ML Chessboard Recognition Part 2

Stephen Huan

Overview

- See previous lecture (in references)
- We have a neural net that acts like a filter
- Integrate into existing corner detection algorithms

Steps

1. Apply OpenCV's function `goodFeaturesToTrack` to get initial corners
2. Binarize the image in preparation for neural network input
3. Run neural network
4. Extract top X corners from output
5. Binarize image with Otsu's binarization

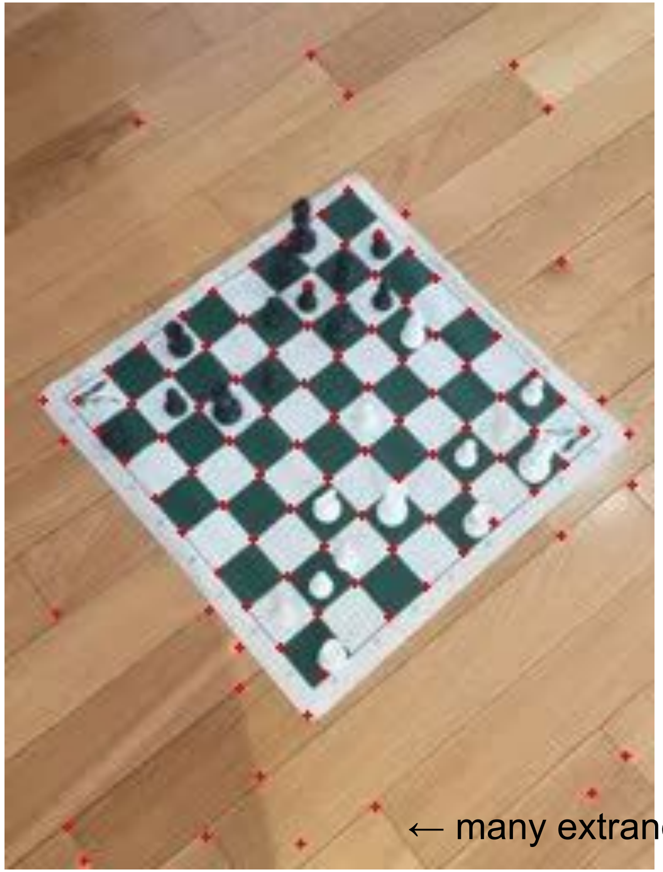
