Problem Set 1: Segmentation and Uncertainty

Group CV-14

August 19,2022

1 Exercises

Exercises (20 points)

1. What is the loss function for a deterministic segmentation model? What is the connection to standard image classification? (5 points)
2. What is the difference between aleatoric and epistemic uncertainty in deep learning? (5 points)
3. To capture epistemic uncertainty, we can model the segmentation network using principles of Bayesian deep learning, where a probability distribution is maintained over each of the individual network weights. Dropout provides a way to approximate such a Bayesian neural network. (10 points)
    a. How does sampling in the Dropout model during test time allow for epistemic uncertainty estimation? Provide a brief explanation and any equations, if applicable.

Solution.

1. The most commonly used loss function for a deterministic segmentation model is the pixel-level cross entropy loss, which will check each pixel one by one. The prediction results of each pixel category (probability distribution vector) are compared with our single hot coding label vector. Semantic segmentation is a classification at the pixel level. Pixels belonging to the same category must be classified into one category. Therefore, semantic segmentation understands images from the pixel level.It can be regarded as an extension of standard image classification.

2. Uncertainty in data : Also known as Aleatoric uncertainty, it describes the inherent noise in the data, that is, unavoidable errors, which cannot be mitigated by increasing sample data. For example, sometimes the hand of the picture will be slightly trembling blurred, this data can not be eliminated by increasing the number of photographs. Therefore, the method to solve this problem is generally to improve the stability of data collection, or to improve the accuracy of measurement indicators to include various objective influencing factors.

   Model uncertainty : also known as epistemic uncertainty. It points out that the model ' s own estimation of input data may be inaccurate due to poor training, insufficient training data and other reasons, independent of a single data. Therefore, the measurement of cognitive uncertainty is the uncertainty of model parameters estimated by the training process itself. This uncertainty can be mitigated or even addressed through targeted adjustments (such as increasing training data).

Epistemic Uncertainty	Aleatoric Uncertainty
Model uncertainty	Data uncertainty
Describes confidence of the prediction	Describes confidence in the input data
High when missing training data	High when input data is noisy
Can be reduced by adding more data	Cannot be reduced by adding more data

Table 1: The difference between the two uncertainties

3.a. Bayesian models offer a mathematically grounded framework to reason about model uncertainty, but usually come with a prohibitive computational cost. Dropout prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. Dropout can be seen as a way of doing an equally-weighted averaging of exponentially many models with shared weights.

Dropout is a regularization method of neural networks, during training, some neurons are randomly turned on or off to prevent the network from relying on specific neurons.

For the output results of a model, we want to obtain the variance of this result to calculate the model uncertainty (cognitive uncertainty). The model's parameters are fixed, and a single output value cannot obtain a variance. If - we can use the same model to predict the exact sample T times, and the predicted values of these T times are different, we can calculate the variance. We can make the learned model parameters not determined values but obey a distribution so that the model parameters can be sampled from this distribution. The model parameters are different for each sampling, so the model results are also different to achieve the purpose.

To make the model parameters not determined but obey a distribution. Dropout satisfies this requirement. When using dropout to train the model, the parameters of the model can be regarded as obeying a Bernoulli distribution (such as dropout radio = 0.5, one is that half of the neurons in this layer will be a dropout, in other words - each neuron in this layer has a probability of 0.5 dropout, which is Bernoulli distribution). But we estimate the model uncertainty must be in the trained model, that is, when testing the model. Therefore, we only need to open dropout when predicting, predict TT times, and take the average value of prediction as the final prediction value. And the variance can be obtained through the average value to get the uncertainty of deep learning. This method is also called MC Dropout Bayesian neural network.

For the regression problem, uncertainty can be expressed by the variance of output value :

$$\frac{1}{T}\sum_{t=1}^{T}\left(f^{\mathrm{W}^{\wedge}\mathrm{t}}(\mathrm{x})-\mathrm{E}(\mathrm{y})\right)^{2} \tag{1}$$

where $E(y)=\frac{1}{T}\sum_{t=1}^{T}f^{\mathrm{W}^{\wedge}\mathrm{t}}$ represents the average output.

