- How I did it
    - To complete the jitter process of this lab, I created two new methods and used np.roll and scipy.
    - For my data structures, I maintained the same overall data structure as MNIST BLUE with a large list filled with numpy arrays for each line in the MNIST train/set file.
    - In my which_jitter(img) method, I first resized my img to be (28x28). I used a list filled with all the different types of jitter possibilities (["normal", "up", "left", "down", "right", "rotate_right", "rotate_left"]) and called np.random.choice() on it to randomly figure out which to use. I also had a dictionary that used the strings from the list of jitter possibilities as keys and had values that corresponded to how the image would be changed. I used np.roll to shift the images up/left/down/right and I varied the axes to do so. I used scipy.ndimage and the rotate function to rotate the images a certain number of degrees. Finally, I resized the matrix back to its original shape(1,784) and continued the code using these distortions.
    - I created the crop_size() method that ensured that the rotated matrix did not add/remove pixels or values to the matrix that would make the shapes/dimension incompatible.
- Was it better?
    - There were more misclassified for the MNIST training set, however; there were fewer misclassified for the MNIST test set than the non-distorted single network.
-

```
Architecture: [784, 300, 10]
Misclassified for training: 2.0433333333333334 %
Misclassified for testing: 2.22 %
Number of Epochs: 12
```