Step 0: Loading and Preprocessing the Image

```
# load image and greyscale
img = cv.imread(path)
grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

Step 1: `goodFeaturesToTrack`

- Use OpenCV’s function to get initial corners
- Ask it for more corners than we need to get as many corners as possible
- We will filter extraneous corners later

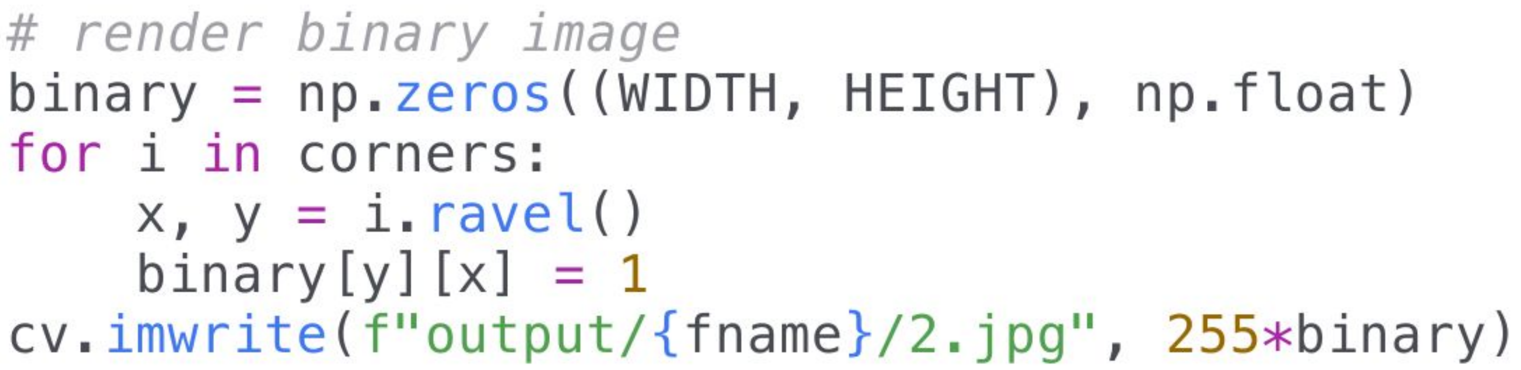
```python
# get initial corners
corners = cv.goodFeaturesToTrack(grey, round(1.5*81), 0.01, 10)
