Creating and testing a dataset for studying the insight phenomenon

Petr Hanák

January 12, 2023

1 Introduction

The insight phenomenon (so-called aha moment) is a moment of sudden realization of the solution [1]. Usually, the moment feels relieving, the solution perfectly obvious and elegant and it is often described by the words "it just clicked" or "now it makes sense". Creating conditions to artificially evoke an insight is difficult. Insights contrary to analytical problem solving are not controlled and they usually take time to appear which all leads to the difficulty of the research. Yet studying the insight phenomenon is important for understanding the capabilities and functionalities of the human mind.

The goal of the project was to create a new dataset of images for studying the insight phenomenon and thus provide another tool for future research. The dataset consists of a series of distorted images which are slowly changing from the most distorted to the least. Various types of distortion are used to add the possibility to compare them.

This is the same goal and the same purpose of the dataset as of the constellation images [2] made by researchers at the University of Tartu. Their main focus is on the "iterative inference" - finding a solution in an iterative fashion by revising it over and over again. On the contrary, the focus of our dataset is more on the timing of the solution and the amount of information needed for it.

Our dataset was tested by a small group of respondents to verify that it can be in fact used for future research and also to discover potential issues with the dataset. The issues were then fixed according to the results.

2 Overview of the dataset

The basic building block of the dataset is a set of variously distorted images ranging from the most distorted to the least distorted. By distorting the image, we mean reducing the clarity of the information present in the image so it is more difficult to name the object in the image. Each set consists of 12 images. The first is completely distorted and it is almost impossible to name the object on the image. The last image is almost the same as the original.

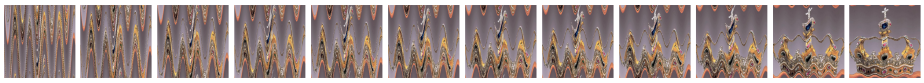

Figure 1: The full set of 12 images

The intended use of the dataset is to show the images from the set sequentially to the respondents. As the clarity of the images improves, the respondent's task is to say at which moment they recognize the object in the image. We will describe this in further detail in Section 3.

Images from the THINGS database [3] were used as the source images for generating the dataset.

2.1 Types of the distortion

We decided not to use only one type of distortion but rather to have different sets distorted in different ways. Thanks to this the dataset can be used to test the differences between different distortions. The types of distortion we used are wavy images, jigsaw images and pixelated images (they will be later referenced by these names).

2.1.1 Wavy images

This type of distortion deforms the image in a wavy fashion. Each pixel is moved on the y-axis depending on the sinus of its x-coordinate. Pixels which should be located over the top of the image are moved to the bottom and vice versa. The amount of distortion is controlled by changing the amplitude of the sinus function. The bigger the amplitude, the more distorted image.

This type of distortion slightly reduces the amount of information in the image because sometimes the pixels override each other. However, it focuses more on reducing the clarity and changing the spatial information.

Figure 2: The first, the middle and the last image from wavy set

2.1.2 Jigsaw images

This type of distortion divides the image into a number of squares and shuffles the positions of these squares. This results in an image which looks like a scrambled jigsaw puzzle. The amount of distortion is controlled by changing the total number of squares. The more squares, the more distorted image.

In this type of distortion, no information about the image is lost. It still contains the same pixels only with changed positions.

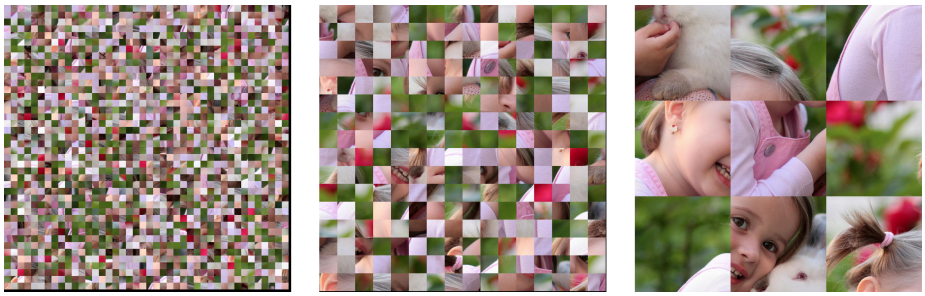

Figure 3: The first, the middle and the last image from jigsaw set

2.1.3 Pixelated images

The final type of distortion creates so-called pixel art from the image. The number of colours is reduced to six using the k-means clustering. The resulting image is then pixelated by averaging the colours in a given square. The amount of distortion is controlled by changing the total number of pixels. The fewer pixels, the more distorted image.

