Introduction to Image Processing and Computer Vision

Human fingernail segmentation

Supervisor: Rafał Jóźwiak

Author: Radosław Dubiel

29 December 2020

1 Introduction

Image is a set of different pixels, some of which possess different attributes that make them belong to one segment of an image. Therefore, they contain information, which can be later used to label a particular set of pixels to create an image object. Collectively, segments should cover the entire image layer, since this process 'only' simplifies its structure, making it easier to locate some objects or regions of interest, which is particularly useful in medical imaging or video footage analysis.

This paper is a walkthrough of the human fingernail segmentation using OpenCV and some functionalities of Numpy in Python 3.8. The dataset used in this approach contains 52 images of human hands or palms in various positions under varied lighting and the same number of labels, which are masks showing desired output from each image.

Figure 1. Sample images from the dataset

After performing the segmentation on the nails, the Intersection over Union metric and Dice coefficient is calculated (using provided labels and output binary masks) in order to assess the effectiveness of the algorithm.

2 Algorithm description

2.1 Background removal

Backgrounds of the provided images in the dataset are very varied - some of them have uniform backgrounds, but many have patchy ones. Therefore, my first step is to extract a hand from the image to get rid of some objects that might interfere with the segmentation. In that case the mask of a hand is needed. To achieve that, the images are converted to YCrCb colour space. Transformation simplicity and explicit separation of luminance and chrominance components makes this colourspace attractive for skin colour modelling, provided that proper intervals are chosen. HSV colour space was taken into consideration too. However, even while regarding most optimal intervals for acceptable pixels, the effect was not as good as in the first case - the output had unwanted cavities and irregularities, often in the vicinity of the fingernail (see Figure 2.). Although both lower and upper YCrCb bounds were obtained mostly by trial and error, HSV ones were adjusted using *hsvTracker* function (see source code), which allowed a quick look-up of output image for a new pair of bounds.

Figure 2. Comparison of effectiveness of colour spaces in hand extraction (in a pair: HSV to the left, YCrCb to the right)

Later, a simple morphological operation of opening (erosion followed by dilation) is performed to erase unwanted artefacts and some noise of skin shade, which was still present in the background.

```
# converting from BGR to YCbCr colour space
img_YCrCb = cv2.cvtColor(img, cv2.COLOR_BGR2YCR_CB)
# skin colour range for YCrCb colour space
lower = (0, 135, 85)
upper = (255, 180, 135)
```

```
    YCrCb_mask = cv2.inRange(img_YCrCb, lower, upper)
    YCrCb_mask = cv2.morphologyEx(YCrCb_mask, cv2.MORPH_OPEN,
                                  np.ones((3,3), np.uint8))
```

2.2 Filling the gaps

As it can be seen in the last pair of images (Figure 2.), in some cases the shade of the fingernails is substantially different from the skin shade and the effect of hand extraction is counterproductive. To overcome that problem the idea of *flood fill* algorithm has been introduced. Provided the (0,0) pixel is a part of the background (i.e. black space), we can start flood filling it - as a result, the pixels that remain unaffected are within the hand boundary. The next step is to invert resulting mask, which should be showing missing gaps, therefore the only operation left is to sum initial mask with the inverted one. The whole process can be followed by the examples below.

Figure 3. The process of the flood-filling

```
    # invert the image mask
    threshold, image_threshold = cv2.threshold(image, 220, 255,
                                               cv2.THRESH_BINARY_INV)
    # operate on a copy of image mask
    image_floodfill = image_threshold.copy()

    # mask used to flood-filling
    h, w = image_threshold.shape[:2]
    mask = np.zeros((h+2, w+2), np.uint8)

    # flood-fill from seed point (0, 0) - actually black border
    cv2.floodFill(image_floodfill, mask, (0,0), 255);
    # invert flood-filled image
    image_floodfill_inv = cv2.bitwise_not(image_floodfill)

    # combine the two images to get the foreground
    image_out = image_threshold | image_floodfill_inv
```

To ensure proper behaviour of *flood-fill* it is needed to create thin black border around initial mask. Otherwise, white regions in the mask would restrict the flow of the filling around the whole hand. It also applies to contours drawing, because the other segments of the hand also need to be somehow accessible and aforementioned border behaves like a link.

```
    # required, otherwise contours/flood-fill will stop too early
    img = cv2.copyMakeBorder(img, 2, 2, 2, 2, cv2.BORDER_CONSTANT,
                             value=(0, 0, 0))
```

