

DSAC Exercise

Titus Leistner
Eric Brachmann

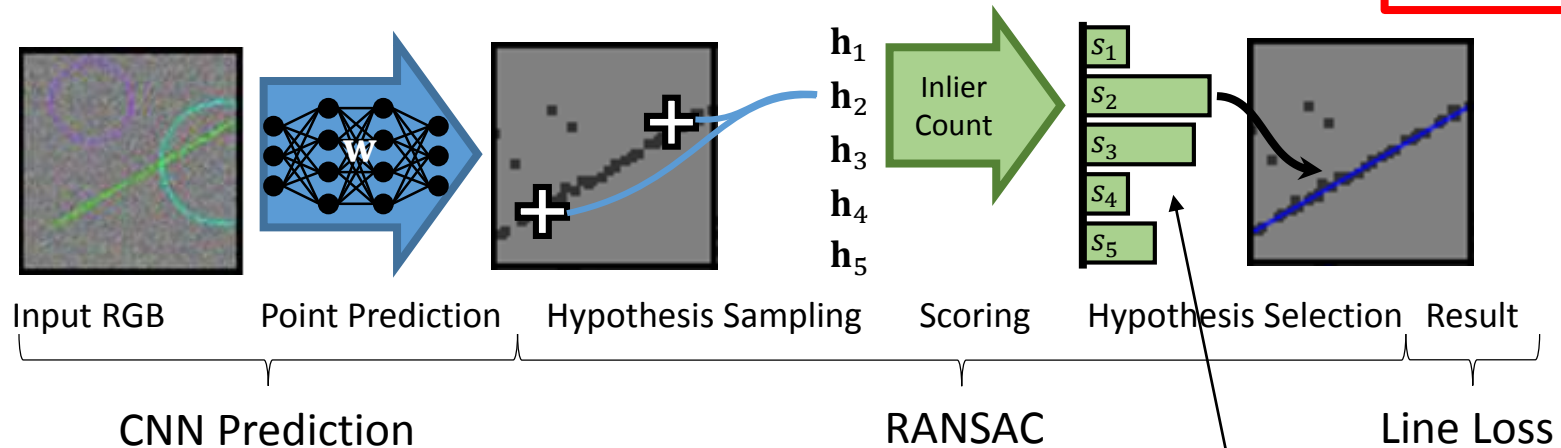


UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386



Fully Differentiable Pipeline

REMINDER



Differentiability needed for:

