RBE 474X
Project 3

Colin Balfour
Worcester Polytechnic Institute
Worcester, Massachussetts 01609

Khang Luu
Worcester Polytechnic Institute
Worcester, Massachussetts 01609

Thinh Nguyen
Worcester Polytechnic Institute
Worcester, Massachussetts 01609

I. Part 1 - Patch Augmentation

We used TorchVision's built in transformation operations to perform patch augmentation, including:

- Perspective transform
- Scaling and rotation
- Color jitter

These transformations are compiled with pytorch's Compose method to create a pipeline of transformations that are applied to the dataset.

A. Perspective transform, scaling, and rotation

Perspective transform is done through TorchVision's RandomPerspective method. Scaling, rotations, and translations are done through RandomAffine. The parameters for these transformations are:

- RandomPerspective: distortion scale = 0.5, p = 0.8
- RandomAffinne: rotation degrees = 20, scale = (0.35, 0.45), translate = (0.2, 0.05)

B. Color jitter

Color jitter is done through TorchVision's ColorJitter method. The parameters are:

- Brightness variability: 0.05
- Contrast variability: 0.1
- Saturation variability: 0.1
- Hue variability: 0.1

II. Part 2 - Patch training and validation

Training is done on the augmented dataset.

The model is trained for as many epoch as it can go. The optimizer used is Adam with a learning rate of 1e-2.with a batch size of 128

- Batch size: 128
- Learning rate: 1e-2

A. Loss function

Loss function used for training is the Disparity loss and total variation (TV) loss. The disparity loss is used to measure the difference between the predicted disparity map and the ground truth disparity map. In this case, loss is only calculated in the region of the augmented patch, and the ground truth is defined as a hyperparameter of target patch depth.

The TV loss is used to measure the smoothness of the disparity map.

In our case, the coefficient of the disparity loss is 1, and the coefficient of the TV loss is 2.5.

B. Results

Below are some results of the generated dataset:

III. Part 3

For this part, we are performing instance segmentation to find the regions of the image/video that contain the window.

A. Network architecture

We used a modified unet for instance segmentation. The output had 4 classes, which assumed that there were 3 windows and 1 background. The output of the model was permuted so that the order of the classes didn't matter, since we wanted to be agnostic to which class was which window.

The architecture diagram for this is depicted in figure 6.

B. Implementation

The model was trained on the dataset generated in Part 1. The hyperparameters used for training are the same as in Part 2, and is shown in Table 1. Loss function used is depicted below.

C. Loss function

For instance segmentation, we trained our network on three different loss functions, each with different goals in mind. Initially, we tried Cross Entropy.

Cross Entropy is a common loss function for classification tasks, and is used to measure the difference between two probability distributions. In this case, it is used to measure the difference between the predicted mask and the ground truth mask.

However, this loss function does not take into account the spatial information of the image, and is not ideal for instance segmentation.

Next, we tried Dice Loss. Dice Loss is a loss function that is used to measure the similarity between two samples. It is used in this case to measure the similarity between the predicted mask and the ground truth mask.

Dice Loss is better suited for instance segmentation than Cross Entropy, as it is more robust to class imbalance, which in this case is due to more single-instance images. Finally, we tried a combination of Cross Entropy and Dice Loss. The combined loss function is a weighted sum of the two loss functions.

D. Results

For instance segmentation, the failure cases were similar to the semantic model, but also included cases where the model couldn't distinguish between the individual instances too well, and it found the "loophole" of just predicting the same class for all instances, or predicting a mixture of two classes which ends up being better on average.

Results for the model are in the vdeo "part3.mp4". Below are the loss curve per epoch plot:

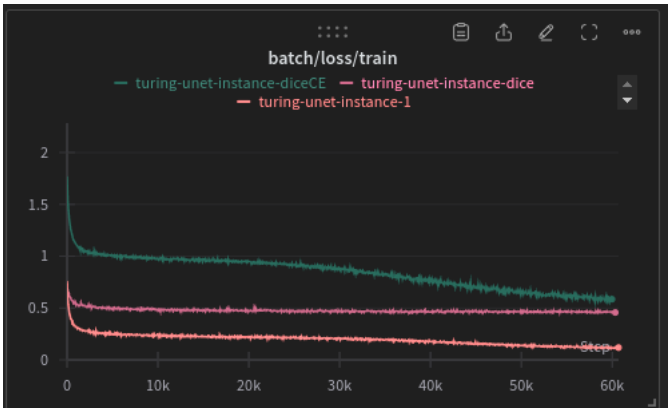

Fig. 1. Training loss curve, batch

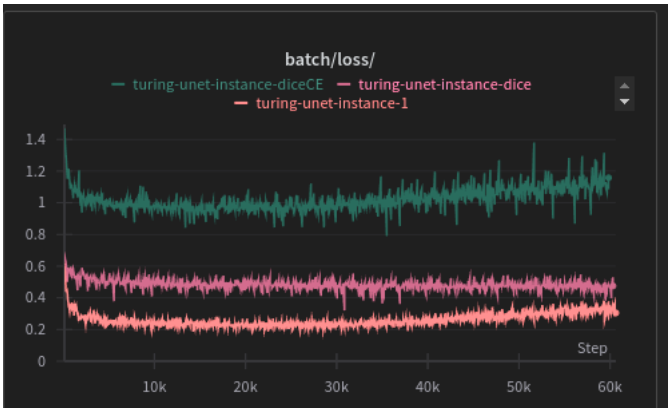

Fig. 2. Evaluate loss curve, batch

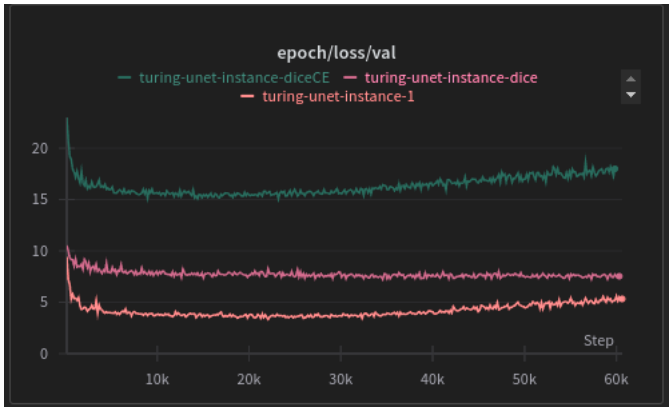

Fig. 3. Evaluate loss curve, epoch