Instagram Engineering Follow
Feb 27, 2012 · 5 min read

Instagram Engineering Challenge: The Unshredder

In our office, we have a pretty amazing paper shredder. Seriously, the thing shreds just about anything. It even has a special slot for credit cards (why anyone would want to regularly shred credit cards is beyond me, but I digress…).

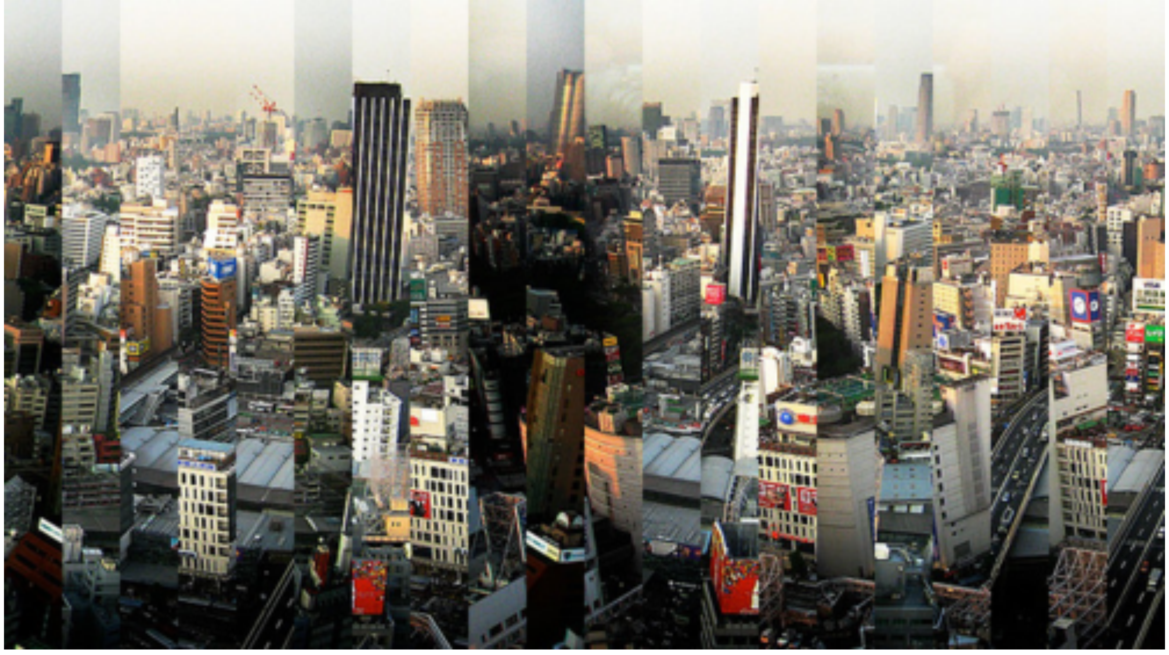
One day, after shredding some paper, I thought to myself: shredding paper is a pretty insecure way of destroying important stuff. I figured, it's a small set of shreds that are all relatively uniform in width and could be pieced back together algorithmically in a fraction of a second.

So, I sat down and though about what approach I'd use to piece the document back together. It's unlike a regular puzzle in that all the pieces are exactly the same size, so you can't rely upon the spatial domain to solve piecing shreds together. However, if you think about it, there's a pretty simple approach that would allow you to find matches in a different domain. That is, imagine you're sitting there trying to find a match between two pieces. What are you looking for to decide whether they're a fit or not?

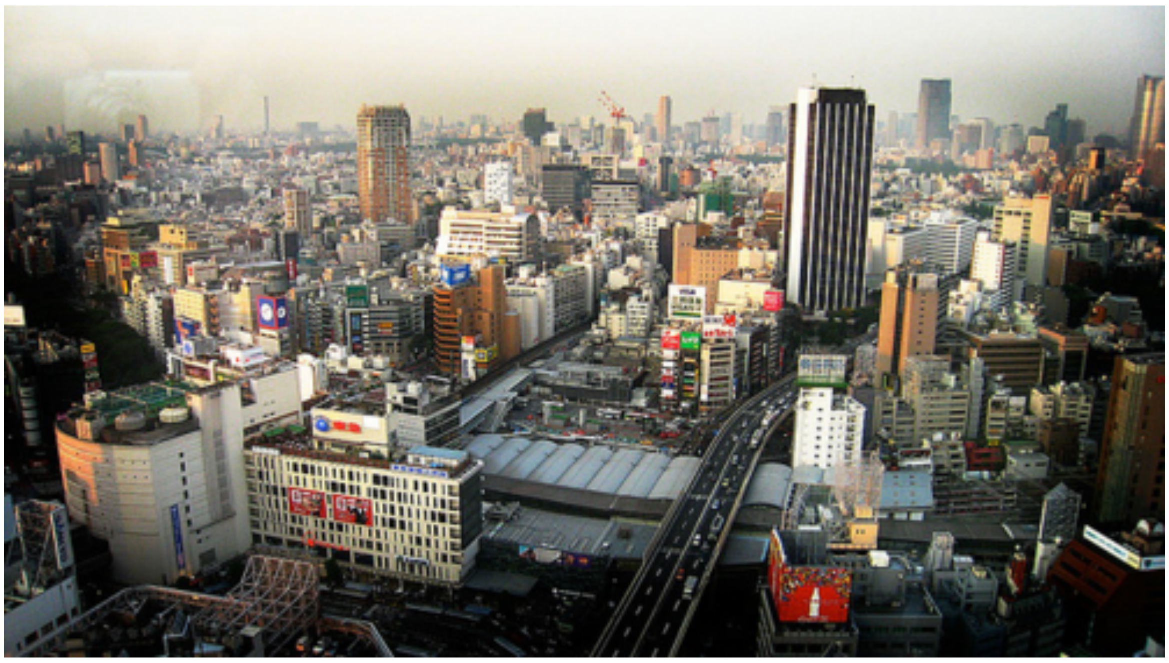
Anyway, we got really excited about writing a script to take in an image of shreds of paper and piece them back into an original document. It's an interesting challenge that marries image processing with an interesting algorithmic challenge as well.

