts a colour image and computes a latent representation. Then, from this latent representation, three decoders compute the output for each task in parallel. The idea is that the encoder learns to transform the input image into a feature map that is a useful representation for all three tasks. The individual decoders then specialise this for a particular task. The encoder portion of the network is quite large, while the decoders are relatively small.

Figure 7 shows the encoder. The majority of the encoder is ResNet101, which is a deep neural network architecture designed for image recognition (He et al. 2016). We use the implementation available in the PyTorch torchvision package. However, the Kendall et al. 2018 remove the final two layers, pooling and fully connected, and instead attach an Atrous Spatial Pyramidal Pooling (ASPP) module.

The reason for this modification is that the pooling layers at the end of ResNet discard spatial information about where particular features are in the image (Chen et al. 2017). For recognition tasks this is not an issue, however for the tasks in this experiment we must preserve the spatial location of features in order to be able to label individual pixels. Chen et al. 2017 describe ASPP to solve this problem. It uses several atrous convolutions at different scales, which are applied in parallel and the results pooled. Atrous (or dilated) convolutions, as shown in figure 9, have padding between the cells of the kernel so they cover a larger area of the input with the same number of parameters. This allows the kernel to detect larger features without the additional computational cost of using a larger dense kernel. The ASPP module applies 4 atrous convolutions in parallel to detect features at different scales, in an arrangement known as spatial pyramidal pooling. Additionally, in parallel, the ASPP module applies global average pooling. The resulting 5 feature maps are concatenated together to produce the output of the encoder.

Chen et al. 2017 state that the advantage of this architecture is that it is able to produce dense feature maps yet can still be initialised using weights from pre-trained ResNet101, and then fine tuned to the specific task, because the majority of the network is the same.

Figure 8 shows a single decoder, which converts the latent representation to the output specific to a particular task. The decoder for each task has the same structure with exception of the second convolution. This is specific to each task:

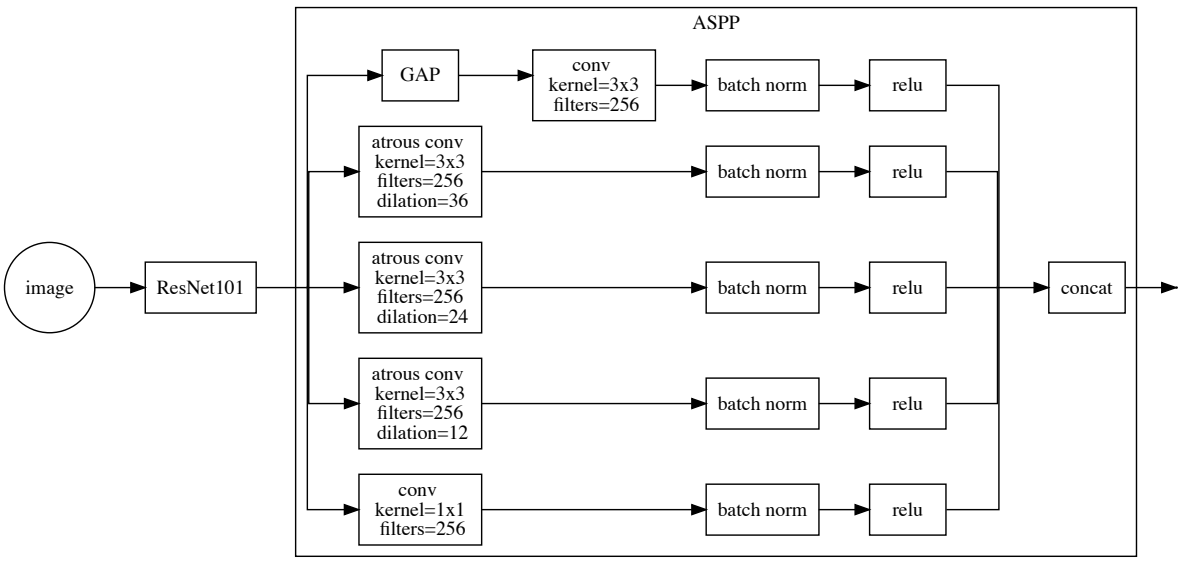

Figure 7: The structure of the encoder.

