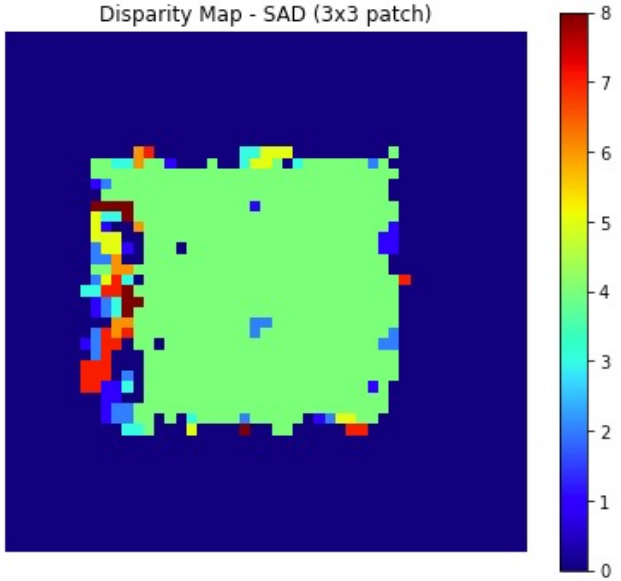
CS 6476 Project 4

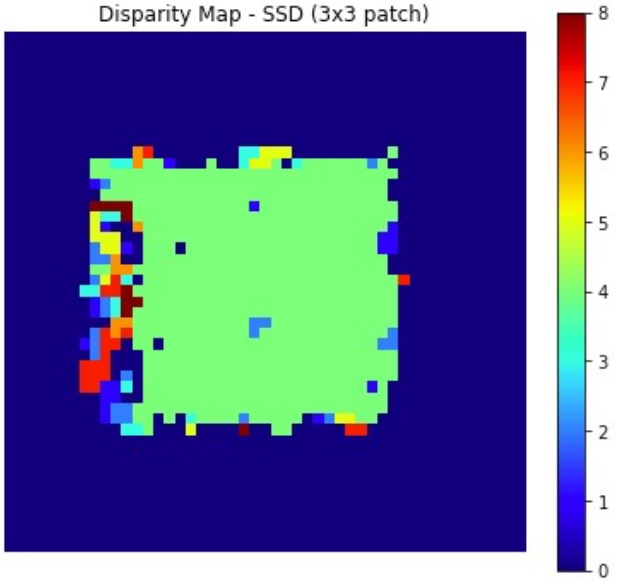
Haoran Wang
Haoran.wang@gatech.edu
hwang827
903543952

Part 1: Simple stereo by matching patches

[insert visualization of random dot stereogram disparity map using 3x3 blocks with SAD]

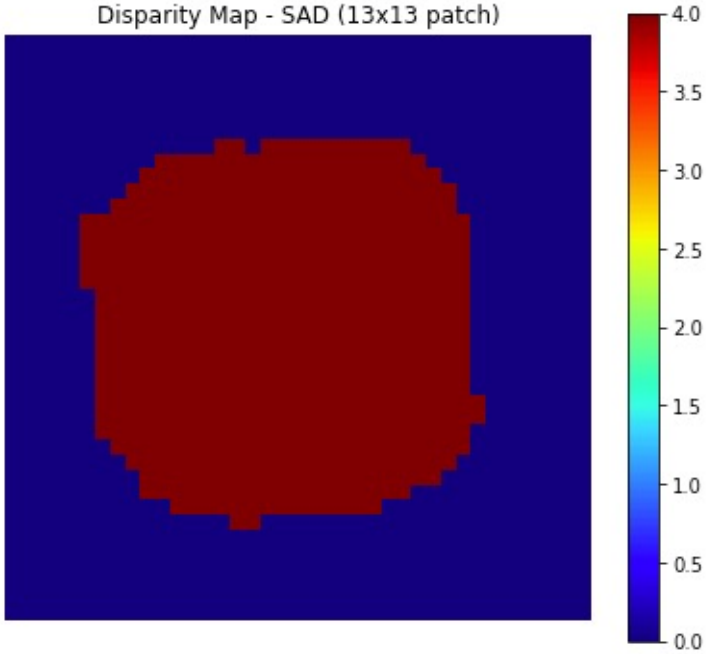


[insert visualization of random dot stereogram disparity map using 3x3 blocks with SSD]