corners = np.int0(corners)
cv.imwrite(f"output/{fname}/1.jpg", render_points(img, corners))
```



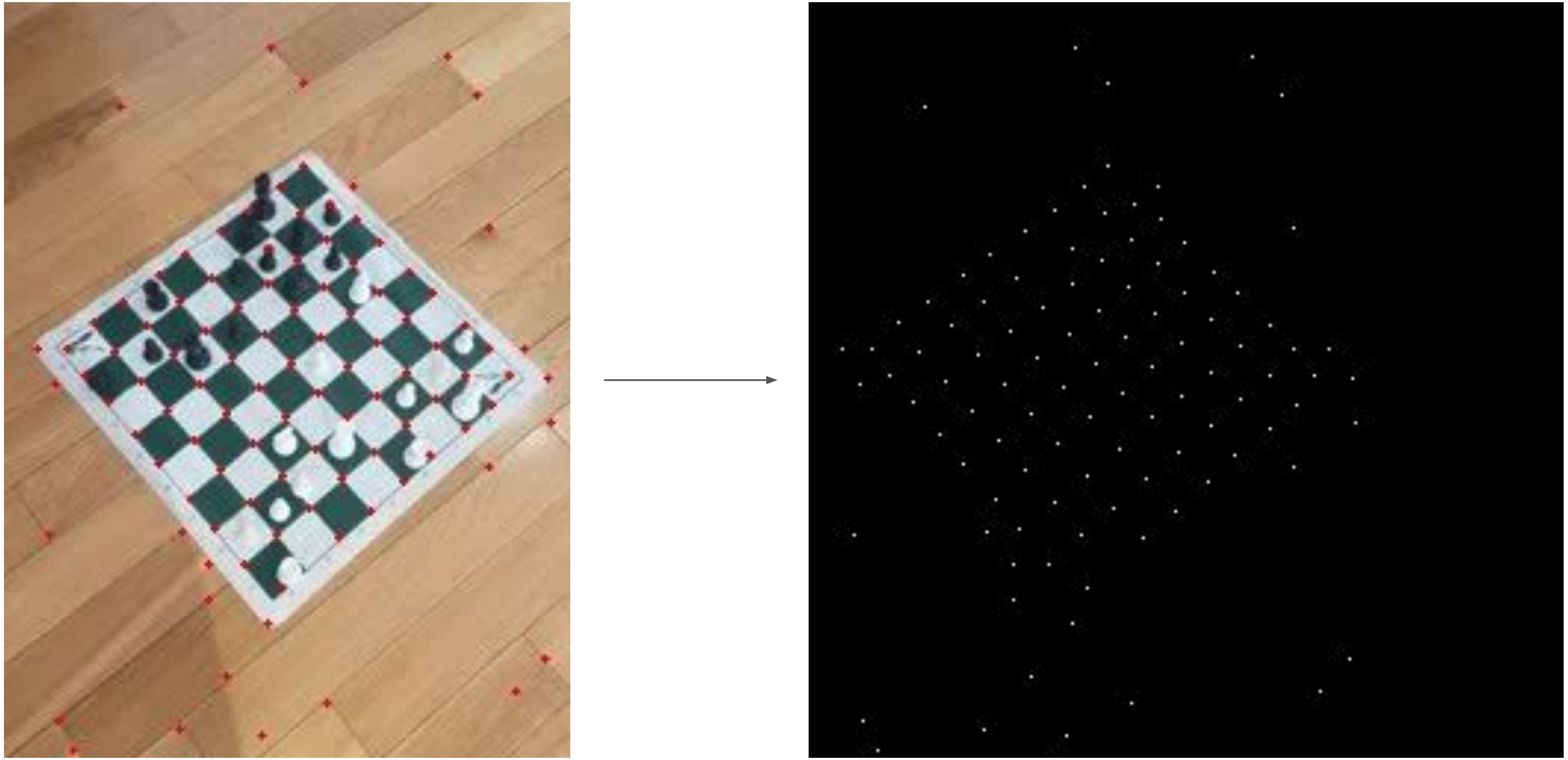
← many extraneous corners

Step 2: Binarize the image

- Neural network takes binary image as output and outputs binary image
- Convert corner data into binary image
- Also make image height/width the same by implicitly padding



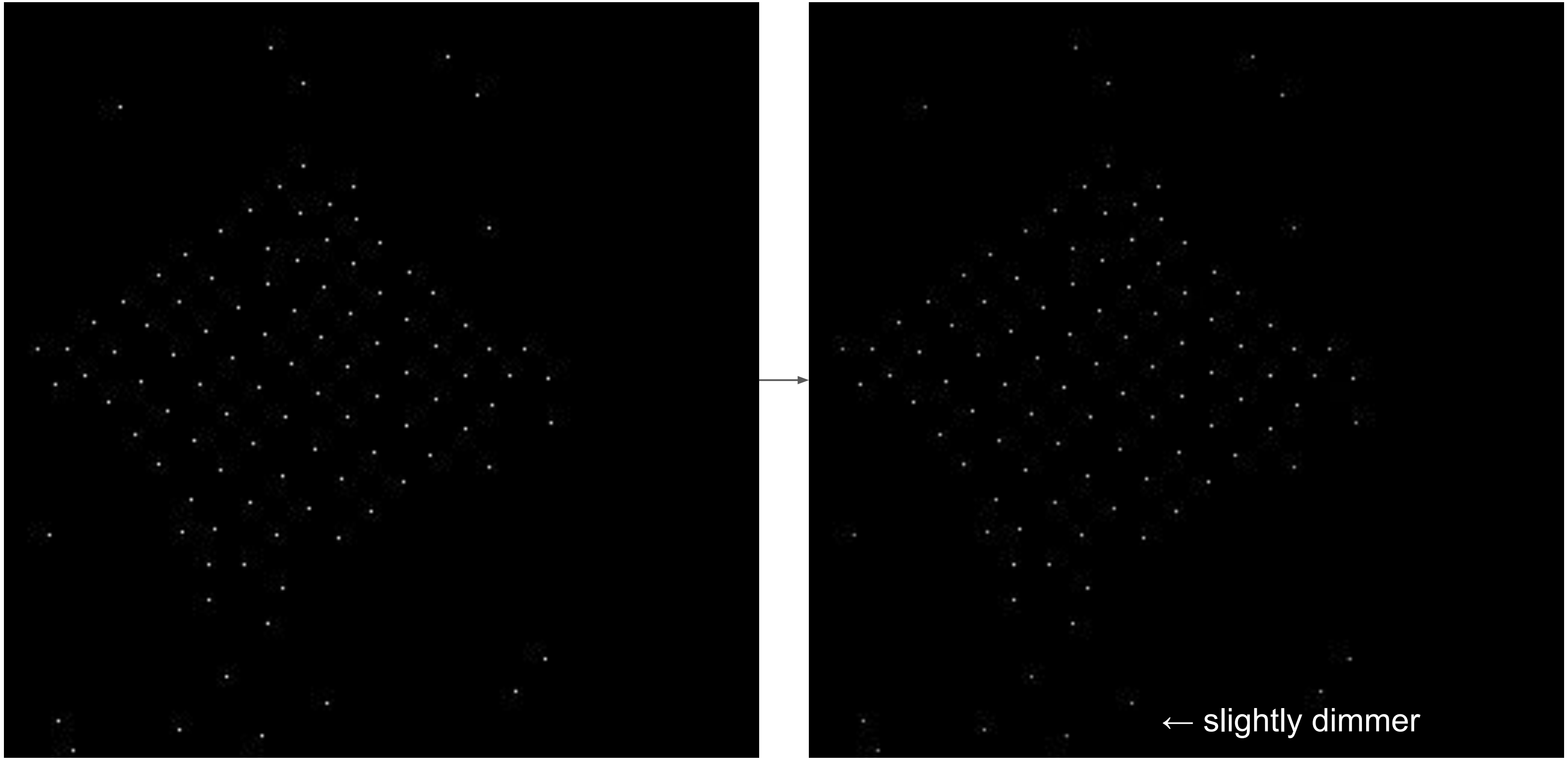
```python