After the whole process the borders have to be removed, as later operations would cause errors about size inconsistencies - in this case it is 2 pixels wide (1 pixel wouldn't be really noticeable).

```
    h, w = image_out.shape[:2]
    borderless_img = image_out[2:h-2, 2:w-2]
```

There might appear a question, namely if morphological closing of the gaps would not perform better than flood-filling. It could, but only to some extent - because background removal was the main goal, closing operation would restore some parts of it, especially tight spaces between fingers.

2.3 Nails segmentation

In most straightforward cases, fingernails in HSV colour space had a distinct shade of orange or purple that could be extracted. Unfortunately, in the Figure 4a, the background was nearly impossible to remove, but it did not stop the algorithm to segment nails in more or less acceptable way.

a)

b)

c)

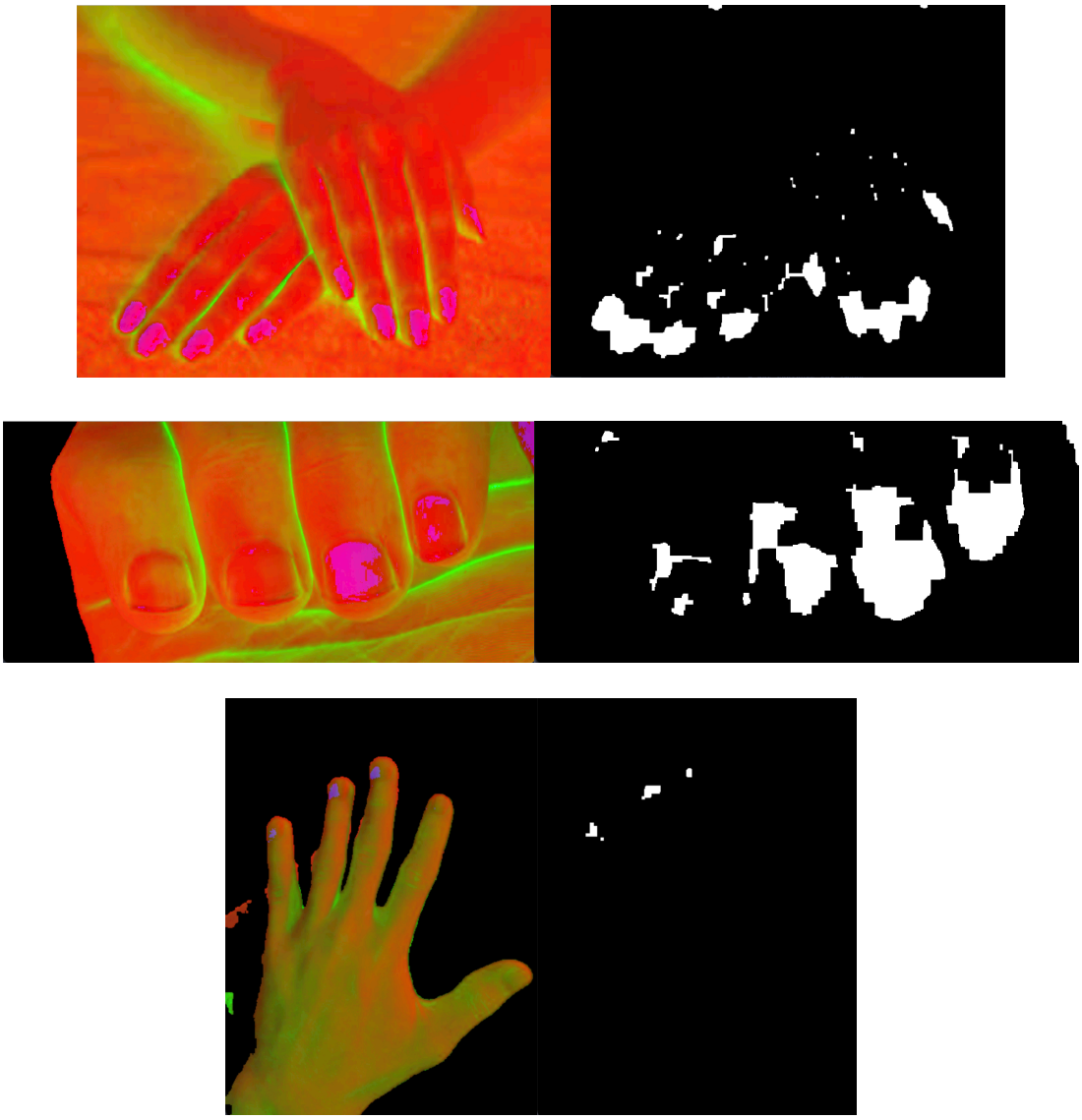

Figure 4. Nail masks extracted from the HSV colour space