This type of distortion purposefully reduces the amount of information in the image. The image has de facto fewer pixels with fewer colours.

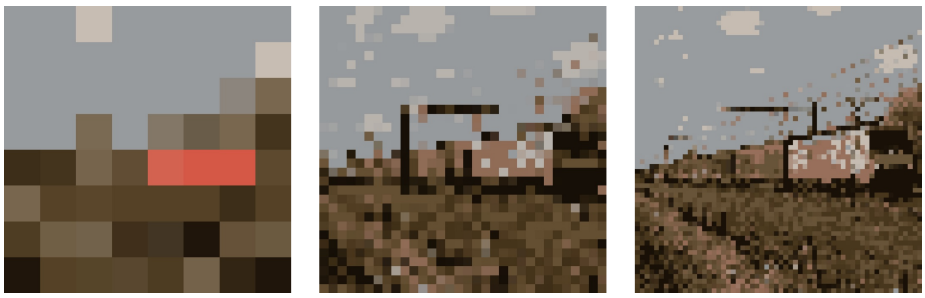

Figure 4: The first, the middle and the last image from pixelated set

2.2 The script for generating the dataset

A script was written in Python to generate the dataset automatically from any source images. See Appendix A for the script. It uses the OpenCV library [4]

for manipulating the images and the NumPy library [5] for matrix calculation. These libraries have to be installed prior to using the script.

The script offers a simple command-line interface. It takes one required argument `source_directory` specifying the directory with the source images. The type of set that needs to be generated is specified with options `-w` or `--wavy` for the wavy set, `-j` or `--jigsaw` for the jigsaw set, `-p` or `--pixelated` for the pixelated set and `-a` or `--all` for generating all the types of sets. The sets are generated to appropriately named directories. Help for the script is displayed with an option `-h`.

3 Testing the dataset

One of the goals of the project was to test if the dataset can be used in actual research. The research itself is out of the scope of this project. The key questions the testing was focusing on were:

- Is it likely to have insight while solving the task?
- Is the range of distortion appropriate? Isn't the first image in the set too distorted or the last too clear?
- Does the results depend on the background of the image?
- Are there any systematic flaws with the dataset?

3.1 The questionnaire

The task for testing the dataset was implemented using a Google Forms questionnaire. The respondent was presented with the image and the possibility to either say what is in the image or to proceed to the next - less distorted - image. After writing what was in the image the respondent was asked if the solution was accompanied by an insight. Then the same process continued with the next image.

At the end of the questionnaire, the respondent was asked to assess the difficulty of the different types of distortion. The respondent could also leave further notes about their strategy for solving the task and other observations.

3.2 The dataset used

The dataset generated for the questionnaire had 15 sets created from 15 different images. Each type of distortion was present in 5 sets. The source images were picked to clearly display one specific object which can be named. 7 images were with a single-colour background and the rest with a background corresponding to the subject.

4 Results

The responses were collected over the course of one month. In total, we managed to collect responses only from 14 respondents. This was possibly caused by the length of the questionnaire. We wanted to make sure that we do not have a group of respondents who would be highly visually oriented so we asked them to assess how much they feel like visual learners. We can see in Figure 5 that the distribution is normal.