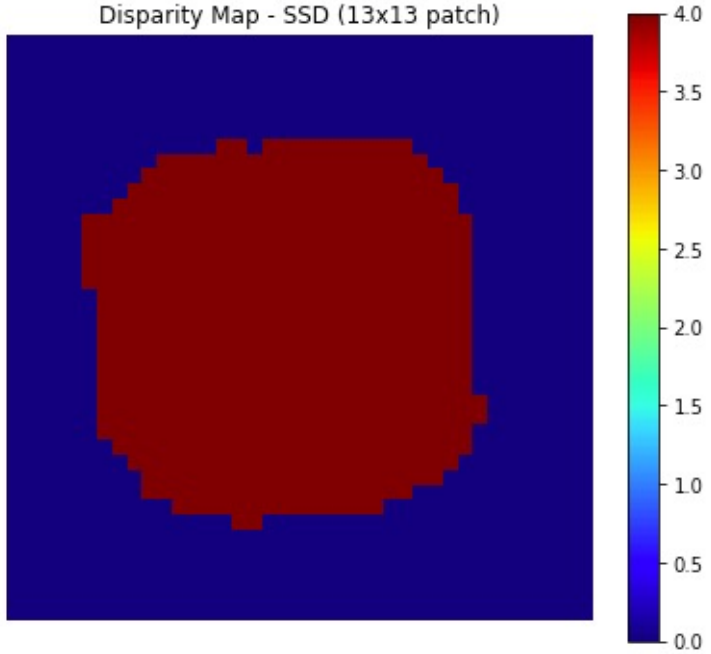


Part 1: Simple stereo by matching patches

[insert visualization of random dot stereogram disparity map using 13x13 blocks with SAD]



[insert visualization of random dot stereogram disparity map using 13x13 blocks with SSD]



Part 1: Simple stereo by matching patches

[What is the effect of increasing the block size for computing the disparity maps?]

Increasing the block size decreases the granularity of calculation, the disparity of neighbor pixels will be close, and there are less sudden change of disparity in the disparity maps, which makes the disparity maps more smooth.

[For the disparity maps, why are the results poor on the left edge but not other edges?]

Because on the left side, it’s possible that the objects are visible to the left camera but out of view of the right camera, or too distant to be valid.

Part 1: Simple stereo by matching patches

[What kind of regions will produce convex error profiles?]

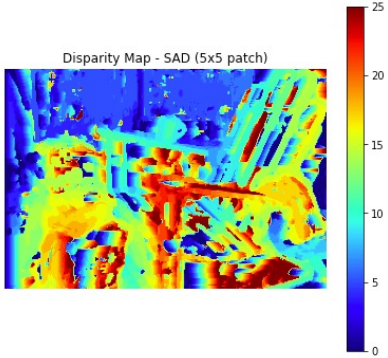
Regions that are not very uniform, for example with sharp color changes.

[What kind of regions will produce non-convex error profiles?]

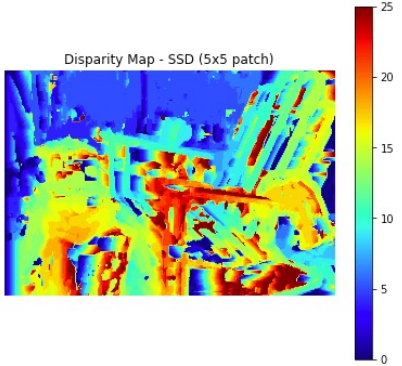
Regions that are quite uniform, for example with only shade change of colors.