The intervals were again deduced by trial and error to achieve most satisfactory IoU score. Later on, morphological closing and opening has been performed to remove unwanted mask pixels and enlarge the area of nail region.

```
        # get purplish colour
        lower = np.array([80, 0, 113])
        upper = np.array([180, 105, 255])
        mask1 = cv2.inRange(hsv_img, lower, upper)

        # get orangish colour
        lower = np.array([0, 0, 180])
        upper = np.array([7, 94, 255])
        mask2 = cv2.inRange(hsv_img, lower, upper)

        mask = mask1 | mask2

        kernel_open = np.ones((4,4),np.uint8)
        kernel_close = np.ones((7,7),np.uint8)
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel_open, iterations=1)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel_close,
                                iterations=3)
```

Below you can find the whole process for the image presented to the left, last image being the desired mask. The final IoU score in that case is only 33%.

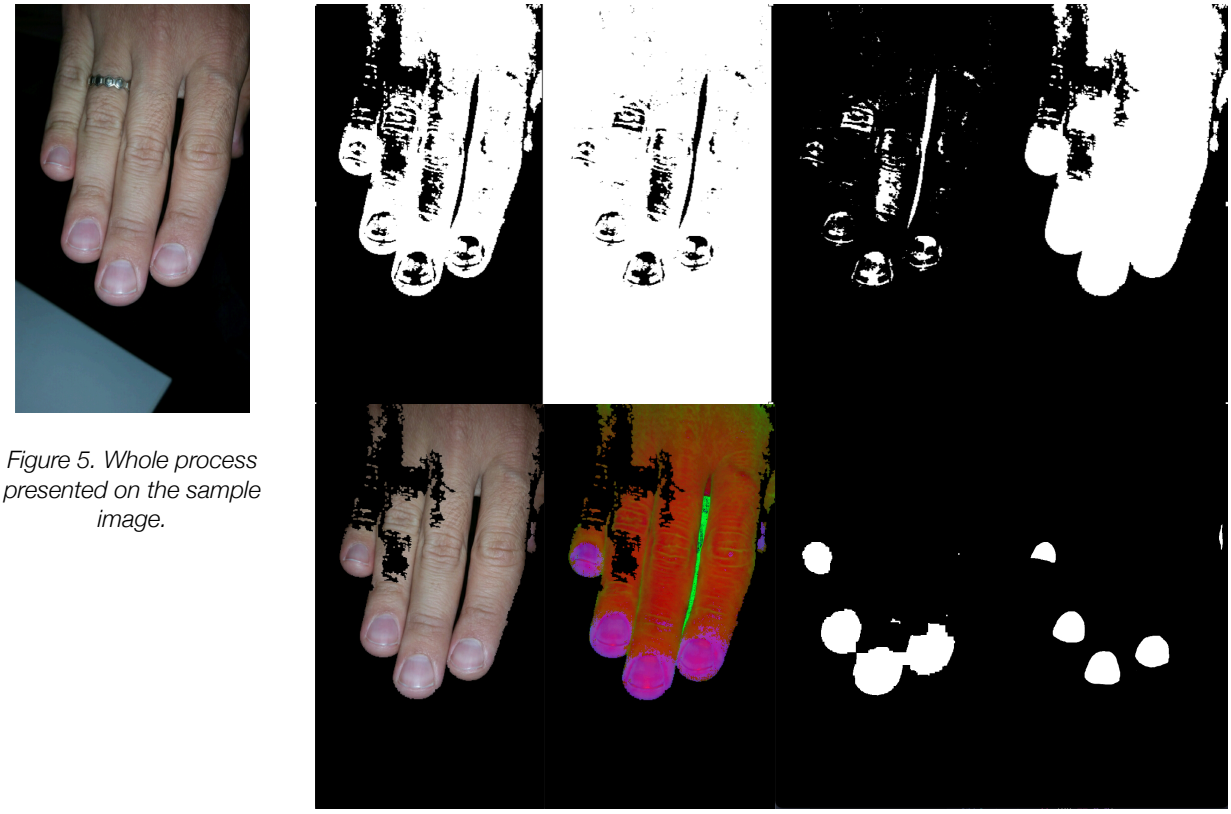

Figure 5. Whole process presented on the sample image.