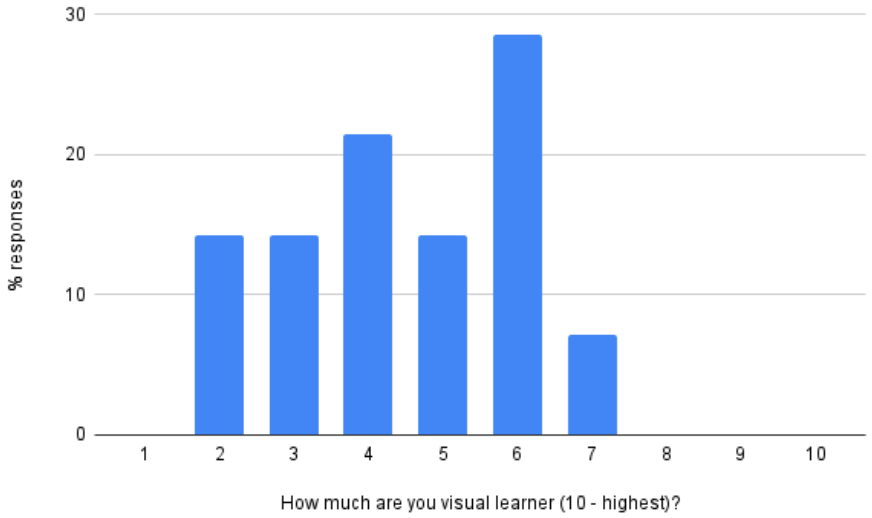

Figure 5: Respondents' assessment how much they are visual learners

The responses from the object identification were classified as correct or wrong by hand. We will now go through all the questions we wanted to answer by testing the dataset.

4.1 Insights while solving the task

The respondents were experiencing insight in a high percentage of their answers. The highest probability of insight was while solving the pixelated images. Almost 63% of the answers were accompanied by an insight. We can see the comparison between different types of distortion in Figure 6.

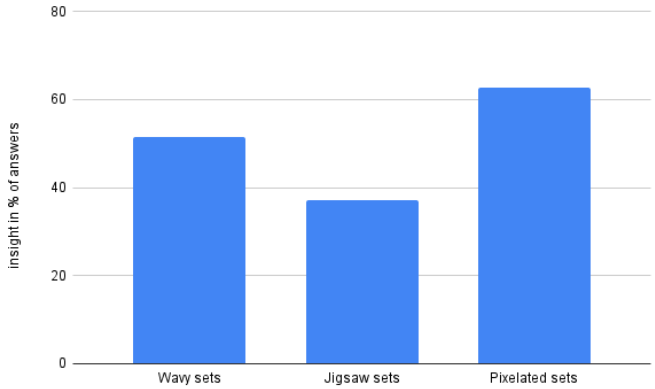

Figure 6: Comparison of insights

We attempted to partially verify that the insights identified by the respondents were in fact insights (and that the respondents understood the definition of insight correctly). We used the fact that the answers with insight should have a lower error rate than analytical answers [6] so we compared the correctness of answers with and without insight. The results in Figure 7 give us enough confidence to say, that the insights the respondents were experiencing were in fact insights.

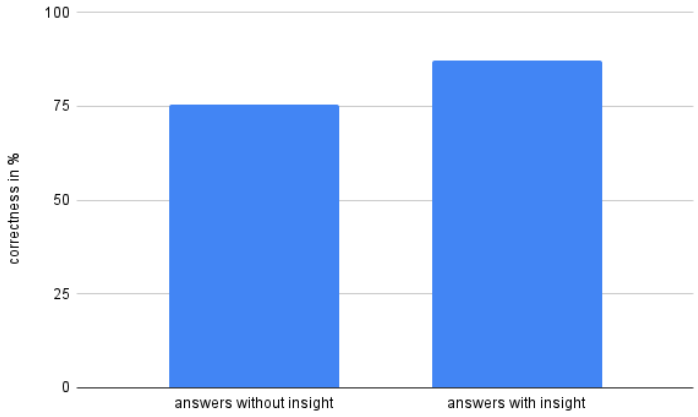

Figure 7: Comparison of correctness

In general, we can say that the percentage of insights is satisfactory and that the dataset achieved the goal in this aspect.

4.2 The range of distortion

We were attempting to answer the question if the range of the distortion is appropriate. By the word appropriate we mean that the respondents are answering from the very beginning to the very end of the set. In Figure 8 we can see that the answer to the question varies across the different types of distortion.

