Automated Instagram Drawing Bot

Rodrigo Menéndez Trejo

November 2025

Abstract

This short project demonstrates how to control an Android phone from a computer to reproduce drawings automatically inside the Instagram application, making use of the new features available in Instagram's chat drawing tool. The system uses computer vision to process a black–and–white image, extract its skeletonized contours, and send swipe gestures through Android Debug Bridge (ADB) to simulate a user drawing on the screen.

1 Introduction

The goal of this project is to create an automated script that can draw images on Instagram's chats using the phone's touch interface. Recently, Instagram introduced a new feature that allows users to draw freely in their chat messages. Due to my limited drawing skills and my desire to experiment with computer vision libraries, I decided to develop this small project to impress my friends with my newly developed "art skills".

In this document, I describe the architecture of the system, the initial configuration needed to set up the environment, the implementation details, the results obtained and the main drawbacks.

2 Architecture

The system is composed of three main modules: the **computer vision module**, the **command generator**, and the **execution layer**. The first module uses OpenCV and scikit image to preprocess, threshold and skeletonize the input image. The command generator converts the pixel coordinates of the processed image into swipe commands for ADB. Finally, the execution layer sends the commands to the phone through ADB, while the user views the live feedback on `scrcpy`.

3 Initial Setup

The main problem to solve lies in controlling the phone's touch interface from a computer. Due to poor compatibility (non-existent) with IOS devices and Windows, I opted to use an Android phone that allows more flexibility in terms of external control. There are several tools available for this purpose, but I chose to use Android Debug Bridge (ADB) since it is widely supported

and allows sending touch input commands directly to the device unlike other tools such as phonelink. For real-time visual feedback `scrcpy` is employed to mirror the device's screen on the computer, enabling both observation and manual interaction when needed.

4 Implementation

4.1 Preprocessing

The input images are first converted to grayscale, reducing them to a single intensity channel. This simplifies subsequent processing while preserving the essential structure of the drawing.

Once in grayscale, the image is binarized using adaptive thresholding. Unlike a global threshold which applies a single cutoff value to all pixels—adaptive thresholding computes a local threshold for each region based on neighborhood statistics (typically the mean or Gaussian weighted average). This makes the system robust to lighting variations and uneven backgrounds, ensuring that all dark lines are detected even if parts of the image are brighter or faded.

After binarization, the resulting binary mask often contains small gaps, isolated pixels, or jagged edges. To address this problem, morphological operations are applied: closing (dilation followed by erosion) bridges tiny discontinuities in the lines, while opening (erosion followed by dilation) removes isolated noise points. These steps produce smoother contours.

Finally, the cleaned binary image undergoes skeletonization. This process iteratively erodes the white regions until each stroke is reduced to a single pixel wide centerline. The skeleton preserves the topology of the original drawing maintaining connectivity and shape but eliminates redundant thickness. The resulting structure provides a minimal, vector like representation of the image that can be directly translated into coordinates for the Android device.

4.2 Gesture Translation

Once the skeletonized representation of the image has been obtained, each continuous contour is interpreted as a potential drawing path. These contours are extracted as ordered sets of pixel coordinates (x_i, y_i) that define the trajectory of a line.

To reproduce these trajectories on the phone's screen, each point in the contour is mapped from image coordinates to *screen coordinates* using a linear transformation that accounts for the resolution of the target device and the desired drawing scale.

The converted coordinates are then sent to the Android device through the ADB. Each pair of consecutive points (x_i, y_i) and (x_{i+1}, y_{i+1}) defines a short swipe command of the form:

```
adb shell input swipe x1 y1 x2 y2 duration
```

where `duration` controls the temporal smoothness of the stroke. A continuous sequence of those commands reconstructs the full drawing on the device.

4.3 Execution Environment

The implementation was tested on a Samsung Galaxy A51 running One UI, connected to a Windows 11 PC with Python 3.10. The necessary libraries include OpenCV, scikit-image, and standard Python Packages to manage the rest of the workflow.