There are also samples crossing the ‘satisfactory’ threshold of 0.5 IoU score, which are 0.69, 0.55 and 0.62, respectively:

a)

b)

c)

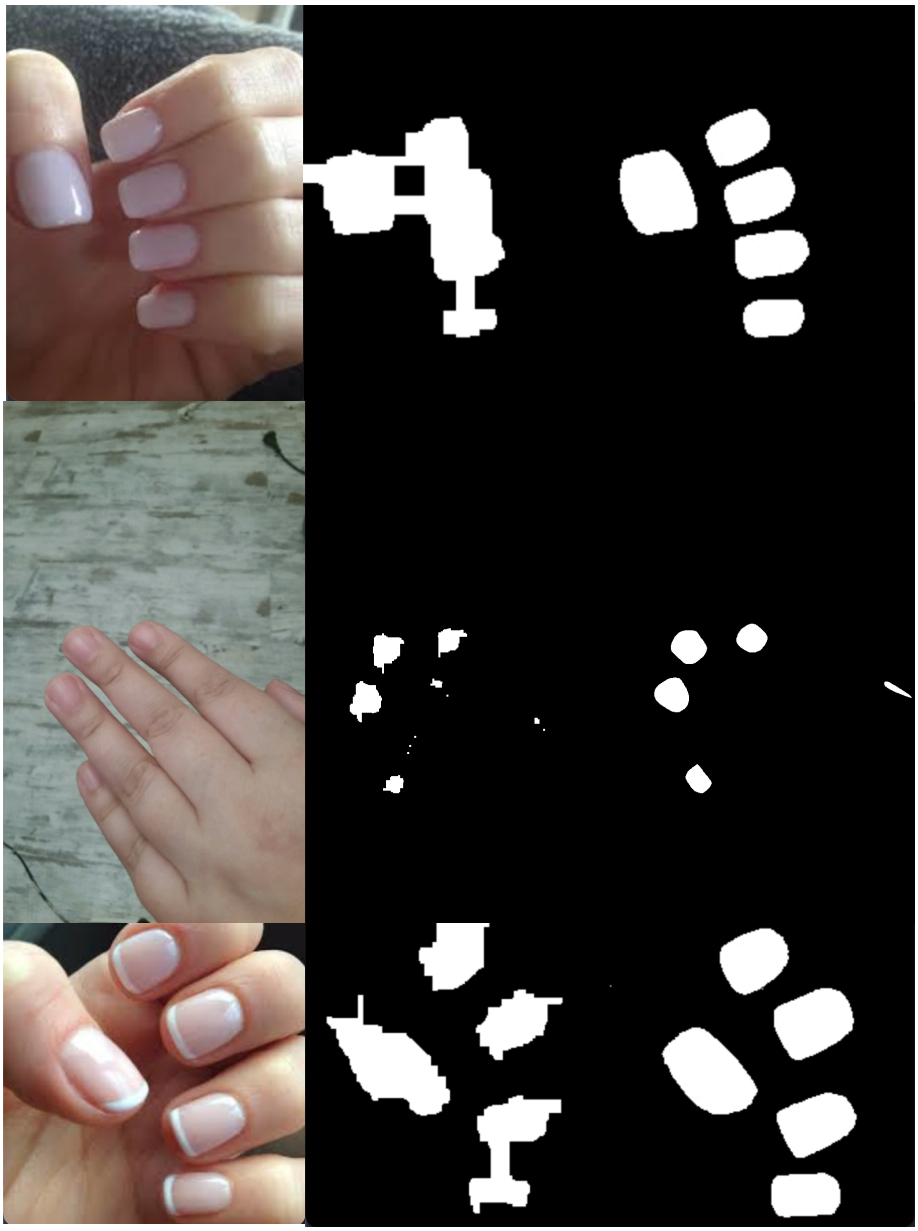

Figure 6. Satisfactory nail segmentation examples

However, there were also cases in which no segmentation could be made, because the nail part in HSV colour space was blending into the skin shade. Working with edge detection or fixing V channel and performing OTSU thresholding introduced no progress. Therefore, an experimental approach has been undertaken, because trying to extract any mask from an image with previous score of 0% could only be beneficial to the average score.