# render binary image
binary = np.zeros((WIDTH, HEIGHT), np.float)
for i in corners:
    x, y = i.ravel()
    binary[y][x] = 1
cv.imwrite(f"output/{fname}/2.jpg", 255*binary)
```



Step 3: Run Neural Network

- We have a neural network trained to remove extraneous points
- See previous lecture for training information
- Takes a binary image and outputs an image of confidences

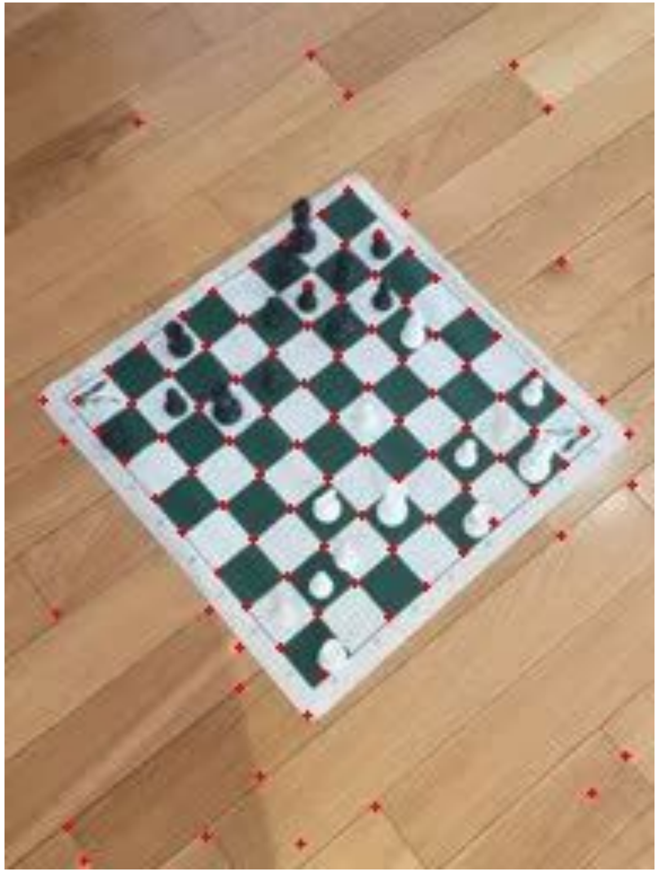
```python
# load pre-trained neural network
model = keras.models.load_model(f"models/model{WIDTH}x{HEIGHT}")