For classification problems, the probability of T prediction is :

$$p(y=c\mid x,X,Y)\approx\frac{1}{T}\sum_{t=1}^{T}\mathrm{Softmax}\left(f^{\mathrm{W}^{\wedge}\mathrm{t}}(\mathrm{x})\right) \tag{2}$$

where $\mathrm{W}^{\wedge}\mathrm{t}\sim q_{\theta}^{*}(W)$ the model parameter for each sampling. Uncertainty can be measured by entropy:

$$H(p)=-\sum_{c=1}^{C}p_{c}\log p_{c} \tag{3}$$

2 Software Lab

Since the problem we are facing is image semantic segmentation, and we expect to achieve end-to-end (that is, we expect the terminal to directly obtain the semantic segmentation results after inputting RGB images), we hope to select an acceptable model based on what we have learned in the course, so We tried U-NeT.The structure and its parameters are showed below:

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_1 (InputLayer)           [(None, 128, 256, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 128, 256, 64  1792        ['input_1[0][0]']
                                )

 batch_normalization (BatchNorm  (None, 128, 256, 64  256        ['conv2d[0][0]']
 alization)                     )

 conv2d_1 (Conv2D)              (None, 128, 256, 64  36928       ['batch_normalization[0][0]']
                                )

 batch_normalization_1 (BatchNo  (None, 128, 256, 64  256        ['conv2d_1[0][0]']
 rmalization)                   )

 max_pooling2d (MaxPooling2D)   (None, 64, 128, 64)  0           ['batch_normalization_1[0][0]']

 conv2d_2 (Conv2D)              (None, 64, 128, 128  73856       ['max_pooling2d[0][0]']
                                )

 dropout (Dropout)              (None, 64, 128, 128  0           ['conv2d_2[0][0]']
                                )

 batch_normalization_2 (BatchNo  (None, 64, 128, 128  512        ['dropout[0][0]']
 rmalization)                   )

 conv2d_3 (Conv2D)              (None, 64, 128, 128  147584      ['batch_normalization_2[0][0]']
                                )

 batch_normalization_3 (BatchNo  (None, 64, 128, 128  512        ['conv2d_3[0][0]']
 rmalization)                   )

 max_pooling2d_1 (MaxPooling2D)  (None, 32, 64, 128)  0          ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 32, 64, 256)  295168      ['max_pooling2d_1[0][0]']

 dropout_1 (Dropout)            (None, 32, 64, 256)  0           ['conv2d_4[0][0]']

 batch_normalization_4 (BatchNo  (None, 32, 64, 256)  1024       ['dropout_1[0][0]']
 rmalization)

 conv2d_5 (Conv2D)              (None, 32, 64, 256)  590080      ['batch_normalization_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 32, 64, 256)  1024       ['conv2d_5[0][0]']
 rmalization)

 max_pooling2d_2 (MaxPooling2D)  (None, 16, 32, 256)  0          ['batch_normalization_5[0][0]']

 conv2d_6 (Conv2D)              (None, 16, 32, 512)  1180160     ['max_pooling2d_2[0][0]']

 batch_normalization_6 (BatchNo  (None, 16, 32, 512)  2048       ['conv2d_6[0][0]']
 rmalization)

 conv2d_7 (Conv2D)              (None, 16, 32, 512)  2359808     ['batch_normalization_6[0][0]']

 batch_normalization_7 (BatchNo  (None, 16, 32, 512)  2048       ['conv2d_7[0][0]']
```