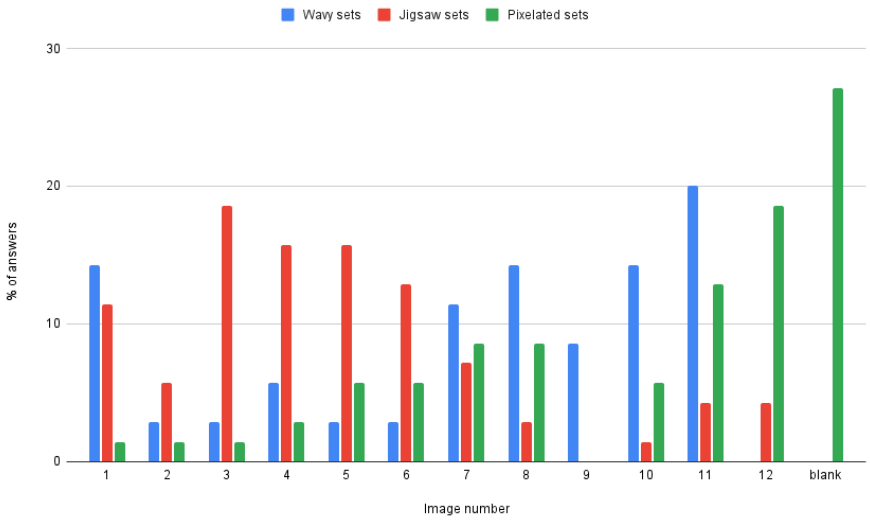

Figure 8: Percentage of answers for a given image in the set

- Wavy sets - the range of distortion is appropriate. The respondents are answering throughout the whole set and no images are so to say useless.
- Jigsaw sets - we can see that there are very few answers for the last four images in the sets. This issue could be solved by making the changes between the images smaller or by starting with a more distorted image.
- Pixelated sets - in these sets the respondents were not able to identify the object even in the last image of the set in almost 30% of cases. Also, there is a very small number of answers for the first three images of the sets. This issue could be solved by starting with a less distorted image or by making the changes between the images bigger.

4.3 Influence of the background of the image

We compared when was an object identified in the sets of images with a single-colour background and with a background corresponding to the object. We can

see the results across all sets in Figure 9. According to these results, we have not found any significant and systematic difference between images with and without background.

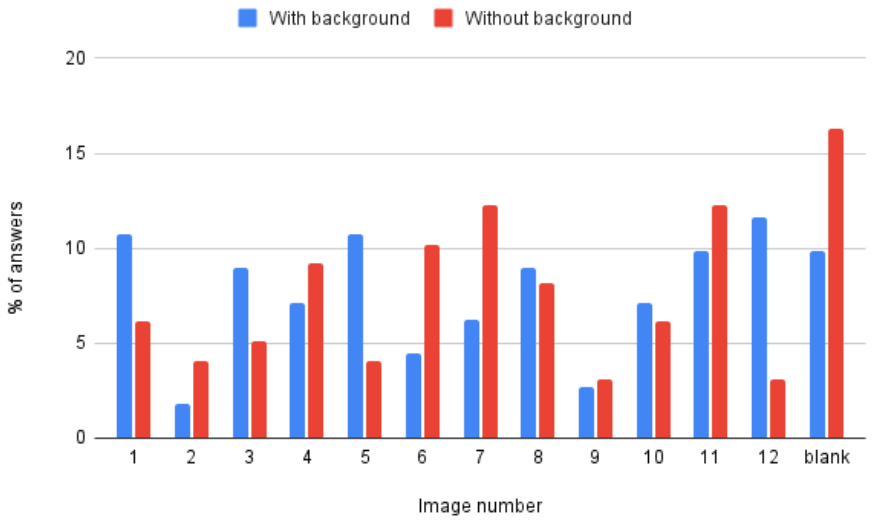

Figure 9: Difference between images with and without background

4.4 Systematic flaws in the dataset

One systematic flaw was found in the dataset. We found out that by applying the correct strategy to some types of distortion the results significantly improves. This could cause a devaluation of the data collected in future research.

For jigsaw images, it is possible to focus only on one square ignoring the rest of the image. It is thus much easier to identify the object from a single undistorted piece of the source image. Some respondents even zoomed in using the web browser as described by one of them: *I zoomed in jigsaw pictures to find a typical part of the object (for example the tip of the banana).* This issue could be solved by increasing the number of squares so that no important feature of the object would be present in a single square.

Pixelated sets also had a successful strategy of looking at them from a distance of a few metres. Or as described by one of the respondents: *I put off my glasses.* Solution for this issue would be to control the task environment and not allow the respondents to leave their chairs while filling in the questionnaire.