yp = model.predict(binary.reshape(1, WIDTH, HEIGHT, 1))
yp = yp[0].reshape(256, 256)
cv.imwrite(f"output/{fname}/3.jpg", tf.math.round(255*yp).numpy())
```

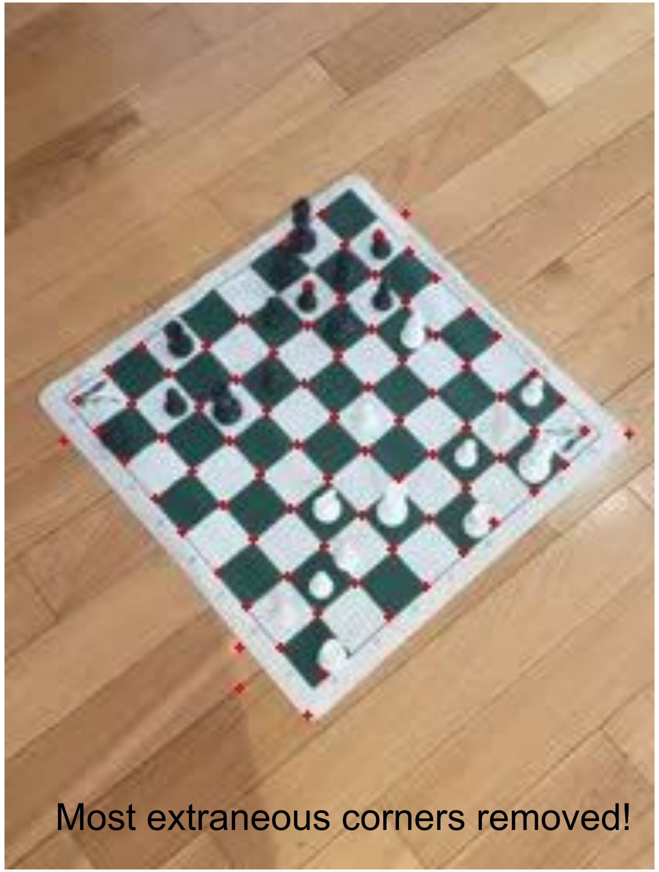

← slightly dimmer

Step 4: Extract Best Corners

- Take the top 81 corners with the highest confidence as final corners

```
# take top 81 intensities
threshold = sorted(yp.ravel(), reverse=True)[81]