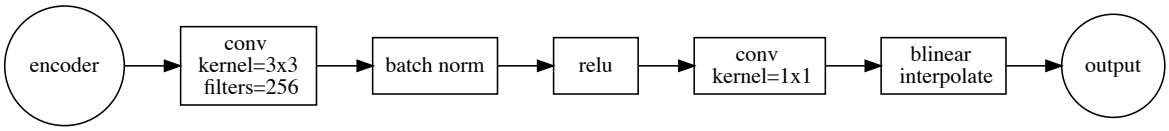

Figure 8: The structure of a single decoder. There are three decoders, one for each task.

(a) No dilation (b) Dilation = 1

Figure 9: A 3x3 convolution kernel with (a) no dilation, (b) dilation 1. Note how both kernels have the same number of parameters, but (b) covers a large area of the input.

- Semantic segmentation: 20 filters, a one-hot encoding of the 20 classes
- Instance segmentation: 2 filters, for the x and y components of the instance vectors respectively
- Depth segmentation: 1 filter, for the inverse depth output

The final bilinear interpolation layer upsamples the convolution output 8 times to the size of the original image.

We initialised the ResNet101 portion of the network using weights pre-trained on ImageNet from the PyTorch model zoo. All other weights are initialised randomly.

4.2.2 Loss function

For experiments with fixed loss weightings we use the following loss function:

$$L_{\text{total}}(\boldsymbol{W}) = w_1\mathcal{L}_1(\boldsymbol{W}) + w_2\mathcal{L}_2(\boldsymbol{W}) + w_3\mathcal{L}_3(\boldsymbol{W}) \tag{34}$$

where $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ are the losses for semantic segmentation, instance segmentation and inverse depth regression respectively (defined later). For experiments involving only a subset of the tasks, we set the weight for those tasks to zero.

For experiments which learn the loss weights, we use the following loss function:

$$\mathcal{L}_{\text{total}}(\boldsymbol{W}, s_1, s_2, s_3) = e^{-s_1}\mathcal{L}_1(\boldsymbol{W}) + \frac{1}{2}e^{-s_2}\mathcal{L}_2(\boldsymbol{W}) + \frac{1}{2}e^{-s_3}\mathcal{L}_3(\boldsymbol{W}) + \frac{1}{2}(s_1 + s_2 + s_3) \tag{35}$$

where again $s_i = \log \sigma_i^2$. Again this is not the correct loss function because as it assumes that $\mathcal{L}_2$ and $\mathcal{L}_3$ are L2 loss functions when they are in fact L1 loss functions. I discuss this in the evaluation.

4.2.3 Data pre-processing

To generate Tiny Cityscapes we downsample using a nearest neighbour algorithm. For the depth information an extra step is required as this information is provided as disparity. As the images are 8 times smaller, to maintain the same depth values we multiply all the disparity values by a factor of 8.

As we initialised the encoder using weights pre-trained on the ImageNet dataset, we normalised all input images to match the mean and standard deviation of ImageNet.

For all experiments we augment the training data using random horizontal flipping, as in the original paper.

4.2.4 Task specifics

This section gives implementation details specific to each task.

Semantic segmentation

Data	We used the 20 classes from the Cityscapes benchmarks (Cordts et al. 2016).
Loss function	Cross entropy loss, ignoring any pixels in unused classes.
Accuracy measure	Intersection of Union (IoU): We calculate for each class $\text{IoU}_c = \frac{\text{TP}}{\text{TP+FP+FN}}$, where TP is the number of pixels labelled c both in the ground truth and the prediction, FP is the number of pixels labelled c in the prediction but not the ground truth and FN is the number of pixels labelled c in the ground truth but not the prediction. We then take the mean over all the classes, $\text{IoU}_{\text{total}} = \frac{1}{\lvert C\rvert}\sum_{c\in C}\text{IoU}_c$ (Cordts et al. 2016).