The Challenge

Your challenge, if you choose to accept it, is to write a simple script that takes a shredded image in as input:



and outputs an unshredded and reconstituted image. That is, imagine if you took an image, divided it into an even number of columns and shuffled those columns randomly to produce a shredded image. Then, take that image into the script and output the original image:



We tackled this, and our solution took a few hours plus another few hours for the bonus challenge (more on that later).

The reward

Due to overwhelming response, we've run out of our entire stock of tee-shirts! With future challenges we'll be offering a reward for the first group of people who respond.

Guidelines

1. Choose a scripting language of your choice. We chose Python for its relative ease prototyping and availability of the Python Imaging Library (PIL) that allowed us to do the image stuff we wanted to do. You can easily use something like C++ or Ruby for this as well.

2. Produce a script that reads in a shredded image (like the one below) and produces the original image. For this image, you can assume shreds are 32 pixels wide and uniformly spaced across the image horizontally. These shreds are scattered at random and if re-arranged, will yield the original image.

3. Your solution should algorithmically unshred the image. This means it should work on arbitrarily shredded images we feed your script that are shredded in the same manner.

4. BONUS CHALLENGE: We went the extra mile and made our script even spiffier by auto-detecting how wide the uniform strips are. Extra bonus points to anyone who works this into their solution. But first, we'd recommend getting your script to work assuming 32 pixel-wide shreds. For this you can assume shreds will never end up next to each other correctly in the source image.

5. The key to this problem is being able to access pixel data in the image. We used Python Imaging Library—PIL (http://www.pythonware.com/products/pil/) which made it very easy to parse. See the PIL tips below. If you're using Ruby, check out RMagick (http://rmagick.rubyforge.org/) which is a gem that serves the same purpose as PIL. C++ has the boost libraries and included is "GIL" which will help you. If you're using another language, there are most certainly equivalents of PIL, RMagick, and GIL.

Use this image as the source image—it's 640 pixels wide and 359 pixels high.

Submit your solution

We're no longer offering the tee-shirt reward but if you're still interested in working with us, please submit your information and a link to your solution here: http://bit.ly/unshredder

PIL tips

```
from PIL import Image
image = Image.open('file.jpg')
data = image.getdata() # This gets pixel data


# Access an arbitrary pixel. Data is stored as a 2d array
where rows are
# sequential. Each element in the array is a RGBA tuple
(red, green, blue,
# alpha).
```

```
x, y = 20, 90
def get_pixel_value(x, y):
   width, height = image.size
   pixel = data[y * width + x]
   return pixel
print get_pixel_value(20, 30)
```

```
# Create a new image of the same size as the original
# and copy a region into the new image
NUMBER_OF_COLUMNS = 5
unshredded = Image.new("RGBA", image.size)
shred_width = unshredded.size[0]/NUMBER_OF_COLUMNS
shred_number = 1
x1, y1 = shred_width * shred_number, 0
x2, y2 = x1 + shred_width, height
source_region = image.crop(x1, y1, x2, y2)
destination_point = (0, 0)
unshredded.paste(source_region, destination_point)
# Output the new image
    unshredded.save("unshredded.jpg", "JPEG")
```

Tips

1. Don't overthink it. Use of really complex algorithms isn't needed. Our solution WITH the bonus exercise comes in at just over 150 lines of python.

2. Think about how you would quantify whether or not two shreds 'fit' together by using pixel data

3. Assume you're using the source image, or other normal photographs without edge-case patterns.

4. There are edge cases where the script we wrote with our approach will not work because of repeating patterns. This is OK in your script as well. Don't worry about special cases—focus on making the sample images work that we've provided.

5. Bonus Challenge: If you decide you want to auto-detect how many columns there are in an image, you should remember that there are a finite amount of columns that are possible given an image of a certain width if you assume columns are evenly distributed and uniformly sized.

Shredder

If you'd like to produce your own sample images, you can use our simple script here to generate some:

```
from PIL import Image
from random import shuffle

SHREDS = 10
image = Image.open(“sample.png”)
shredded = Image.new(“RGBA”, image.size)
width, height = image.size
shred_width = width/SHREDS
sequence = range(0, SHREDS)
shuffle(sequence)

for i, shred_index in enumerate(sequence):
    shred_x1, shred_y1 = shred_width * shred_index, 0
    shred_x2, shred_y2 = shred_x1 + shred_width, height
    region =image.crop((shred_x1, shred_y1, shred_x2,
shred_y2))
    shredded.paste(region, (shred_width * i, 0))

    shredded.save(“sample_shredded.png”)
```