Part 1: Simple stereo by matching patches

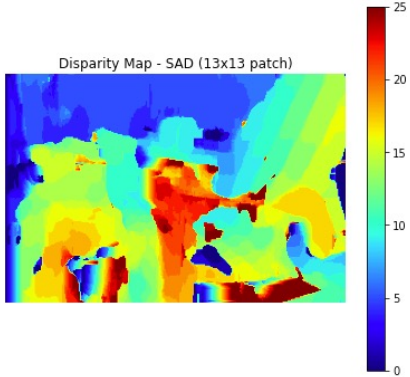
[insert visualization of set 1 (Adirondack) disparity map using 5x5 blocks with SAD]



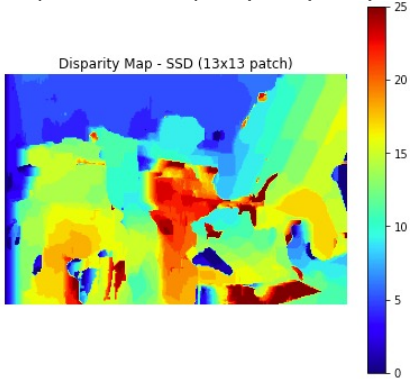
[insert visualization of set 1 (Adirondack) disparity map using 5x5 blocks with SSD]



[insert visualization of set 1 (Adirondack) disparity map using 13x13 blocks with SAD]

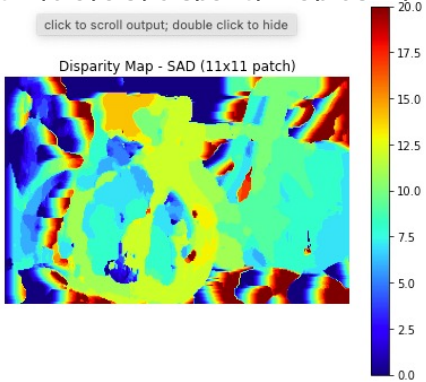


[insert visualization of set 1 (Adirondack) disparity map using 13x13 blocks with SSD]

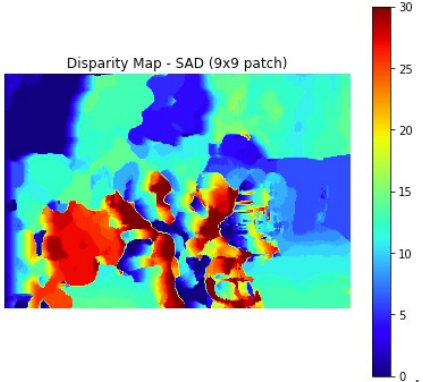


Part 1: Simple stereo by matching patches

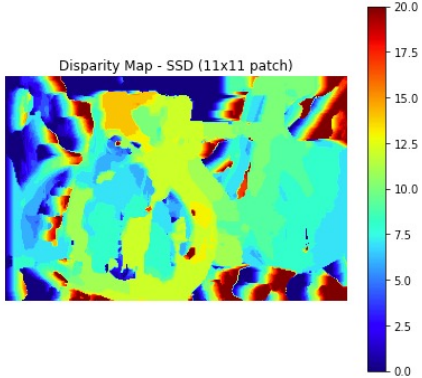
[insert visualization of set 2 (bicycle) disparity map using 11x11 blocks with SAD]



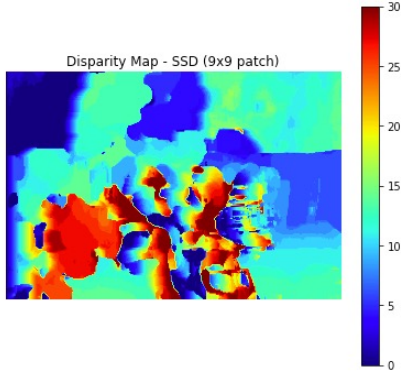
[insert visualization of set 5 (flowers) disparity map using 9x9 blocks with SAD]



[insert visualization of set 2 (bicycle) disparity map using 11x11 blocks with SSD]



[insert visualization of set 5 (flowers) disparity map using 9x9 blocks with SSD]



Part 1: Simple stereo by matching patches

[Explain your implementation of calculate_disparity_map (don’t just paste code).]