Instance segmentation

Data	Framed as a regression task. For each pixel associated with an instance, we compute a vector which points at the centre of the instance. We only include instances for the classes enabled in Cityscapes benchmarks (Cordts et al. 2016).
Loss function	L1 loss, ignoring any pixels which are not part of an instance.
Visualisation	Use the semantic segmentation output to filter to pixels assigned to classes for which instances are enabled. Cluster the instance vector associated with each pixel. Each cluster represents an instance. Kendall et al. 2018 use the OPTICS algorithm for clustering. For simplicity we instead used DBSCAN, which is a similar algorithm available in Scikit-Learn (Pedregosa et al. 2011).
Accuracy measure	L1 loss

Depth regression

Data	The model does not regress depth directly but instead the disparity value of each pixel. Any pixels in the semantic class "sky" have their disparity set to 0, which is equivalent to infinite depth.
Loss function	L1 loss
Visualisation	Convert disparity values to depth using $\frac{\text{baseline} \cdot \text{focal length})}{\text{disparity}}$.
Accuracy measure	L1 loss

4.3 Experiments and results

We chose to reproduce the results from Table 1 in the paper. This is the full three task version of the Fashion-MNIST results above. We chose these results because they are sufficient to support the hypothesis, while requiring less computing power than the other results in the paper because they use Tiny Cityscapes. We used the hyperparameters given in the paper. Our results are visible in table 2, and qualitative results of the three task learned uncertainty weights model are visible in figure 10.

Examining the absolute numbers we can see that for semantic and instance segmentation our results are substantially weaker than the original paper in all cases. For depth our results appear to be better than the original paper, and indeed the qualitative results in 10 reflect this. However it was not completely clear how the original authors pre-processed the disparity data, so the discrepancy could be due to this.

Examining the relative difference between results we also differ. While the original paper finds that semantic and depth segmentation perform better in the 3 task uncertainty weighting experiment, we find that they perform better in the 2 task uncertainty weighting experiment. Additionally, we also find that instance segmentation performs better when trained alone rather than with semantic segmentation as in the original paper.

The results above led us to believe that : there was a bug in the model, the hyperparameters in the paper were incorrect, or there was a difference between our data processing and that in the paper which meant the hyperparameters in the paper did not apply. Thus we decided to first debug the model, and then perform a hyperparameter search. Rather than debug the entire model we focussed on the semantic segmentation task, where we tried to match the single task IoU value in the original paper of 59.4%.

Single batch test To check for obvious bugs in the model we tested training it on a single batch of size 3. Our model achieve an IoU of 64% out of a maximum possible of 70%.

The maximum possible value is not 100% because small output of the network restricts the performance. The output is 8 times smaller than the image, in our case the 16x32 compared to 128x256. We then bilinearly interpolate the output up to the size of the input. Cordts et al. 2016, table 3 reports the expected maximum IoU values when scaling up from smaller outputs to the full size Cityscapes. However, as our ground truth has lower resolution we expect different values. To approximate the maximum IoU we directly optimised a 16x32 image to match the ground truth when interpolated up to 128x256. This method found an IoU of 70%. Thus we decided that an IoU of 64% was reasonable.

We also performed a detailed comparison of our network architecture to a similar PyTorch implementation of DeepLabV3 (Liu 2018), and found no errors. We also checked the weights for NaNs. Given this and the reasonable result on a single batch above we decided to proceed with the hyperparameter search. The results are summarised in table 3.

Learning rate search The results of the learning rate search are in table 3a. We found that a high learning rate of 0.01 resulted in a higher IoU. Figure 11 shows the training loss and validation IoU curves for the different learning rates, showing that this high learning rate did not cause the loss to be unstable. We later added a learning rate scheduler which decreased the learning rate by a factor of 10 when the validation loss plateaued for 10 epochs.

Dilation sizes The ASSP module in the encoder (see figure 7) uses dilated convolutions with dilations of sizes 12, 24, 46. However, when using Tiny Cityscapes the input to these convolutions is 16x32, which means that the majority of the filter will positioned over the padding rather than the image. We ran an experiment using smaller dilations (2, 4, 8) but this did not improve the result, see table 3c.