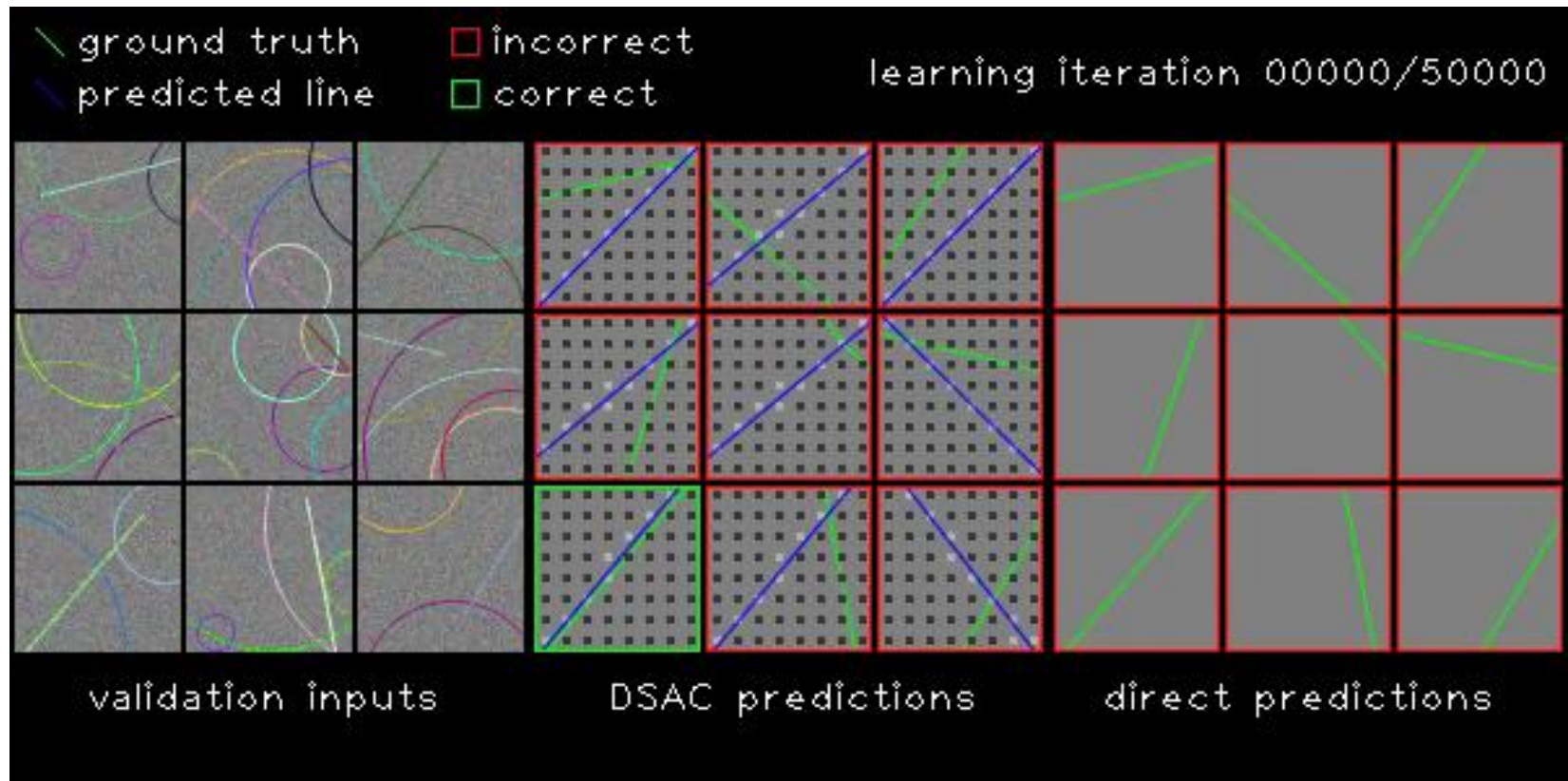
- CNN Prediction (standard backprob)
- Hypothesis sampling:
 - Uniform sampling
 - Line parameter estimation
- Scoring
- Hypothesis selection:
 - Argmax
 - Soft Argmax
 - Probabilistic Selection
- Line Loss Function

The code uses an additional refinement step: re-fitting the line to the (soft) inliers. This is just an additional function in the chain of derivatives.

DSAC Exercise

In this exercise, you will adapt the line fitting pipeline presented in the lecture to **fit circles**, instead.

- You have access to the full, working line fitting example
- You also get the code frame for circle fitting but essential methods are still blank
- The code requires Python and PyTorch, a GPU is not required



- PyTorch is a deep learning framework accessible from python
- It is easy to install and easy to use
 - Installation: <https://pytorch.org/>
 - Documentation: <https://pytorch.org/docs/stable/index.html>
 - Tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
 - An example of deep learning in PyTorch:

```
cnn = SomeCNNClass()
optimizer = optim.Adam(cnn.parameters())
```