I loop through patch by patch. For each patch, I get the corresponding patch disparity value in the right graph with smallest similarity error by search through each small patches within the max horizontal distance, calculating the similarity between the small patch from the left graph with them and keeping track of the offset with the smallest similarity.

[What are some qualitative ways that the results look poor when compared to the ground truth?]

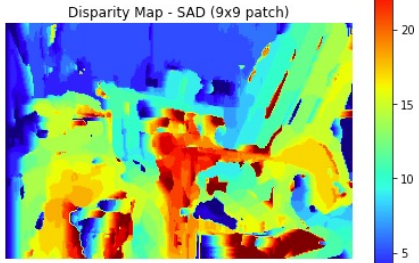
When the disparity window is too small or too large, the results will look poor.

[Why are we not able to get better results with this method? Where does it fail?]

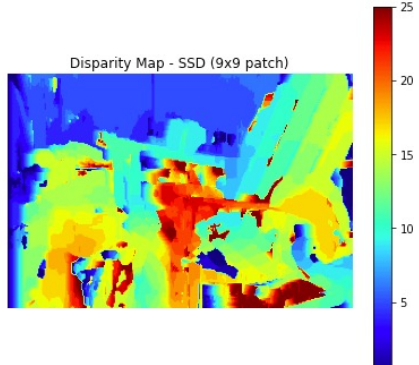
Too small a window might not be able to distinguish unique features of an image, but too large a window would mean many patches would likely have many more things in common, leading to less helpful matches.

Part 1: Simple stereo by matching patches

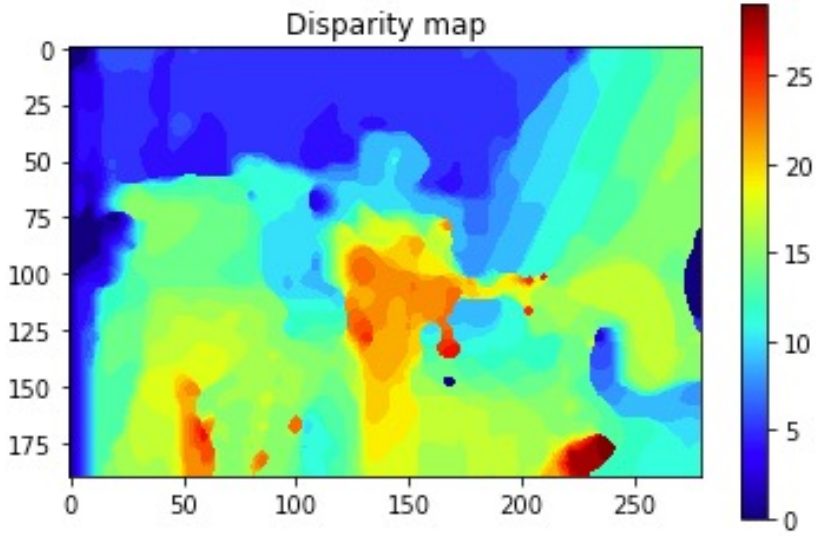
[insert visualization of set 1 (Adirondack) disparity map using 9x9 blocks with SAD (without smoothing)]



[insert visualization of set 1 (Adirondack) disparity map using 9x9 blocks with SSD (without smoothing)]



[insert visualization of set 1 (Adirondack) disparity map using 9x9 blocks with SAD after smoothing]



Part 1: Simple stereo by matching patches

[What regions of the image does smoothing seem to perform better on? Why do you think so?]

Smoothing performs best on background/uniform areas, disparity values that are already similar to each other, so smoothing doesn't have to make any drastic changes. Pixels near each other are supposed to have similar disparity values, so the technique makes sense in these regions. Smoothing can also help in areas of occlusion, where SAD may not be able to find any suitable match. Smoothing will curb the problems by pushing the disparity of a pixel to be similar to that of its neighbors. This is because the cost volume is the error profile of the horizontal patches and the energy function is penalizing disparity based on the error profile or the surrounded area within the default max disparity value, it will penalize more if the disparities of the neighbors are large.