```
rmalization)

max_pooling2d_3 (MaxPooling2D)  (None, 8, 16, 512)  0           ['batch_normalization_7[0][0]']

conv2d_8 (Conv2D)               (None, 8, 16, 1024)  4719616     ['max_pooling2d_3[0][0]']

batch_normalization_8 (BatchNo  (None, 8, 16, 1024)  4096        ['conv2d_8[0][0]']
rmalization)

conv2d_9 (Conv2D)               (None, 8, 16, 1024)  9438208     ['batch_normalization_8[0][0]']

batch_normalization_9 (BatchNo  (None, 8, 16, 1024)  4096        ['conv2d_9[0][0]']
rmalization)

conv2d_transpose (Conv2DTransp  (None, 16, 32, 512)  2097664     ['batch_normalization_9[0][0]']
ose)

batch_normalization_10 (BatchN  (None, 16, 32, 512)  2048        ['conv2d_transpose[0][0]']
ormalization)

tf.concat (TFOpLambda)          (None, 16, 32, 1024  0           ['batch_normalization_7[0][0]',
                                )                                 'batch_normalization_10[0][0]']

conv2d_10 (Conv2D)              (None, 16, 32, 512)  4719104     ['tf.concat[0][0]']

batch_normalization_11 (BatchN  (None, 16, 32, 512)  2048        ['conv2d_10[0][0]']
ormalization)

conv2d_11 (Conv2D)              (None, 16, 32, 512)  2359808     ['batch_normalization_11[0][0]']

batch_normalization_12 (BatchN  (None, 16, 32, 512)  2048        ['conv2d_11[0][0]']
ormalization)

conv2d_transpose_1 (Conv2DTran  (None, 32, 64, 256)  524544      ['batch_normalization_12[0][0]']
spose)

batch_normalization_13 (BatchN  (None, 32, 64, 256)  1024        ['conv2d_transpose_1[0][0]']
ormalization)

tf.concat_1 (TFOpLambda)        (None, 32, 64, 512)  0           ['batch_normalization_5[0][0]',
                                                                  'batch_normalization_13[0][0]']

conv2d_12 (Conv2D)              (None, 32, 64, 256)  1179904     ['tf.concat_1[0][0]']

dropout_2 (Dropout)             (None, 32, 64, 256)  0           ['conv2d_12[0][0]']

batch_normalization_14 (BatchN  (None, 32, 64, 256)  1024        ['dropout_2[0][0]']
ormalization)

conv2d_13 (Conv2D)              (None, 32, 64, 256)  590080      ['batch_normalization_14[0][0]']

batch_normalization_15 (BatchN  (None, 32, 64, 256)  1024        ['conv2d_13[0][0]']
ormalization)

conv2d_transpose_2 (Conv2DTran  (None, 64, 128, 128  131200      ['batch_normalization_15[0][0]']
spose)                          )

batch_normalization_16 (BatchN  (None, 64, 128, 128  512         ['conv2d_transpose_2[0][0]']
ormalization)                   )
```

```
tf.concat_2 (TFOpLambda)       (None, 64, 128, 256  0           ['batch_normalization_3[0][0]',
                               )                                 'batch_normalization_16[0][0]']

conv2d_14 (Conv2D)             (None, 64, 128, 128  295040      ['tf.concat_2[0][0]']
                               )

batch_normalization_17 (BatchN  (None, 64, 128, 128  512        ['conv2d_14[0][0]']
ormalization)                  )

conv2d_15 (Conv2D)             (None, 64, 128, 128  147584      ['batch_normalization_17[0][0]']
                               )

batch_normalization_18 (BatchN  (None, 64, 128, 128  512        ['conv2d_15[0][0]']
ormalization)                  )

conv2d_transpose_3 (Conv2DTran  (None, 128, 256, 64  32832      ['batch_normalization_18[0][0]']
spose)                         )

batch_normalization_19 (BatchN  (None, 128, 256, 64  256        ['conv2d_transpose_3[0][0]']
ormalization)                  )

tf.concat_3 (TFOpLambda)       (None, 128, 256, 12  0           ['batch_normalization_1[0][0]',
                               8)                                'batch_normalization_19[0][0]']

conv2d_16 (Conv2D)             (None, 128, 256, 64  73792       ['tf.concat_3[0][0]']
                               )

dropout_3 (Dropout)            (None, 128, 256, 64  0           ['conv2d_16[0][0]']
                               )

batch_normalization_20 (BatchN  (None, 128, 256, 64  256        ['dropout_3[0][0]']
ormalization)                  )

conv2d_17 (Conv2D)             (None, 128, 256, 64  36928       ['batch_normalization_20[0][0]']
                               )

batch_normalization_21 (BatchN  (None, 128, 256, 64  256        ['conv2d_17[0][0]']
ormalization)                  )

conv2d_18 (Conv2D)             (None, 128, 256, 34  2210        ['batch_normalization_21[0][0]']
                               )