repeat

```
prediction = model(input)
result = process(prediction)
loss = loss_function(result, gt)
```

```
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

Will be given in the code.

Forward pass.

Do something with the prediction, e.g. fit a line or a circle to the prediction. PyTorch will automatically keep track of everything you do.

Calculate the error of the result. **Note:** For DSAC we instead calculate the expected loss.

Calculate all gradients. PyTorch will do this for you.

Update CNN parameters.

Reset gradient buffers.

Tasks

1. Install PyTorch und run the line fitting example (~100 iterations). Plot the training losses. **(2pt)**
2. Implement the generation of synthetic circle images. **(1pt)**
3. Implement the circle loss function. The direct CNN prediction should work now. Run the code (~100 iterations) and plot the training loss. **(2pt)**
4. Implement DSAC circle fitting.
 1. Implement hypothesis sampling. **(1pt)**
 2. Implement soft inlier count. **(2pt)**
 3. Implement refinement with hard inliers **(2pt)**
 4. **Bonus:** Implement refinement with soft inliers **(+1pt)**
5. Run everything, plot losses, play with parameters, look at images **(priceless)**

Bonus task: DSAC and direct prediction perform similarly well in this setup. Try to adapt the synthetic image generation to cater to the advantages of either method, i.e. make the task easier or more difficult in certain ways. **(incredible fame)**

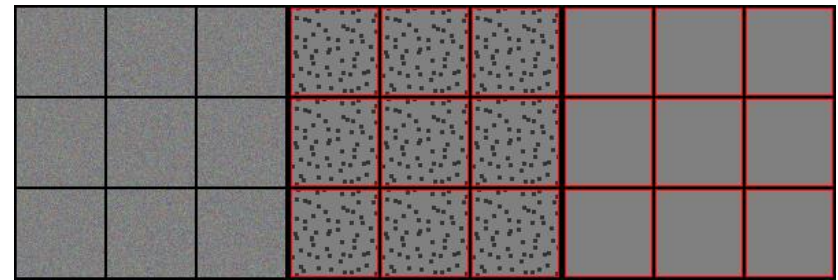
Tasks

1. Install PyTorch und run the line fitting example (~100 iterations). Plot the training losses. **(2pt)**
 - You will find the line fitting code in your exercise materials, but you can also find it online for a nicer formatting of the readme.md:
<https://github.com/titus-leistner/dsac-students>
 - The code generates a log*.txt file with the training losses per iteration.
 - Column 1: iteration
 - Column 2: expected loss of DSAC
 - Column 3: loss of DSAC winning hypothesis
 - Column 4: loss of direct prediction