Reducing overfitting From the training and validation loss curves in figure 12 it is clear that the model is overfitting. Thus we tested several techniques for reducing overfitting. First we ran a search over the optimiser weight decay parameter, which is equivalent to L2 regularisation when using the SGD optimiser (Goodfellow et al. 2016). This improved the results, see table 3b. We experimented with randomly cropping the images in

Loss	Task weights			Semantic IoU (%)	Instance Mean Error (px)	Depth Mean Error (px)
	Seg.	Ins.	Depth			
Semantic only	1	0	0	33.3	-	-
Instance only	0	1	0	-	**5.6**	-
Depth only	0	0	1	-	-	0.46
Unweighted sum of losses	0.333	0.333	0.333	31.4	6.5	0.47
Approx. optimal weights	0.89	0.01	0.1	32.6	8.5	0.54
2 task uncertainty weighting	✓	✓		34.3	7.0	-
	✓		✓	**33.7**	-	**0.43**
		✓	✓	-	6.7	0.44
3 task uncertainty weighting	✓	✓	✓	33.5	6.8	0.44

Table 2: Our results for Table 1 in Kendall et al. 2018, using the same hyperparameters and approximate optimal weights as in the original paper.

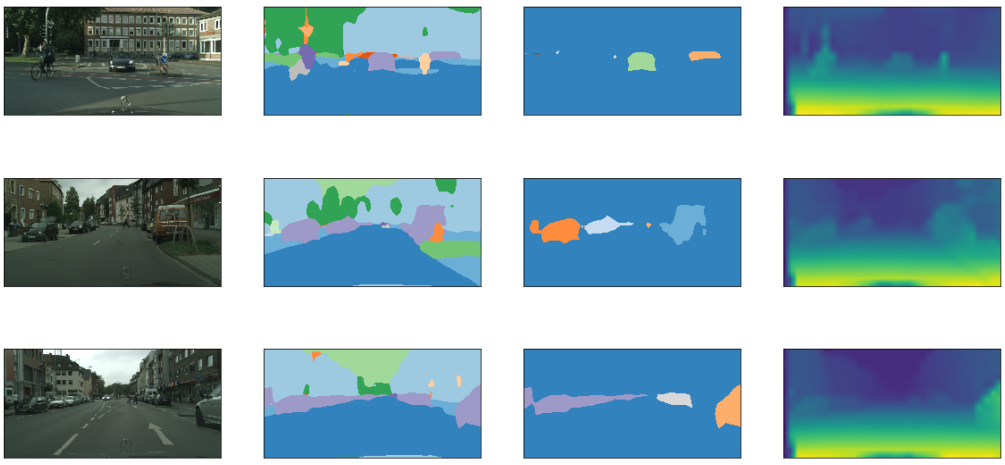

Figure 10: Qualitative outputs of our model on an image in the validation set. From left to right: input image, semantic segmentation, instance segmentation, inverse depth regression.

half, in addition to the random flipping which the authors use in the original paper. This did not improve the results, see table 3d.

We also ran experiments with dropout which is known to reduce overfitting (Srivastava et al. 2014), see table 3e. In experiment 2 we included dropout between the last layer of Resnet and the first layer of the ASPP, and in experiment 3 we included dropout between the final layer of the ASPP and the first layer of the decoder. We decided to add dropout to the encoder rather than the decoders because the encoder is very large compared to the decoders so more likely to suffer from overfitting, and because we observed overfitting in all 3 tasks. Neither experiment improved the results.

IoU breakdown by class I examined the breakdown of the IoU by class, along with the percentage of total pixels which that class makes up, see table 4. There is significant variance in performance on difference classes.

	learning rate	IoU (%)
1	0.01	**33.6**
2	0.001	33.5
3	0.0001	30.5
4	0.00001	24.5

(a) Learning rate search using SGD

	weight decay	IoU (%)
1	0.01	32.6
2	0.001	**37.3**
3	0.0001	30.5

(b) Weight decay search using SGD

	ASPP dilation size	IoU (%)
1	(12, 24, 36)	**33.5**
2	(2, 4, 8)	32.7

(c) Dilation size search

	random crop size	IoU (%)
1	none	**35.1**
2	128x128	33.4

(d) Cropping

	dropout	IoU (%)
1	none	**35.1**
2	after layer 4	32.1
3	after ASPP	32.5

(e) Dropout

Table 3: Results of grid search of various hyperparameters, for experiments with only semantic segmentation enabled. All results are at epoch 180.

