AUTOMATIC FACE ROTATION: IMPROVED DISPLAY ORIENTATION FOR HANDHELD DEVICES USING CONVOLUTIONAL NEURAL NETWORKS

**Riley Tallman**

Arizona State University

rtallman@asu.edu

ABSTRACT

We propose an improved method of rotating smartphones, tablets, and other device displays with Automatic Face Rotation using a convolutional neural network (CNN). Many users of smartphones and tablets experience a problem with their device screen being oriented the wrong way during use. This paper introduces a new algorithm to fix that issue by correctly orienting the screen relative to the user’s face using a CNN. The CNN model is trained to predict the rotation of faces in a variety of environments. The algorithm uses a confidence threshold and analyzes multiple images to be robust. Our solution solves the existing rotation problem, is battery and CPU efficient, and causes no noticeable lag to the user during use.

1 INTRODUCTION

Handheld devices such as smartphones and tablets can be used in different orientations. They can be held vertically, to a side, or upside-down, and the device’s screen automatically changes to match which orientation the device is being held in. We will refer to the four possible orientations as *portrait*, *upsidedown*, *landscapeLeft*, and *landscapeRight*. The problem of automatic display orientation is a challenge of determining what position the user is using the device in. Current solutions commonly use the Earth’s gravity to calculate the rotation of the device. Specifically, modern devices use an accelerometer to to measure the acceleration (and/or force) acting on the device due to gravity in the , , and directions. These observations are then used to confidently orient the display relative to gravity.

Unfortunately, using the accelerometer fails when a user lies down to one side. Let’s say for example that a user is sitting upright and holding a smartphone vertically, so the phone’s display is oriented to *portrait*. This orientation is correct since the user is also vertical. When the user lies down to the right, the device rotates from *portrait* to *landscapeRight* because that is the correct orientation relative to gravity. However, this configuration is wrong since the user has not actually rotated the device; they are still holding it vertically relative to themselves. The screen should remain *portrait*, but the accelerometer/gravity solution fails to detect this. In the remaining sections, we loosely refer to the current solution as “gravity” because it uses gravity to orient the device’s display.

Motivated to solve this, we propose Automatic Face Rotation (Auto-Face Rotation, Auto-Face, or Face Rotation) which automatically orients a display relative to the user’s face. Auto-Face solves the problem with the current solution using convolutional neural networks (CNNs). It can confidently detect the rotation that the user is using the device instead of just using gravity. In this paper we demonstrate the fast and battery efficient algorithm that incurs no noticeable lag to the user. In the following sections we describe our implementation, future work, etc.

2 AUTOMATIC FACE ROTATION

We present our Automatic Face Rotation algorithm which uses gravity to detect rotation changes and a convolutional neural network to correctly orient a device’s display. The accelerometer is still useful to detect rotation changes. Once a change is detected, the CNN begins analyzing images from the front-facing camera. After analysis, the CNN will either confirm that the gravity rotation is correct, or will override the gravity orientation (aka, perform a “gravity override”). The verification step is performed with a CNN that detects the user’s face and outputs what orientation the screen should be, relative to the user. If the CNN output is the same orientation that gravity outputs, then the program rotates to that orientation. If, however the CNN output does not match the gravity output, the CNN will override gravity and rotate to the correct orientation that is relative to the user if there is enough confidence in the CNN’s output. This case is depicted in the first sample output of Table 1.

**Confidence.** Misclassification may occur if an image is terribly blurry, dark, or bright, or if the user’s face is obstructed. To combat this, we define a confidence threshold to control gravity overrides, and analyze multiple images during analysis. In order for the CNN model to override the gravity orientation, it needs to reach a confidence greater than some threshold on two consecutive images that it analyzes. If the first two images fail to meet the confidence threshold, then the program will continue to analyze images from the front camera up to images.

**ImageQueue.** Auto-Face Rotation can analyze multiple images (the *imageQueue*) to ensure correctness. This technique is analogous to using multiple keys to authorize a military strike: all keys need to work in order to execute the operation. The parameter specifies how many images, or keys, to use in order to override gravity. The loop on line 5 verifies that all confidence outputs are above the threshold and all predictions are consistent (all the same orientation, i.e., *portrait* & *portrait*) within the *imageQueue*. Setting greater than 1 helps prevent misclassifications.