2.4 Naive segmentation

The naive approach is undertaken only if the previous attempt led to a score of less than 5%. It is about drawing circles just below the fingertips, where the fingernails should be. Unfortunately this method can only work with an image of a hand, which is loosely open, because it that case fingertips should be easy to detect.

Starting from the extracted hand image in HSV colour space, we can create a convex hull circumscribed on it. To do that, we need to get the contour that has the largest area. The main idea is to draw circles on the points, where the hull is flexing. It can be improved by calculating the centroid of the hand image, which indicates in which direction the circle should be shifted.

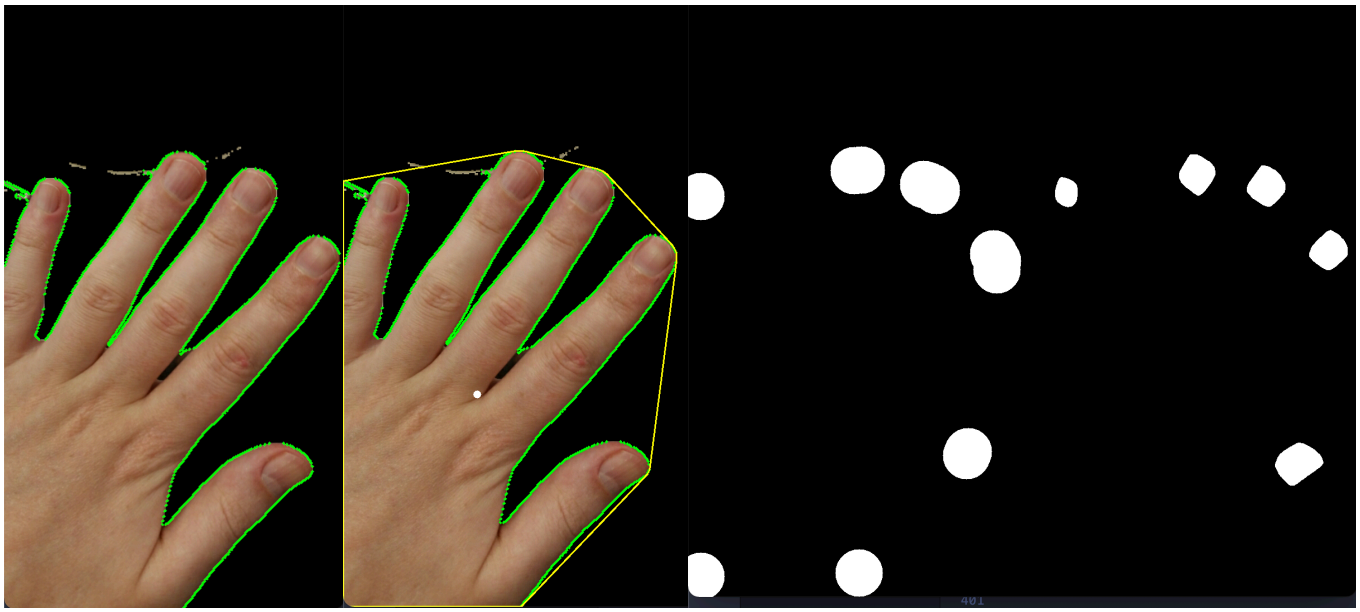

Figure 7. The example of the naive approach

In the above example the result has jumped from 0% to 30% accuracy, which is still not good. Moreover, this approach would need some additional improvement to remove unwanted circles, e.g. getting the variant of mask for which the accuracy is the highest. In some cases it leads to hilarious outputs, because we do not know how much area will the fingernails take and the convex hull has more corners than it is visible at the first sight.