 - Plot column 3 (resp. column 4) vs. column 1. You may use any plotting tool, but we recommend matplotlib within a JuPyter notebook. GnuPlot will also do.
 - Since the training loss is noisy, we recommend using a running average to smooth the training curve. This can e.g. be done with the Numpy `convolve` function.

Tasks

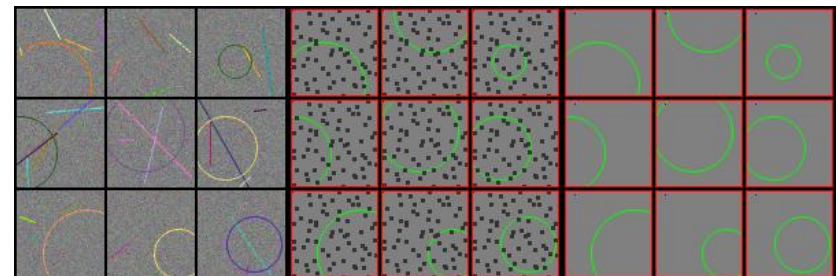
2. Implement the generation of synthetic circle images. (1pt)

- Go to the circle fitting code, run it via `python main.py`
 - The code will create a new folder for output images
 - In the beginning, these will look like this:



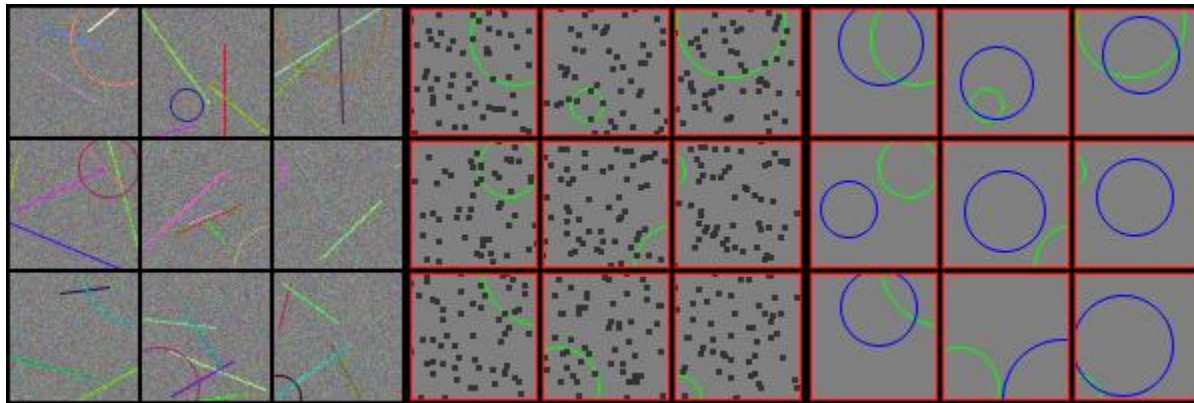
- Fill in the blank method `sample_circles` in `circle_dataset.py`. More details are given in the code.

- The output should then look like this:
- **Note** that the colored circles on the left (input images) match the green circles on the right (ground truth labels).



Tasks

- Implement the circle loss function. The direct CNN prediction should work now. Run the code (~100 iterations) and plot the training loss. **(2pt)**
- Fill in the blank method `__call__` in `circle_loss.py`. More details are given in the code.
 - The training loss should clearly decrease within a few iterations. The output could look like this after e.g. 100 iterations. (Still not very good of course)



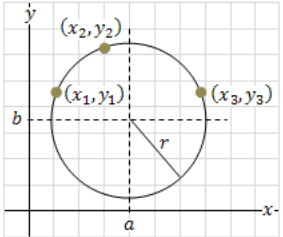
- Note** that in the beginning, most boxes should be marked red (i.e. incorrect). In case your boxes appear green, you likely did not scale the loss by the image size.

Tasks

4. Implement DSAC circle fitting.

1. Implement hypothesis sampling. (1pt)

- Fill in the blank method `__sample_hyp` in `dsac.py`.
More details are given in the code.
- Left is a description of how to fit a circle to 3 points.
- Note** that PyTorch offers many handy functions like determinants of matrices.
- Note** that your circle should interpolate the 3 points perfectly. In later stages, you should always have at least 3 inliers.

