Deep Rectangling – Computer Vision Laboratory

Kareem Jabareen - 211406343

Malik Egbarya - 318585627

Introduction:

In this work, we are going over a recent Deep Learning based solution for fixing the shape of stitched images. The solution presented has good results (as it appears in their paper linked to this work GitHub repository). However, the developers of this solution chose less than six thousand images among sixty thousand images from the original test dataset, which means that the solution works only on specific images and is hard to generalize.

Our goal:

We are trying to generalize the solution presented above. To achieve that, we need to read the paper discussing the work and understand the structure of the solution models.

Idea 1 - Modify hyper-parameters:

We have tried to lower the learning rate to avoid overfitting situation but that did not seem to work, The previous owners of this project probably have tested many values for the hyper-parameter in general and the learning rate.

Idea 2 – Add more training data:

We have tried to add more training data and it appeared to be helpful. The real problem was that we could not find enough data of stitched images which led us to our third idea.

Idea 3 – Generate training data:

This time we chose to generate new dataset from the current data, For each image in the training dataset we are creating several augmentations in several ways to enrich our dataset and train the model on them too.

We have tried different ways of creating augmentations:

- Blurring image.
- Changing image brightness.
- Flipping image from left to right.
- Cropping image.

Blurring image:

We have used the gaussian blur 5x5 filter. This method did not seem to improve the model much, so our conclusion was to drop it due to its heavy processing time along with the time complexity needed to run the script.

Changing image brightness:

In this method, we are choose to change the brightness of the input image and its ground truth by the same factor, and run the training code on them. However, this method also hadn't help the model much, So we dropped it too.

Flipping image from left to right:

In this method, we choose to flip the input image, its ground truth and the mask from left to right. This creates a new bundle that ready to enter it as batch in to training model with no extra modifications needed. This method seemed to be very helpful, so we kept it, despite its long computation time.

Cropping image:

Among all the methods we have mention earlier, this method is the most complicated, useful, and flexible one. Its idea tend to be simple, for each image, we are choosing several rectangle crops in the ground truth image, and then mapping them to the relevant window in the input image (not necessarily a rectangle), and then we create the right mask using the mesh we have, finally we insert it as new batch to the training model.

For our model, we chose to create 4 crop augmentations per image, the first two are created by cutting the image in the middle horizontally, the other two are created by cutting the image in the middle vertically.

We have thought about cutting the image to four quarters or more little rectangles but that would take the script ages to finish execution.

Workflow:

We have modified the utils.py file to generate the augmentations discussed above. In addition, we have modified the constant.py file to run it on our machine.

Furthermore, we have created a new file called make_mesh.py which is responsible of calculating the mesh for each image in the training dataset.

We added a lot of functions, aside with there documentations to help users understand how the program works, and maybe they improve it with new ideas .

In addition, we fixed some bugs and improved the time complexity for several functions, got rid of the legacy code, we did a huge refactor, and added useful comments.

Note: The augmentations we created do not get saved on disk, which makes the augmentations thing perfect. Instead of saving the augmentations on disk which might take a lot of storage locally, we are creating augmentations in each training loop, processing them, then deleting it. This way we significantly optimize the code to run on less memory and achieve better benchmarks.

Results:

After the train file finished execution, it created our new model. To use the new model, we had to modify the inference.py file. We ran the inference file and got the following results a bit lower SSIM and PSNR values from the ones we got after running the pretrained model of the first solution developers.

Here are the values we got:

PSNR: 19.54

SSIM: 0.531

Here are some samples of our model test results:

However, when running our model on a new test image, most of the times it performs better than the other model. That is because the first model is specific to some types of images and is inaccurate when it comes to general data.

Note: The results of testing our model can be found under the directory “our_final_rectangling”.

Conclusion:

After completing this work, we can say we have learned some great points:

- Data augmentations can decrease the model accuracy. In our example, although the new model gives better generalization and it fails in minor parts, it still got lower accuracy than the first pre-trained model.

- Geometric augmentations had greater influence on the model learning which shows that creating augmentations should be wise and relevant to the problem we are trying to solve.

- It is advised always to test the model on greater and relevant datasets, and to not drop data from the test dataset to higher the numeric results. A deep learning model should be as general as possible since input data is unpredictable.