[What regions of the image does smoothing seem to perform worse on? Why do you think so?]

Smoothing performs poorly in areas of fine detail, with narrow objects. It also performs poorly in areas with many edges and depth discontinuities. This is because SGM is penalizing based on the error profile of the neighbor pixels within the default max disparity value, if the region is larger than the max disparity value, SGM will penalize less, since it thinks there is not much difference between pixels.

Part 1: Simple stereo by matching patches

[Describe how semi-global matching works.]

SGM works by penalizing jumps in disparity between adjacent pixels by adding a penalty term on top of the matching cost term. It calculates the pixel-wise dissimilarity cost at pixel with disparity and the regularization cost between the pixel and all pairs of its neighboring pixels to perform the penalizing.

[Would smoothing still work for images with both horizontal and vertical shifts?]

It won’t, at least it will have bad performance on images with both horizontal and vertical shifts.

Part 2: Learning-based stereo matching

[Copy-paste your MCNET architecture as printed out in part2_disparity.ipynb]

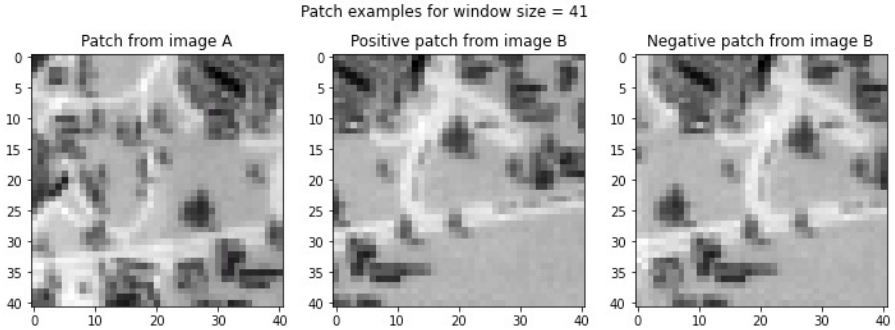
```
MCNET(
  (conv): Sequential(
    (0): Conv2d(1, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(112, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): Conv2d(112, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU()
    (6): Conv2d(112, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): Conv2d(112, 112, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU()
  )
  (classifier): Sequential(
    (0): Linear(in_features=27104, out_features=384, bias=True)
    (1): ReLU()
    (2): Linear(in_features=384, out_features=384, bias=True)
    (3): ReLU()
    (4): Linear(in_features=384, out_features=1, bias=True)
    (5): Sigmoid()
  )
  (criterion): BCELoss()
)
```

[What does the ReLU activation function do? Why do we use it?]

ReLUs output the input directly if it is positive, otherwise, it will output zero, since negative number doesn't have meaning in our case. Therefore, ReLU improve neural networks by speeding up training. The gradient computation is very simple (either 0 or 1 depending on the sign of x x). Also, the computational step of a ReLU is easy: any negative elements are set to 0.0 -- no exponentials, no multiplication or division operations.

Part 2: Learning-based stereo matching

[insert visualization of a patch with window size 41 from image A + positive patch from image B + negative patch from image B]



[Given a true disparity map for each stereo pair, how do we extract positive and negative patches for training?]

At each image position where the true disparity is known we extract one negative and one positive training example. This ensures that the data set contains an equal number of positive and negative examples. A positive example is a pair of patches, one from the left and one from the right image, whose center pixels are the images of the same 3D point, while a negative example is a pair of patches where this is not the case.

Part 2: Learning-based stereo matching

[Train MCNET to achieve the best stereo evaluation. Insert a plot of the train and validation losses of your best network.]

[Insert a plot of the train and validation accuracies of your best network.]

Part 2: Learning-based stereo matching

[How does using a large learning rate value (e.g., 1) vs. a small value (e.g., 1e-5) affect training? Why?]