**Multiple Images.** To be robust, the algorithm can analyze more images than are in the *imageSet*. When the CNN is not confident or if the predictions are inconsistent, the process if repeated with one new image, and the first in the *imageQueue* is discarded. This creates a sliding window that shifts over one image in the camera feed up to times. Once a total of images are analyzed, Auto-Face returns “Low Confidence” and the CNN predictions are discarded. When this happens, whatever orientation that gravity has predicted will be used to rotate the display.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Class Confidence | | | | Result |
| portrait | upsidedown | landscapeLeft | landscapeRight |
| Gravity |  |  |  | 1 | OVERRIDE:  **portrait** |
| CNN | **0.93** | 0.05 | 0.01 | 0.01 |
| CNN | **0.93** | 0.06 | 0.01 | 0 |
| Gravity |  |  |  | **1** | Low Confidence:  **landscapeRight** |
| CNN | 0.54 | 0.23 | 0.11 | 0.12 |
| CNN | 0.56 | 0.20 | 0.01 | 0.13 |
| CNN | … |  |  |  |  |

Table 1: Two sample outputs and their resulting orientation. CNN [1] is the first analyzed image, and CNN (2) is the second. The top sample output is oriented to portrait because both images are above a confidence threshold of . All analyzed images in the second sample are below , so the output from gravity is used.

|  |
| --- |
| **Algorithm 1** CNN with Confidence Threshold and Multiple Image Testing |
| 1: *imageQueue =* analyzeNextImages  2: **do**: |
| 3: *imageQueue* = analyzeNextImages  4: *override* = true |
| 5: **for** *image* **in** *imageSet*:  6: **if** *image*.confidence < **and** isInconsistent(*image*.prediction): |
| 7: *override* = false  8: **if** *override*:  9: return ‘OVERRIDE’  10: *imageQueue*.pop() |
| 11: **while**  12: return ‘Low Confidence’ |

4 CUSTOM CNN

We’ve created and trained a custom convolutional neural network to identify the face of the user and determine the correct orientation. We formulate the problem as a 4-class classification problem and train our model to classify the four possible orientations: *portrait*, *upsidedown*, *landscapeLeft*, and *landscapeRight*. Using a single input image, the CNN model outputs four numbers that sum to 1, each representing the confidence of the image belonging to its class (See Table 1).

4.1 ARCHITECTURE DESIGN

The CNN architecture is most important for engineering efficient image analysis models. The size and efficiency of CNNs are characterized by the number of parameters they have, which depends on the architecture. The number of parameters directly influence the number of calculations that are executed during image analysis, and of course, more calculations take more time. A CNN that is too big (has too many parameters) will not be able to analyze images quickly. On the other hand, a CNN that is too small will not learn very well and will have many misclassifications.

As mentioned before, our architecture needs to be small enough and quickly analyze images so that the user does not notice lag before rotation. Our model uses a VGG style network shown in figure 4. The architecture is composed of 4 nodes which each represent two convolutional layers followed by a single maxpooling and dropout layer. All convolutions use kernels with a stride of (1 in both and direction). In our testing, adding nodes helped our model’s accuracy.

**Preprocessing.** To help the CNN discover relevant features that generalize to the problem, we use a of Laplacian edge detection filter before feeding it into the first convolutional layer. Auto-Face is also trained using normalized values.

4.3 TRAINING

We selected the Helen dataset because the majority of images had faces that were looking at the camera lens, which is like how a user would be looking at a smartphone or tablet’s camera while using it. Every image was flipped horizontally, then rotated three times to create 7 new images for each image. Finally, the images were scaled down to 128x128 pixels. During training, additional data augmentation was used like small random rotations and brightness alterations. The images were also linearly blurred to simulate the user shaking their device while rotating. These techniques create a robust training set which allows our model to correctly classify many general cases.

5 RESULTS

The accuracy of our CNN classifier represents how well the model will perform. Accuracy is defined in Eq. (1), and it is a number between 0 and 1.

Our goal is to correctly orient a device screen relative to the user using the front-facing camera while remaining extremely battery and CPU efficient. Another goal is to be fast enough so that user cannot notice any additional lag compared to gravity. Our algorithm builds on top of the ideas of fully convolutional neural networks and deeply-supervised nets.