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. Each run that we have run is tuned differently on hyperparameters.

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

1) First run: For our first run, we used:

- Batch size: 256
- Learning rate: 1e-2
- Target disparity: 70

This is ran until epoch 779. Below is the loss plot:

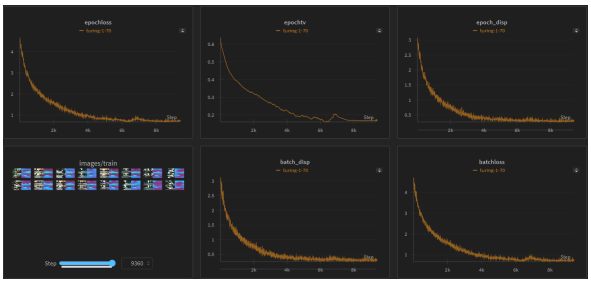

Fig. 1. Plot for first run

2) Second run: For our second run, we used:

- Batch size: 128
- Learning rate: 1e-1
- Target disparity: 70

This is ran until epoch 929. Below is the loss plot:

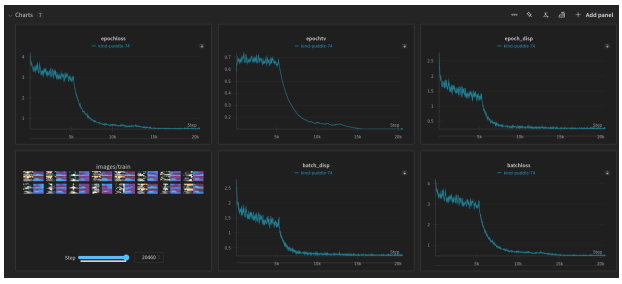

Fig. 2. Plot for second run

3) Third run: For our third run, we used:

- Batch size: 128
- Learning rate: 1e-2
- Target disparity: 10

This is ran until epoch 970. Below is the loss plot:

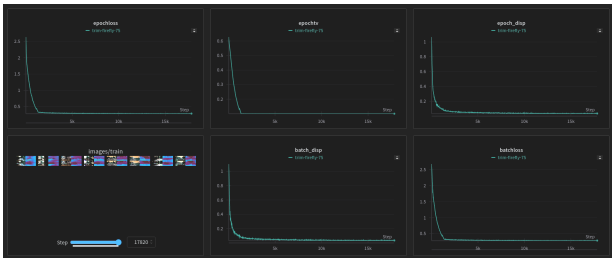

Fig. 3. Plot for third run

III. Part 3 - Real world testing

For a control run, we held up an empty piece of paper to the camera, and the model was able to predict the disparity map of the paper. The disparity map is shown in the file "control.mp4".

1) First run: For our first run, we used:

- Batch size: 256
- Learning rate: 1e-2
- Target disparity: 70

The example video for this is shown in the file Run1.mp4 and Run1-2.mp4

2) Second run: For our second run, we used:

- Batch size: 128
- Learning rate: 1e-1
- Target disparity: 70

The example video for this is shown in the file Run2.mp4 and Run2-2.mp4

3) Third run: For our third run, we used:

- Batch size: 128
- Learning rate: 1e-2
- Target disparity: 10

The example video for this is shown in the file Run3.mp4 and Run3-2.mp4

IV. Fast Gradient Sign Method

For this part, we will talk about how fast gradient sign method (FGSM) can be used to attacking depth networks.

FGSM uses the gradients of the neural network to generate adversariality. For an image, it uses the gradients of the loss to create a new image that maximises the loss. This new image is called the adversarial image. The goal of this image generation is to cause wrong predictions. Despite its simplicity, it effectively highlights the vulnerability of deep networks to small, targeted perturbations.

A. FGSM Attack Process on Deep Networks

The FGSM attack process on deep networks is as follows:

When applied to a deep network, such as those used for image recognition or natural language processing, FGSM creates adversarial examples that appear almost identical to the original inputs but lead to incorrect predictions. The strength of the adversarial attack is governed by the epsilon parameter:

A small epsilon produces smaller perturbation, making it harder for humans to notice the change, but it will be harder to fool the model. A larger epsilon will cause the perturbation to become more visible, reducing the stealthiness of the attack.

B. FGSM Attack, adversarial patches

In contrast to pixel-level perturbations across the entire image, as in traditional FGSM, we could also keep the perturbation within a confined region of the image: the patch. The patch is initialized with random values. In traning, FGSM is used to modify the patch:

Patch Optimization: Use the FGSM method to update the patch's pixel values by computing the gradient of the loss with respect to the patch region, not the entire image. The perturbation is applied only to the patch while keeping the rest of the image unchanged.