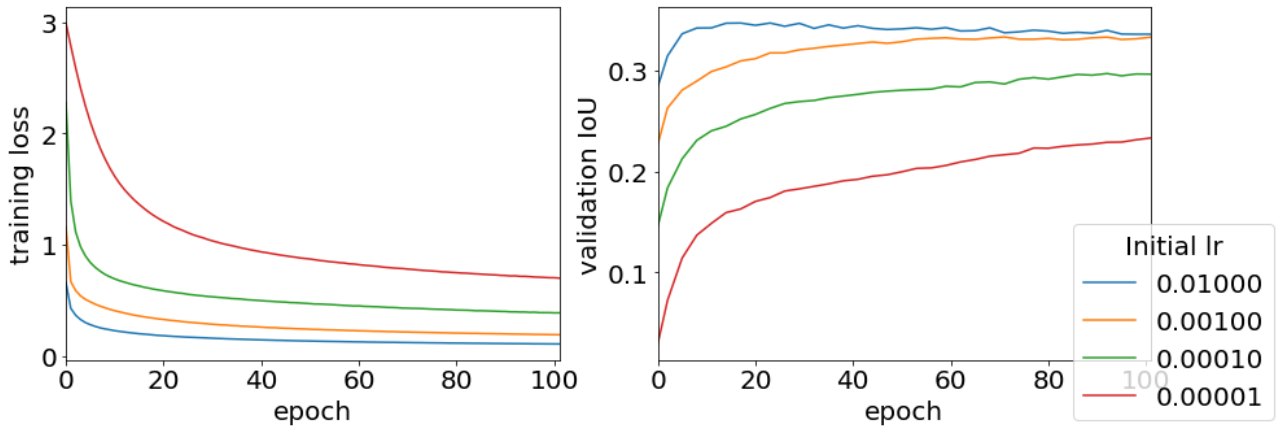

Figure 11: Plot of training loss (left) and IoU on the validation set (right) for various initial learning rates, with only semantic segmentation enabled.

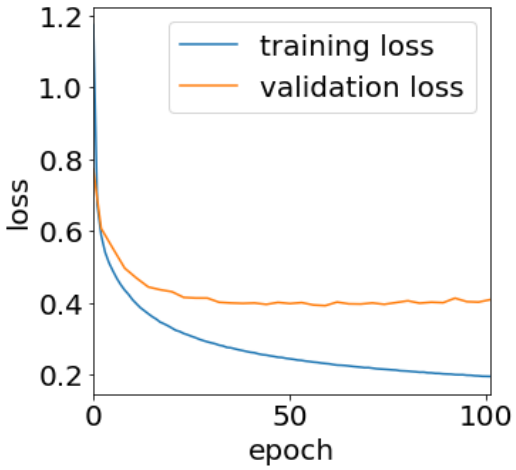

Figure 12: Plot of training and validation loss demonstrating overfitting, for a semantic segmentation only experiment.

class	IoU (%)	total portion of scene (%)	class	IoU (%)	total portion of scene (%)
road	77.8	37.7	sky	75.0	3.3
sidewalk	46.2	5.4	person	47.0	1.3
building	69.7	21.9	rider	15.3	0.2
wall	29.6	0.7	car	70.4	6.5
fence	3.1	0.8	truck	0.0	0.3
pole	6.4	1.5	bus	54.5	0.4
traffic light	6.3	0.2	train	0.0	0.1
traffic sign	17.5	0.7	motorcycle	0.0	0.1
vegetation	71.1	17.3	bicycle	23.9	0.7
terrain	0.3	0.8			

Table 4: IoU on the validation set, broken down by class.

5 Engineering details

We implemented both models using PyTorch (Paszke et al. 2017). We used Sacred (Greff et al. 2017) to store details of our experiments and results.

