2) Possible improvements

1. Add the following constraint for the user: set maximum file size or allowed extensions on the uploading form. This is a simple and quick thing to do and helps reduce the amount of data in the very beginning.

2. Resize the image on the server side and/or...

3. ...Change the format of the file on the server side (eg. change PNG to JPG, which weights much less, and the quality of the image is nearly identical)… or

4. ...Having PNG file, compress it without loss using external libraries, eg. OptiPNG… or

5 ...Strip the meta-data of the image (eg. we don't need information when the photo was taken, info about the camera, ISO etc.).

Points 2-5 require some processing (might be costly), but can be done later (not during the uploading process, eg. at night, when the CPU load is lower).

When the user accesses the *image/name* url, he/she gets the smaller image served, meaning the amount of data needed to be sent over the network is smaller.

6. Keep images on different servers/use Content Delivery Networks, which may be costly in the terms of money.

3) Antivirus check

Let's say we have:

Unix – OS

ClamAV – open-source antivirus engine; can be installed on unix systems

Initial assumptions:

The antivirus scanning process can be initialized directly from Python code. Every time the user uploads a file, we want to scan just that file (not the whole directory/file system; more comments on that case in next paragraphs). In order to start the scanning process we run a shell command. The exit code of that command determines if the file is a virus or not. The scan should be executed before the file is saved in the file system.

The new version of our application (including the scanning mechanism) should be deployed both to QA and production. Say we have several machines on production. We might upload the new version to some production machines, so we can observe how costly the scanning is in the 'real world'. If we accept it, upload the app to the rest of machines.

Additionally, we should have a possibility to dynamically switch the scanning on and off without the need to restart the app. We can achieve that by adding an entry to the app configuration file (kind of a boolean flag). The file would be read periodically by the app. We should also define a timeout for scanning an uploaded image in the config file. Timeout on QA is not 100% necessary (depends on the performance of the environment), but if we have it, the timeout on the QA environment would probably be higher than in production.

The virus definitions should be always up-to-date. Freshclam is the update tool for ClamAV. We can write a script (bash/Python...), which runs Freshclam and saves some logs. The script can be run periodically using crontab. This part should be done in production, not necessarily in QA.

Another issue is whether we want to scan existing image files after every update of virus definitions. If so, another process should be spawned to do such scan (can be started by the script mentioned it the previous paragraph). Again, not necessarily in QA.

Production should be also monitored by some tools like Zabbix. So we know that everything works, especially that scanning process is executed as expected.

4) Adding statistics

We might use a decorator function, which would count hits on pages according to the image name (see *image_upload/uploader/decorators.py* file in the repo). The decorated function would be the function responsible for showing the image to the user (*show_image* function from *views.py*). Basically the decorator function counts function calls.

Let's assume we have a relational database for collecting statistics. The decorator function might also update the statistics in the database.

Low traffic scenario: update the statistics every time the user accesses the page. So the stats will be always up to date.

High traffic scenario: update the statistics periodically (after some time or after specific number of hits) in a bulk operation. Another solution is to save a temporary file with statistics by the decorator function, and then another process reads the file and updates the stats in the database.