points = np.array([(x, y) for x in range(WIDTH) for y in range(HEIGHT)
                   if yp[y][x] > threshold])
cv.imwrite(f"output/{fname}/4.jpg", render_points(img, points))
```




Most extraneous corners removed!

Step 5: Otsu’s Binarization

- Binarize just the chessboard
- But OpenCV’s API only runs on images, which are matrices
- Chessboard isn’t rectangular because of perspective
- Could use our own Otsu’s implementation which works on sets of pixels
- Or we could use a sampling trick

Sampling Trick

- Problem: pixels outside of chessboard included in bounding box
- Solution: re-sample these pixels according to the chessboard distribution
- This maintains the same image distribution
- Guarantees the proper threshold is picked

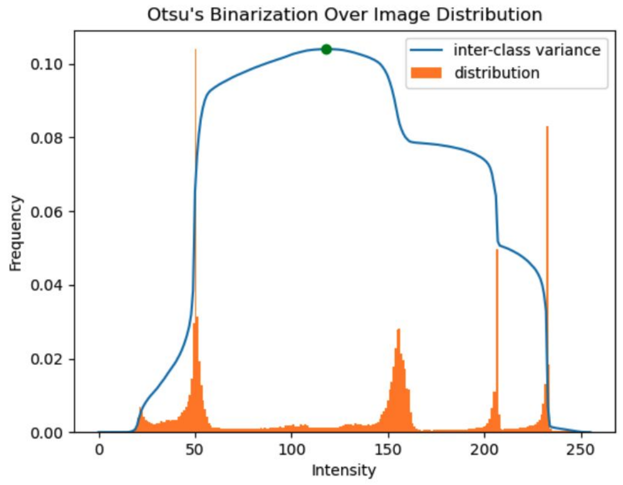
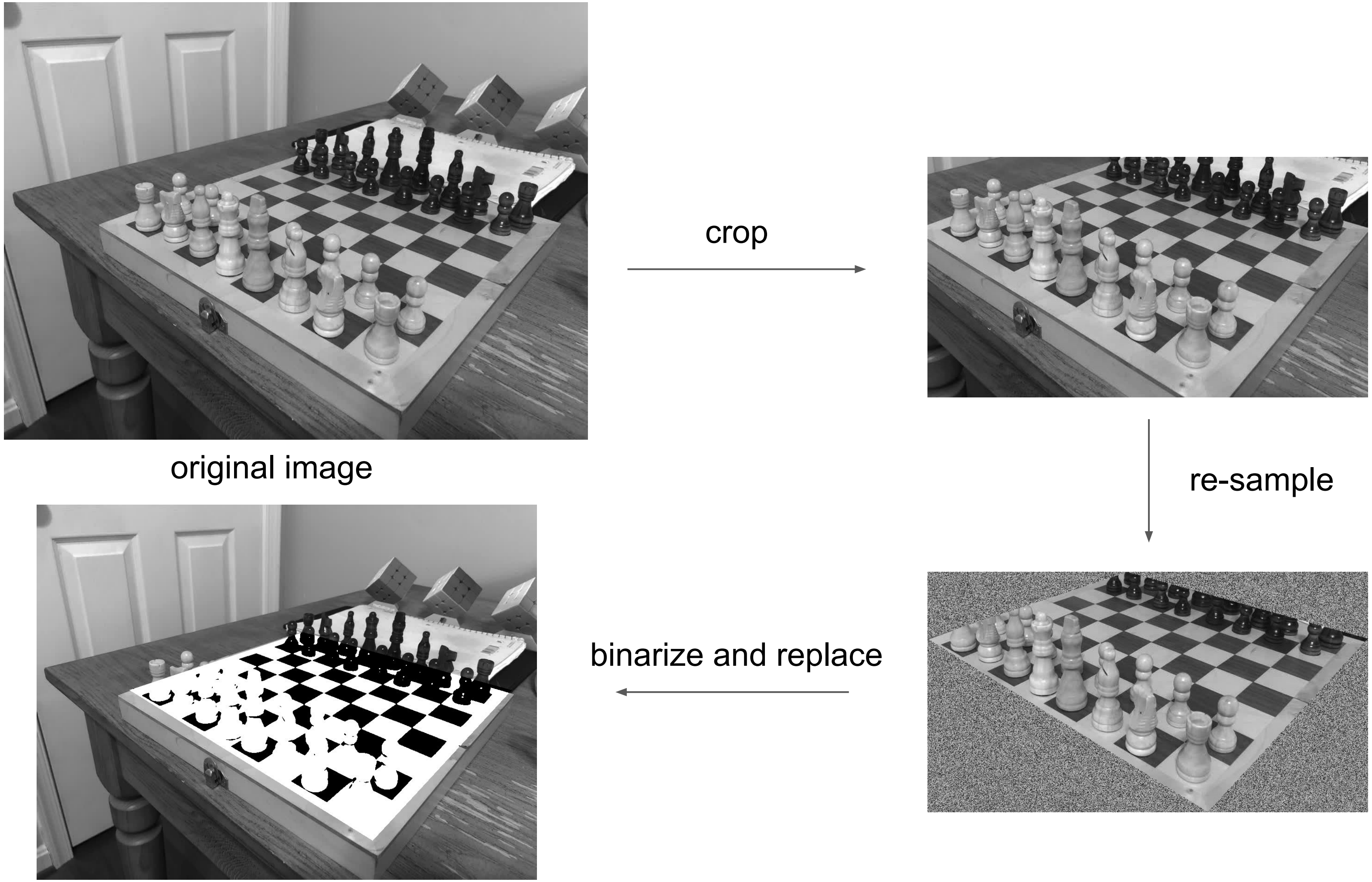
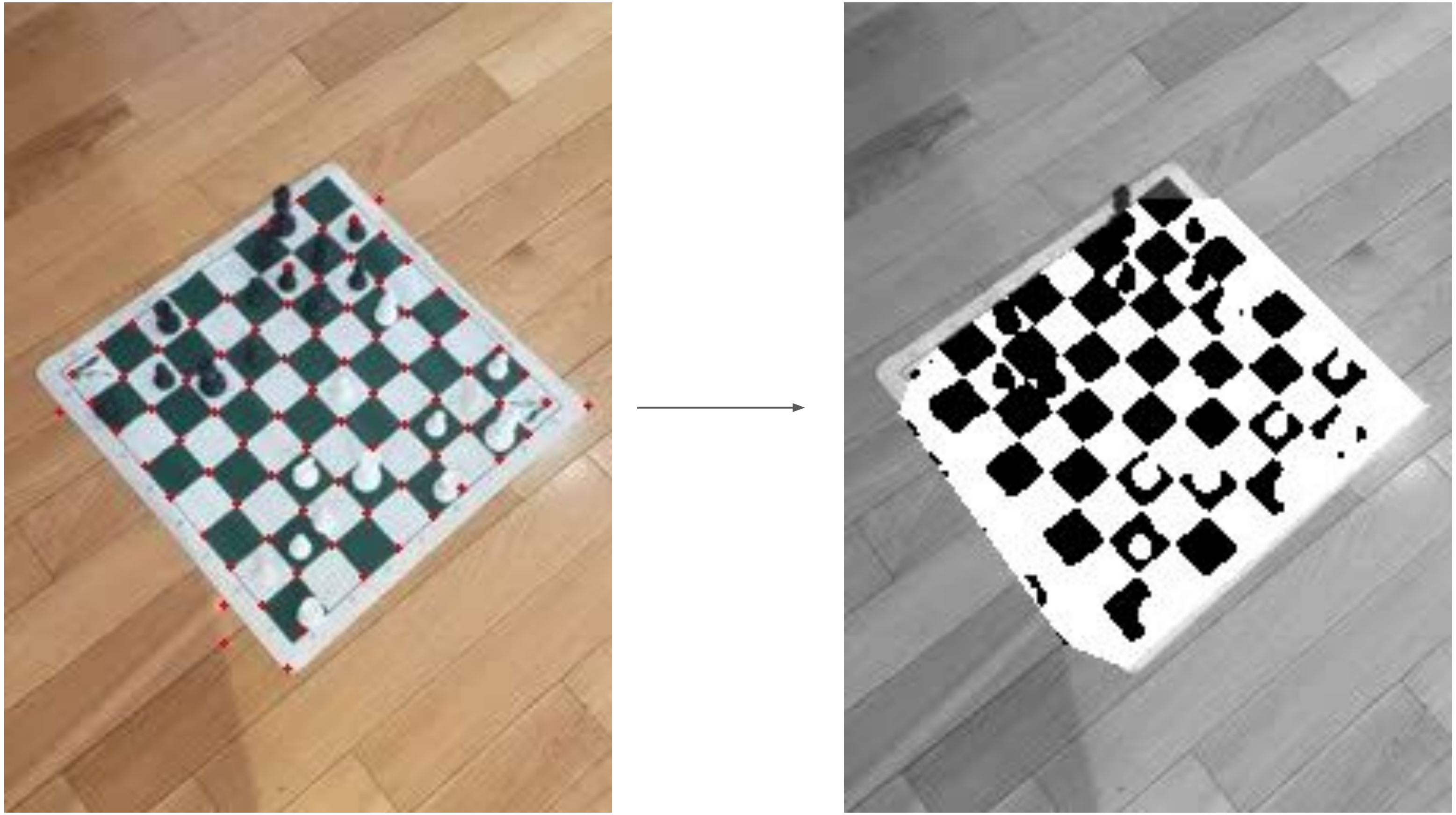


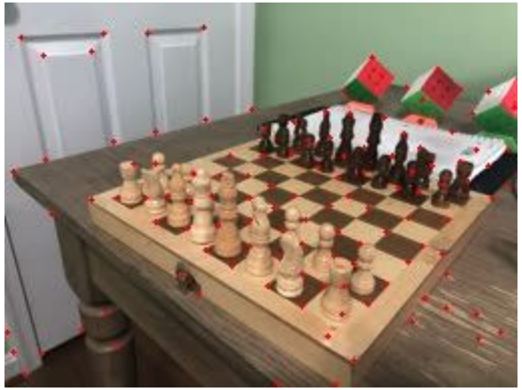
Figure: Inter-class variance over increasing threshold value.