6 My contributions

We used pair programming for much of the work. I contributed the following as part of a pair:

- Implementation of Fashion MNIST experiments (`multitask-learning/mnist/`)
- Cityscapes learned uncertainty weights loss function (`multitask-learning/losses.py`)
- Cityscapes data loading and pre-processing (`multitask-learning/cityscapes.py`)
- Cityscapes decoders (`multitask-learning/decoders.py`)
- Cityscapes results visualisation (`notebooks/visualize.ipynb`)
- Sacred integration (`multitask-learning/checkpointing.py`)

I individually contributed the following:

- Ran the Fashion MNIST experiments
- Computed of maximum possible IoU (`notebooks/downsample_upsample_test.ipynb`)
- Visualisation for Fashion MNIST results (`notebooks/mnist_results.ipynb`)
- Discovered mistake regarding L1 loss function

7 Evaluation

Our Fashion MNIST results support the hypothesis, but not as strongly as the results in the original paper. The Fashion MNIST task is not as rich as the majority of computer vision tasks, so we cannot conclude from these results that the learned uncertainty weights will perform well on many "real world" tasks. Additionally, we perform a two way comparison between a classification task and a reconstruction task, rather than a classification task and two regression tasks, thus the results are not directly comparable.

Our Cityscapes do not support the hypothesis. However, as we were unable to reproduce the baseline results on semantic segmentation this strongly suggests that there is a bug in our implementation. We know that the baselines results that Kendall et al. 2018 report are possible because other similar models, such as Chen et al. 2017, achieve an IoU of 81.3% on Cityscapes. Thus, our inability to reproduce the results does not cast doubt on the results of Kendall et al. 2018.

We must treat our results with caution because for both experiments we used a loss function with incorrect coefficients for the L1 tasks. It is unclear how much difference this makes. The model can compensate for the different scale of coefficient, however the two loss terms $\frac{1}{\sigma}$ and $\frac{1}{2\sigma^2}$ have different ratios to the regularisation term $\log \sigma$. It would be interesting to run an experiments to compare the different coefficients, perhaps using different coefficients for different types of loss function is not necessary in practice.

With more time, we would have re-run the experiments from table 1 using the optimal hyperparameters we discovered during our grid search. However, it seems unlikely this would have raised the IoU above 40%. Thus we would debug the model further. One option would be to write unit tests. Additionally, we can see from figure 12 that our model is quickly overfitting within 40 epochs. In comparison, the original authors train for 134 epochs. It is possible that a portion of our model is substantially larger than it should be, leading to this overfitting. Finally, 4 shows that the model has substantially worse performance on some classes than others. To improve this we could try using a weighted loss function with larger weights on these classes. During this extra debugging we would create an even lower resolution dataset to reduce our iteration time.

We would improve the original paper by including a precise specification of the model, for example by listing the layers in a table. We had to guess several of the details, and then confirm them by contacting the authors, which took a lot of time.

8 Conclusion

We tested the hypothesis of of Kendall et al. 2018 that, in a multi-task setting, learned loss weightings for each task perform better than fixed loss weightings or running each task individually. Our results on the Fashion MNIST task support the hypothesis, while on the Cityscapes task we were unable to reproduce the baseline results likely due to a bug in our model. Thus, overall we weakly support the hypothesis.

References

Baxter, Jonathan (2000). "A model of inductive bias learning". In: *Journal of artificial intelligence research* 12, pp. 149–198.

Chen, Liang-Chieh et al. (2017). "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *CoRR* abs/1706.05587. arXiv: 1706.05587. URL: http://arxiv.org/abs/1706.05587.

Cordts, Marius et al. (2016). "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Greff, Klaus et al. (2017). "The Sacred Infrastructure for Computational Research". In: *Proceedings of the 16th Python in Science Conference*. Ed. by Katy Huff et al., pp. 49 –56. DOI: 10.25080/shinma-7f4c6e7-008.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Kendall, Alex, Yarin Gal, and Roberto Cipolla (2018). "Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, Chenxi (2018). *DeepLabv3.pytorch*. https://github.com/chenxi116/DeepLabv3.pytorch.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. MIT press.

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In: *NIPS-W*.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Treml, Michael et al. (2016). "Speeding up semantic segmentation for autonomous driving". In: *MLITS, NIPS Workshop*.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (Aug. 28, 2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv: cs.LG/1708.07747 [cs.LG].