==================================================================================================
Total params: 31,061,282
Trainable params: 31,047,586
Non-trainable params: 13,696
--------------------------------------------------------------------------------------------------
```

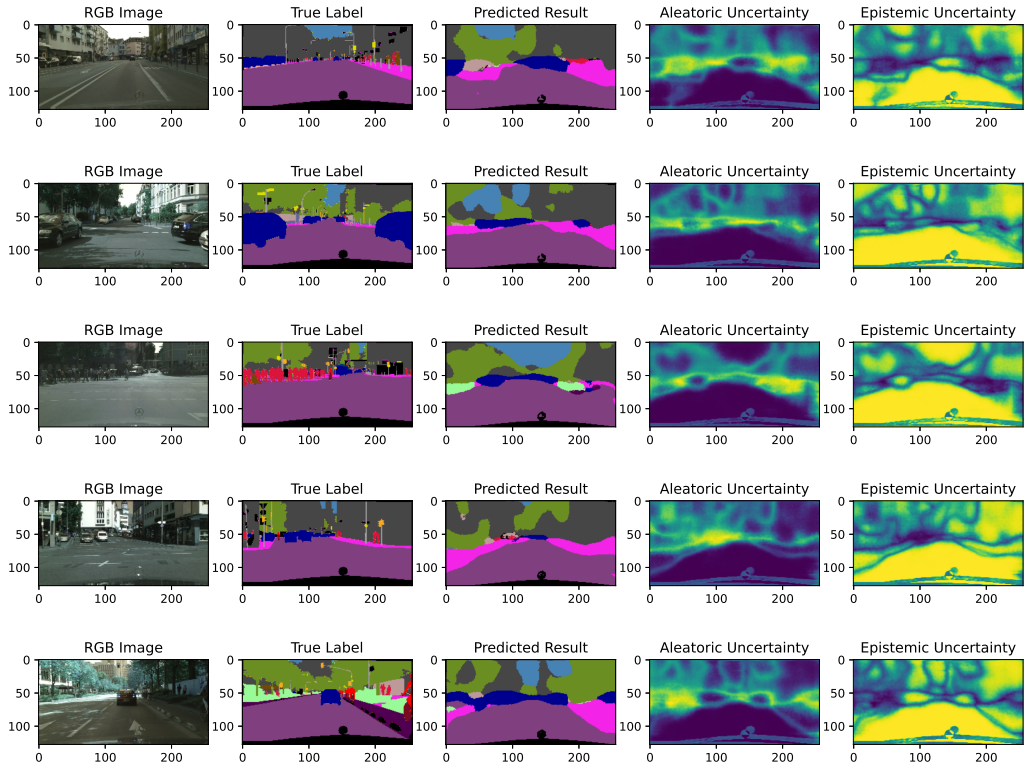

Figure 1: Final Result

About the loss function,Since there are as many as 34 categories involved in semantic segmentation, numerical encoding is more suitable for this problem. Therefore, we choose sparse categorical crossentropy as the loss function we use. Then we adopt the Adam optimization algorithm, which is simple to implement, computationally efficient, requires less memory, and can naturally implement the step-size annealing process (automatically adjust the learning rate). In addition, it is very suitable for large-scale data and parameter scenarios, and is suitable for problems with sparse gradients or very noisy gradients. Since we use a more traditional U-NET model, the training time is long, and we may refer to advanced research results to improve the structure of U-NET. In addition, to enhance the robustness of the model, we can introduce a Bayesian layer based on the original model.