A learning rate that is too large, for example 1, can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small, for example 1e-5, can cause the process to get stuck. The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.

[What is the effect of varying the window size on performance? Do you think there is an optimal window size for all images? Why or why not?]

The error due to perspective distortion becomes larger and the match accuracy decreases as the size of the window grows. Since pixels in the two correlation windows are increasingly misaligned, even when their center pixels are perfectly aligned. However, smaller size of the window could cause more matching errors as well, each of which potentially can have very large spike error as an outlier. There is not an optimal window size for all images. Since each images are taken from different angles with different perspective, angle, lightning and so forth. But we can tune for the best window size for each pair.

Part 2: Learning-based stereo matching

[insert plot of matching cost comparison with SSD, SAD, and MC-CNN]

[Is MC-CNN or SAD/SSD better?]

MC-CNN is better.

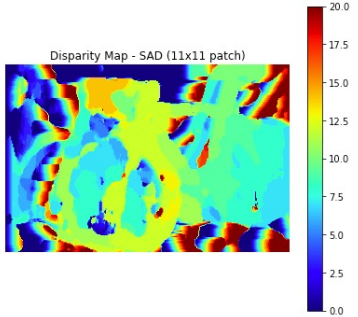
Part 2: Learning-based stereo matching

[What techniques are discussed in the MC-CNN paper to augment the size of the training set?]
They randomly rotate, scale and shear the training patches; they also change their brightness and contrast.
The steps are:

- Rotate the left patch by rotate degrees and the right patch by rotate + rotate diff degrees.
- Scale the left patch by scale and the right patch by scale · scale diff.
- Scale the left patch in the horizontal direction by horizontal scale and the right patch
- by horizontal scale · horizontal scale diff.
- Shear the left patch in the horizontal direction by horizontal shear and the right patch
- by horizontal shear + horizontal shear diff.
- Translate the right patch in the vertical direction by vertical disparity.
- Adjust the brightness and contrast by setting the left and right image patches to:

  PL ← PL · contrast + brightness and
  PR ← PR · (contrast · contrast diff) + (brightness + brightness diff), with addition and multiplication carried out element-wise where appropriate.

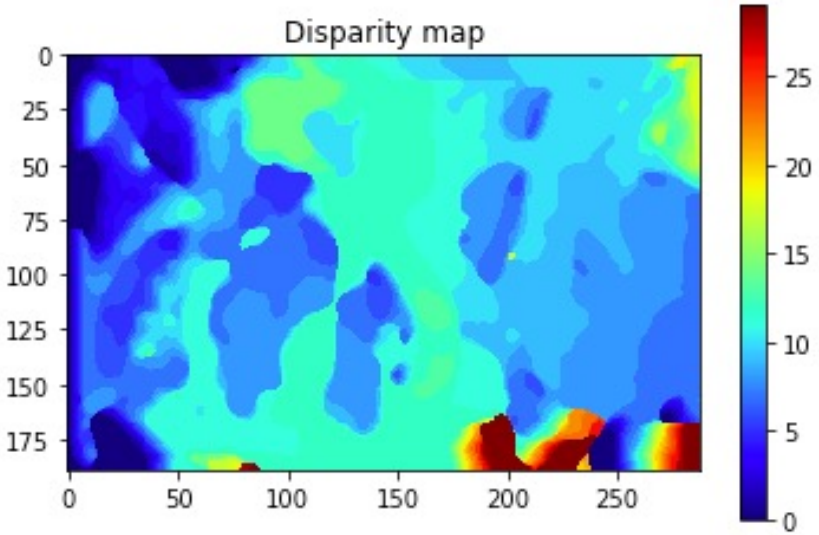
[insert visualization of the bicycle disparity map using 11x11 blocks with SAD]



[insert visualization of the bicycle disparity map using 11x11 blocks with MC-CNN]

Part 2: Learning-based stereo matching

[insert visualization of the bicycle disparity map using 11x11 blocks with SAD+SGM]



[insert visualization of the bicycle disparity map using 11x11 blocks with MC-CNN+SGM]