4.5 Other observations

We have a few other observations from testing the dataset to mention.

The respondents' evaluation of the difficulty of the different types of distortion is corresponding with reality. The pixelated images were marked as the most difficult which is in accord with the 30% of unrecognized objects. The jigsaw images felt the easiest for the respondents which could be also expected from the previous results.

The correctness of the responses was quite high. 75% for wavy and pixelated sets and 90% for the jigsaw images. We are not counting respondents which did not identify the object at all as incorrect.

5 Modification of the dataset

The script for generating the dataset was modified to solve the discovered issues.

The first image in jigsaw sets is now more distorted and the changes throughout the dataset are more gradual. The range of distortion is thus narrower.

The first image in the pixelated sets is now less distorted and the changes are faster throughout the dataset. As a result, the range of the distortion is now wider.

6 Conclusion

We created a dataset for studying the insight phenomenon. The dataset consists of a series of distorted images where the respondent should identify the object on them. There are three different types of distortion available in the dataset so they can be compared against each other.

The script for generating the dataset is available in Appendix A so it can be used to generate the dataset from different source images. The advantage of the dataset is that the parameters can be easily modified in the script and a special version of the dataset can be created according to the current needs. The generation is also reasonably fast (1 - 2 seconds for one set) and fully automatic so the sets can be potentially generated on the go while the respondent is doing the task.

The dataset was tested by a small group of respondents. It was verified that the dataset works as expected. The respondents were experiencing insight in more than 50% of cases. Minor issues were found in the range of the distortion in a set and they were fixed according to the results.

The dataset is verified that it can serve its purpose and can be in fact used for actual research.

References

[1] Antonio J. Osuna-Mascaró and Alice M. I. Auersperg. Current understanding of the "insight" phenomenon across disciplines. *Frontiers in Psychology*, December 15, 2021.

[2] Tarun Khajuria, Kadi Tulver, Taavi Luik, and Jaan Aru. Constellations: A novel dataset for studying iterative inference in humans and ai. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 5138–5148, 2022.

[3] L. M. Stoinski, J. Perkuhn, and M. N. Hebart. Things+: New norms and metadata for the things database of 1,854 object concepts and 26,107 natural object images. July 20, 2022.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[6] Carola Salvi, Emanuela Bricolo, John Kounios, Edward Bowden, and Mark Beeman. Insight solutions are correct more often than analytic solutions. *Think Reason*, 22(4):443–460, February 2016.