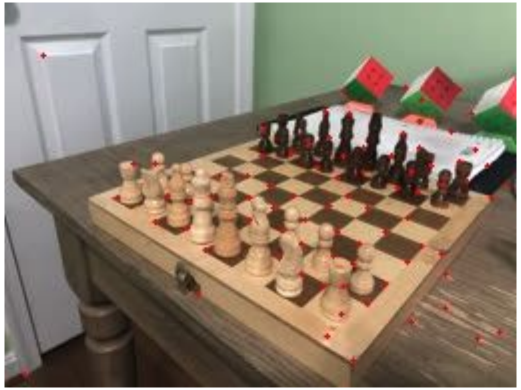

crop
original image
re-sample
binarize and replace

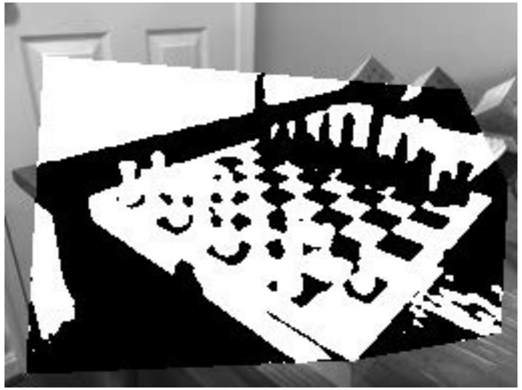
```python
def binary_iid(img: np.array, points: np.array, rect: bool=False) -> np.array:
    """ Bounding box contains extraneous pixels, sample pixels outside
    of chessboard i.i.d. from chessboard distribution which maintains
    original distribution therefore not affecting Otsu's binarization. """
    # find bounding box
    x, y = points[:, 0], points[:, 1]
    i, j, k, l = np.min(x), np.max(x), np.min(y), np.max(y)
    box = np.array(img[i:j + 1, k:l + 1])
    # find chessboard mask
    hull = cv.convexHull(points)
    mask = [[cv.pointPolygonTest(hull, (x + i, y + k), measureDist=False) >= 0
             for y in range(box.shape[1])] for x in range(box.shape[0])]
    mask, on = np.array(mask, dtype=bool), np.sum(mask)
    # generate chessboard pixel distribution
    p = np.zeros(1 << 8, dtype=np.float64)
    for v in box[mask]:
        p[v] += 1
    p /= on
    # replace pixels outside of chessboard with i.i.d. sample
    box[~mask] = rng.choice(256, np.prod(box.shape) - on, p=p)
    # apply blurring then thresholding
    blur = cv.GaussianBlur(box, (5, 5), 0)
    ret, dst = cv.threshold(blur, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    img[i:j + 1, k:l + 1][mask] = dst[mask]
    return img
```

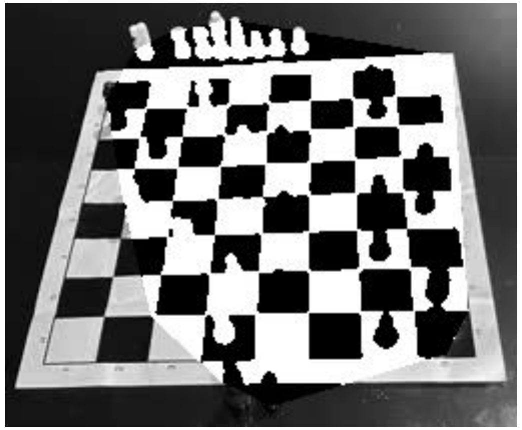


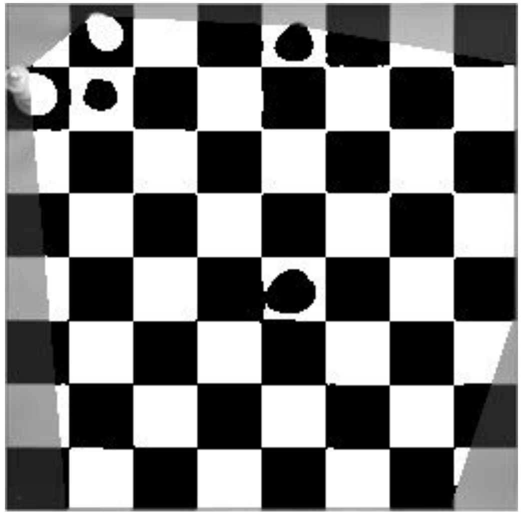
Additional Examples


initial corners


after NN


binary





References

- Implementation
- ML Chessboard Recognition Part 1
- My lecture on Otsu’s binarization
- OpenCV documentation: Shi-Tomasi Corner Detector