5 Results

The results demonstrate that the implemented system can reproduce a variety of black-and-white images with reasonable fidelity. Simple line drawings and high-contrast images yield the best results, while more complex images with fine details or gradients may lose some information during preprocessing due to the limitations explained in the section 6. The following figures illustrate some of the input images alongside their corresponding skeleton and finally the outputs drawn on Instagram, both outputs images were captured directly from the phone screen using `scrcpy` and edited to remove the phone interface for better visualization:

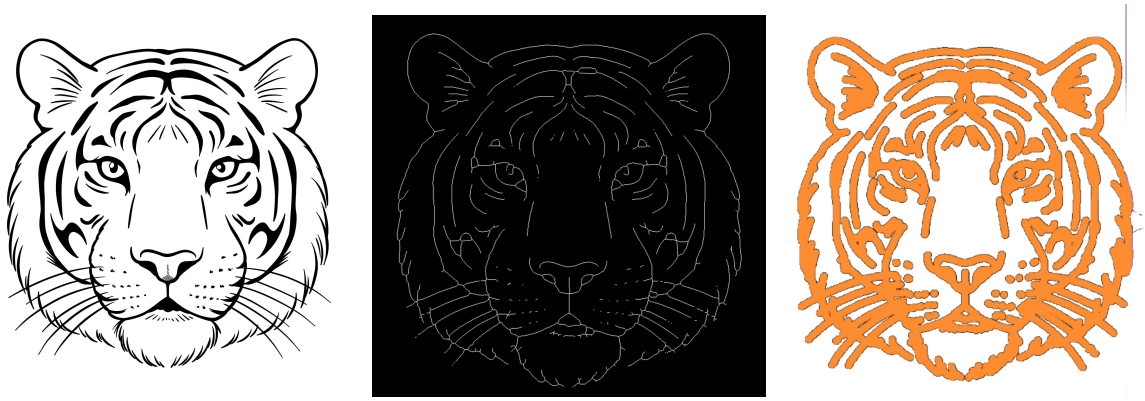

Figure 1: Input image (left), skeletonized contour (center), and Instagram drawing output (right) of a tiger face.

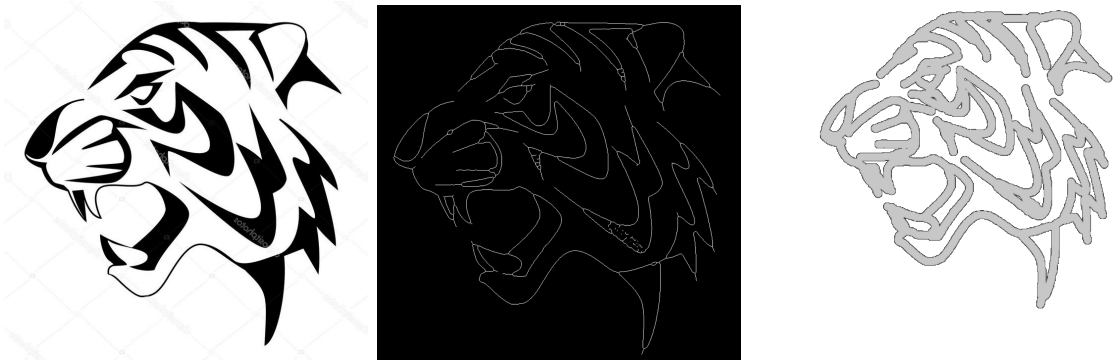

Figure 2: Input image (left), skeletonized contour (center), and Instagram drawing output (right) of a tribal tiger face.

6 Limitations

One of the main drawbacks of the current implementation is its limited execution speed. Because each `swipe` command is sent sequentially, processing complex images can require a considerable amount of time. Another limitation arises from the Instagram drawing feature itself, which does not provide smaller brush sizes, making it difficult to capture fine or specific details. In addition, the available drawing area on the mobile interface is relatively small, and when the script is executed directly on the phone, this restriction further reduces the scale and precision of the reproduced image. The combination of these factors constrains the system's ability to generate large or highly detailed sketches efficiently.

7 Conclusion

This project is merely a fun experiment combining computer vision techniques with mobile phone control. Nevertheless, the results are promising taking into account the simplicity of the approach and the small amount of time invested, only one afternoon. This project has a large scalability potential, as it could be extended to other drawing applications with higher precision and more advanced tools.