A Script for generating the dataset

```python
#!/usr/bin/env python3

import cv2
from random import randint
import numpy as np
import math
import os
import argparse

# the speed of the change throughout the sets
PIXELATED_SPEED = [16, 19, 23, 28, 33, 39, 46, 53, 60, 70, 80, 90]
WAVY_SPEED = [300, 200, 150, 125, 115, 100, 82, 75, 62, 50, 30, 20]
JIGSAW_SPEED = [50, 46, 42, 34, 30, 24, 22, 20, 18, 16, 14, 10]


# generate pixelated set from image "img" and save the result with
                                    the name "name"
def pixelate(img, name):
    height, width = img.shape[:2]
    print(f"\t[+] Generating pixelated set")
    for i, number_of_pixels in enumerate(PIXELATED_SPEED):
        print(f"\t\t[+] Image {i + 1}/{len(PIXELATED_SPEED)}")
        Z = np.float32(img).reshape(-1, 3)
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.
                                        TERM_CRITERIA_MAX_ITER,
                                        20, 1.0)
        K = 6
        ret, label, center = cv2.kmeans(Z, K, None, criteria, 10,
                                        cv2.KMEANS_RANDOM_CENTERS
                                        )
        center = np.uint8(center)
        temp = center[label.flatten()]
        temp = temp.reshape(img.shape)

        temp = cv2.resize(temp, (number_of_pixels, number_of_pixels
                                        ), interpolation=cv2.
                                        INTER_LINEAR)
        output = cv2.resize(temp, (width, height), interpolation=
                                        cv2.INTER_NEAREST)

        cv2.imwrite(f'pixelated/{name}{i}-{number_of_pixels}.jpg',
                                        output)


# generate wavy set from image "img" and save the result with the
                                    name "name"
def wave(image, name):
    height, width = image.shape[:2]

    print("\t[+] Generating wavy set")
    for num, intensity in enumerate(WAVY_SPEED):
        output = image.copy()
        print(f"\t\t[+] Image {num + 1}/{len(WAVY_SPEED)}")
        for i in range(height):
            for j in range(width):
```

```
                try:
                    output[i + int(intensity * math.sin(j / int(
                                                        width / 30)))
                                                        , j] = image[
                                                        i, j]
                except:
                    output[i + int(intensity * math.sin(j / int(
                                                        width / 30)))
                                                         - height, j]
                                                         = image[i, j
                                                        ]
        cv2.imwrite(f'wavy/{name}{num}-{intensity}.jpg', output)


# generate jigsaw set from image "img" and save the result with the
                                    name "name"
def jigsaw(img, name):
    img = cv2.resize(img, (1150, 1150))
    height, width = img.shape[:2]
    print("\t[+] Generating jigsaw set")
    for num, square_num in enumerate(JIGSAW_SPEED):
        print(f"\t\t[+] Image {num + 1}/{len(JIGSAW_SPEED)}")
        output = np.zeros((height, width, 3), dtype=np.uint8)
        occupied = []
        square_size = int(width / square_num)
        for i in range(0, width - square_size + 1, square_size):
            for j in range(0, height - square_size + 1, square_size
                                                    ):
                position = (randint(0, square_num - 1) *
                                                    square_size,
                                                    randint(0,
                                                    square_num - 1) *
                                                     square_size)
                while position in occupied:
                    position = (randint(0, square_num - 1) *
                                                    square_size,
                                                    randint(0,
                                                    square_num -
                                                    1) *
                                                    square_size)
                occupied.append(position)
                section = img[j: j + square_size, i:i + square_size
                                                    ]
                output[position[0]: position[0] + square_size,
                                                    position[1]:
                                                    position[1] +
                                                    square_size] =
                                                    section

        cv2.imwrite(f'jigsaw/{name}{num}-{square_num}.jpg', output)


def run(args):
    directory = args.source_directory
    if args.all:
        os.mkdir("pixelated")
        os.mkdir("wavy")
```

```
        os.mkdir("jigsaw")
    else:
        if args.pixelated:
            os.mkdir("pixelated")
        if args.wavy:
            os.mkdir("wavy")
        if args.jigsaw:
            os.mkdir("jigsaw")
    num_of_img = len([name for name in os.listdir(directory) if os.
                                        path.isfile(os.path.join(
                                        directory, name))])

    for i, filename in enumerate(os.listdir(directory)):
        print(f"[+] Running generation for image {i + 1}/{
                                            num_of_img}")
        f = os.path.join(directory, filename)
        if os.path.isfile(f):
            img = cv2.imread(f)
            if args.all:
                pixelate(img, os.path.splitext(filename)[0])
                wave(img, os.path.splitext(filename)[0])
                jigsaw(img, os.path.splitext(filename)[0])
            else:
                if args.pixelated:
                    pixelate(img, os.path.splitext(filename)[0])
                if args.wavy:
                    wave(img, os.path.splitext(filename)[0])
                if args.jigsaw:
                    jigsaw(img, os.path.splitext(filename)[0])


parser = argparse.ArgumentParser(prog="Dataset generator")
parser.add_argument('source_directory', help="The directory, where
                                    the source images are located")
parser.add_argument('-a', '--all', action="store_true",
                    help="Generate all the types of distortion (
                                                    alternative
                                                    to -w -j -p)"
                                                    )
parser.add_argument('-w', '--wavy', action="store_true", help="
                                    Generate wavy images")
parser.add_argument('-j', '--jigsaw', action="store_true", help="
                                    Generate jigsaw images")
parser.add_argument('-p', '--pixelated', action="store_true", help=
                                    "Generate pixelated images")
parser.set_defaults(all=False, wavy=False, jigsaw=False, pixelated=
                                    False)
args = parser.parse_args()
if not (args.all or args.wavy or args.jigsaw or args.pixelated):
    parser.error('No action requested, specify the type of
                                        distortion with one or more
                                        option from (-w -j -p -a)')

run(args)
```