| Equation of a circle passing through 3 points (x_1, y_1) (x_2, y_2) and (x_3, y_3) . | |
|---|--|
|  | <p>The equation of the circle is described by the equation:</p> $Ax^2 + Ay^2 + Bx + Cy + D = 0$ <p>After substituting the three given points which lies on the circle we get the set of equations that can be described by the determinant:</p> $\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0$ |
| <p>The coefficients A, B, C and D can be found by solving the following determinants:</p> $A = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad B = - \begin{vmatrix} x_1^2 + y_1^2 & y_1 & 1 \\ x_2^2 + y_2^2 & y_2 & 1 \\ x_3^2 + y_3^2 & y_3 & 1 \end{vmatrix} \quad C = \begin{vmatrix} x_1^2 + y_1^2 & x_1 & 1 \\ x_2^2 + y_2^2 & x_2 & 1 \\ x_3^2 + y_3^2 & x_3 & 1 \end{vmatrix} \quad D = - \begin{vmatrix} x_1^2 + y_1^2 & x_1 & y_1 \\ x_2^2 + y_2^2 & x_2 & y_2 \\ x_3^2 + y_3^2 & x_3 & y_3 \end{vmatrix}$ <p>The values of A, B, C and D will be after solving the determinants:</p> $A = x_1(y_2 - y_3) - y_1(x_2 - x_3) + x_2y_3 - x_3y_2$ $B = (x_1^2 + y_1^2)(y_3 - y_2) + (x_2^2 + y_2^2)(y_1 - y_3) + (x_3^2 + y_3^2)(y_2 - y_1)$ $C = (x_1^2 + y_1^2)(x_2 - x_3) + (x_2^2 + y_2^2)(x_3 - x_1) + (x_3^2 + y_3^2)(x_1 - x_2)$ $D = (x_1^2 + y_1^2)(x_3y_2 - x_2y_3) + (x_2^2 + y_2^2)(x_1y_3 - x_3y_1) + (x_3^2 + y_3^2)(x_2y_1 - x_1y_2)$ | |
| <p>Center point (x, y) and the radius of a circle passing through 3 points (x_1, y_1) (x_2, y_2) and (x_3, y_3) are:</p> $x = \frac{(x_1^2 + y_1^2)(y_2 - y_3) + (x_2^2 + y_2^2)(y_3 - y_1) + (x_3^2 + y_3^2)(y_1 - y_2)}{2(x_1(y_2 - y_3) - y_1(x_2 - x_3) + x_2y_3 - x_3y_2)} = -\frac{B}{2A}$ $y = \frac{(x_1^2 + y_1^2)(x_3 - x_2) + (x_2^2 + y_2^2)(x_1 - x_3) + (x_3^2 + y_3^2)(x_2 - x_1)}{2(x_1(y_2 - y_3) - y_1(x_2 - x_3) + x_2y_3 - x_3y_2)} = -\frac{C}{2A}$ $r = \sqrt{(x - x_1)^2 + (y - y_1)^2} = \sqrt{\frac{B^2 + C^2 - 4AD}{4A^2}}$ | |

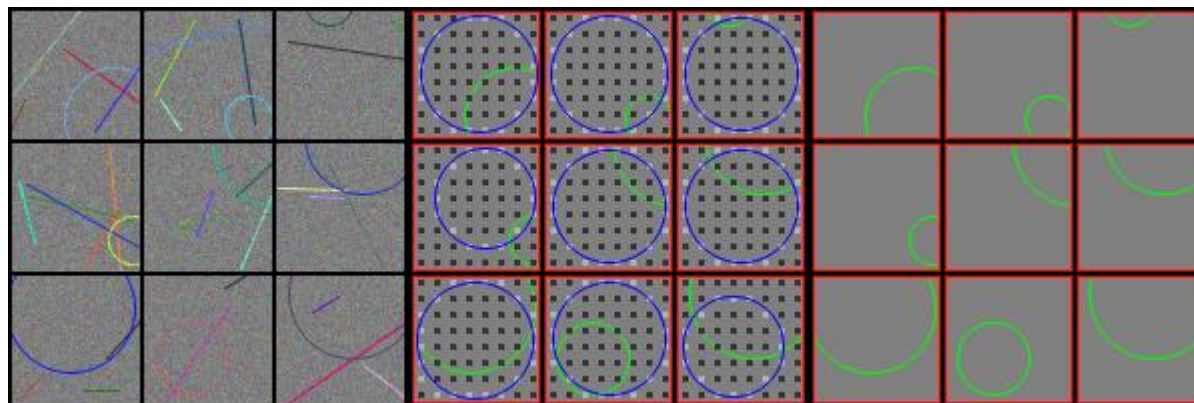
[Source: <http://www.ambrsoft.com/trigocalc/circle3d.htm>]

Tasks

4. Implement DSAC circle fitting.

2. Implement soft inlier count. (2pt)

- Fill in the blank method `__soft_inlier_count` in `dsac.py`. More details are given in the code.
- Calculate the distance of each predicted point to the circle, and turn distance into soft inliers by applying a sigmoid function.
- Look at the line fitting code for reference.
- The code should be running now. Below you see the first image generated. **Note** how DSAC chooses circles which almost fill the image (middle), because they have most inliers (bright points).

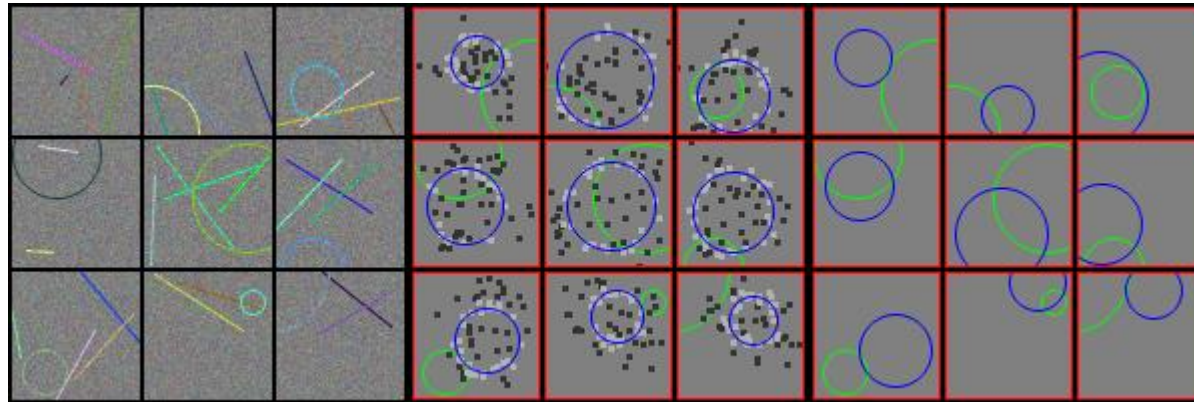


Tasks

4. Implement DSAC circle fitting.

3. Implement refinement with hard inliers (**2pt**)

- Fill in the blank method `__refine_hyp` in `dsac.py`. More details are given in the code.
- A description of a least-squares circle fit can be found in `materials/circle_fit.pdf` (source: https://dtcenter.org/met/users/docs/write_ups/circle_fit.pdf)
- Fit the circle to all soft inliers above a threshold (e.g. 0.5).



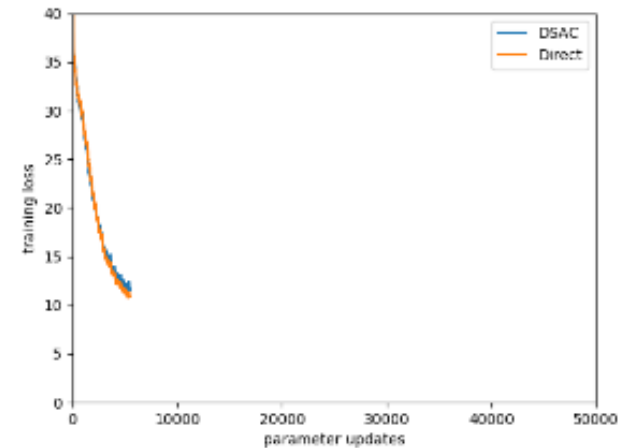
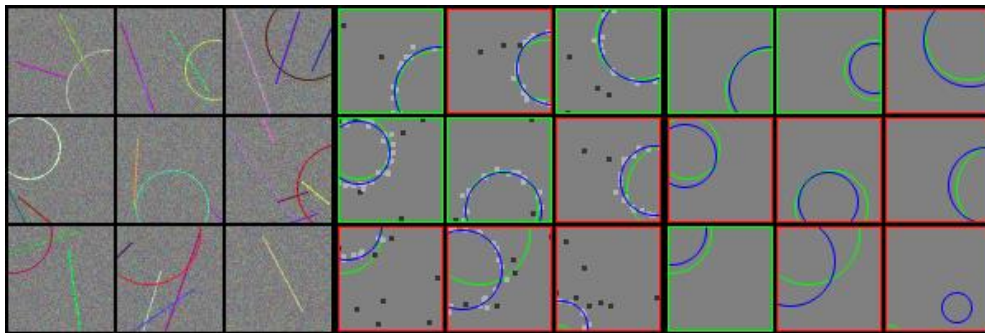
Note that the refined circles goes through the white inlier points.

- **Bonus task:** The refinement method can be adapted to fit the circle to all soft inliers instead, using the soft inlier score as weight in the appropriate places.

Tasks

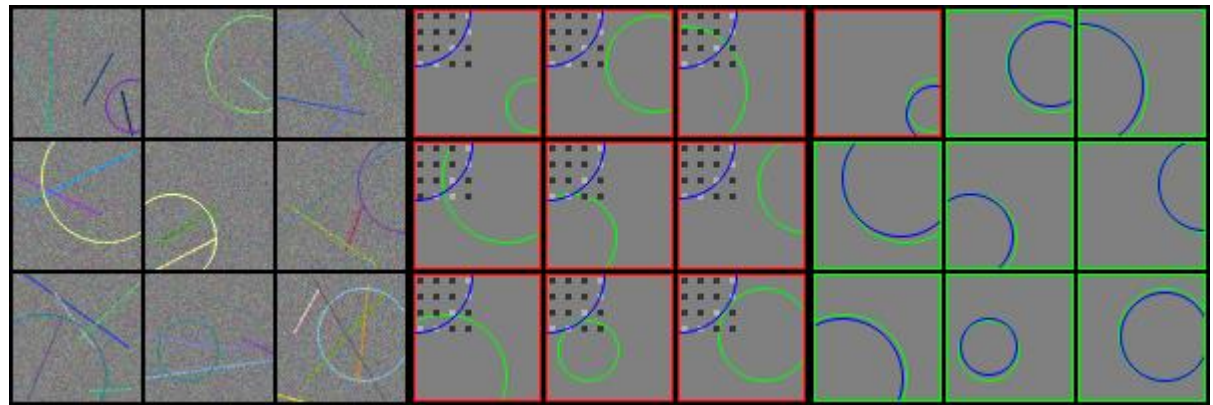
5. Run everything, plot losses, play with parameters, look at images (**priceless**)

- The output after 5000 iterations will look something like this (runs a while, you do not need to run it that long):



PyTorch Pittfall

- PyTorch calculates all gradients for you.
- However, some functions have discontinuities, and gradients might become `nan` or `inf` without PyTorch telling you (in the current version)
- This will send your network in a non-working state:



- For example, `torch.sqrt(0)` will return an `inf` gradient during backprop. When multiplied by 0 this gives `nan`.
 - Workaround: `torch.sqrt(expression + 0.00001)`

Code on HeiBox

Have Fun!