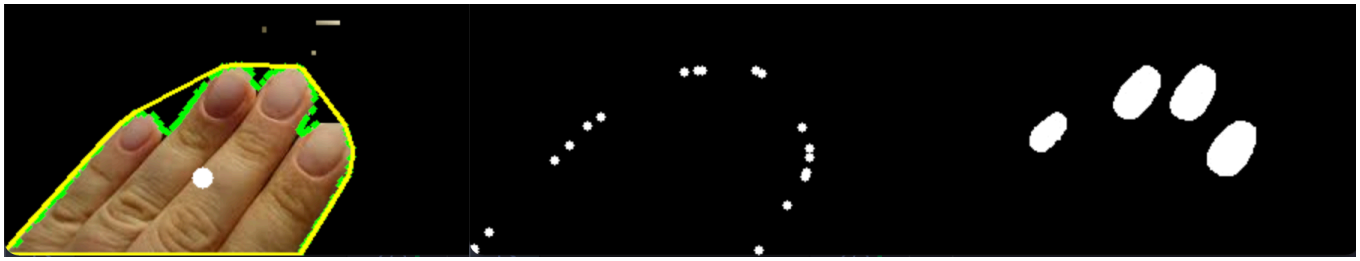

Figure 8. Another example of the naive approach

```
        # get contours for image and get the biggest contour
        gray = cv2.cvtColor(hand, cv2.COLOR_BGR2GRAY)
        contours, hierarchy = cv2.findContours(grey, cv2.RETR_TREE,
                                               cv2.CHAIN_APPROX_SIMPLE)
        cont_max = max(contours, key=cv2.contourArea)
        cv2.drawContours(hand, cont_max, -1, (0,255,0), 3)

        # get convex hull for contour
        hull = cv2.convexHull(cont_max)
        cv2.drawContours(hand, [hull], -1, (0, 255, 255), 2)

        # compute the centre of the contour
        M = cv2.moments(cont_max)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        cv2.circle(hand, (cX, cY), 7, (255, 255, 255), -1)

        height, width, channels = hand.shape

        # create black image for drawing circles
        blank = hand.copy()
        blank[:, :, :] = [0, 0, 0]
        for p in hull[2:]:
            x = p[0][0]
            y = p[0][1]
            dist = height * width / 15000
            dot_radius = height * width / 15000
            a, b = get_point_between_points(x, y, cX, cY, dist)
            cv2.circle((blank), (int(a), int(b)), int(dot_radius),
                       (255, 255, 255), -1)
```

3 Evaluation and conclusion

The results are generally not very satisfactory - the average accuracy (IoU score) of the algorithm is only **0.28** and Dice coefficient is **0.26**. The dataset that has been processed was particularly hard to deal with - various lighting, number of fingers, hands or different positions of those made the task very difficult. Many of the photos in the dataset had not uniform background and similarities between human skin and fingernail were hard to overcome. This approach could be further improved by some decimal points by testing different intervals of colour acceptance or by enhancing the aforementioned naive approach.

Needless to say, trying out different methods and approaches definitely were very developing and eyes-opening to the possibilities that image processing offers.

4 Results

The results show the name, the IoU score and Dice coefficient, respectively.

d97db2d2-18ff-456d-8d85-23bfb7109aef.jpg: 0.3, 0.61
bf93c2e2-7b5f-4108-ae85-4ef68564d418.jpg: 0.42, 0.61
34404E67-4BB8-432E-863E-C1BEF5EB37E0.jpg: 0.38, 0.61
4c490eae-e402-11e8-97db-0242ac1c0002.jpg: 0.3, 0.6
d61a32a0-db67-11e8-9658-0242ac1c0002.jpg: 0.53, 0.6
b25b81b1-d562-4891-ae4f-83a82a2b64b5.jpg: 0.09, 0.18
4c4a6402-e402-11e8-97db-0242ac1c0002.jpg: 0.35, 0.71
3493127D-7B19-4E50-94AE-2401BD2A91C8.jpg: 0.0, 0.0
09aefeec-e05f-11e8-87a6-0242ac1c0002.jpg: 0.41, 0.0
2C29D473-CCB4-458C-926B-99D0042161E6.jpg: 0.55, 0.0
ABD5AB5C-E7CA-4420-A983-2161029ECC9E.jpg: 0.05, 0.09
4c478f3e-e402-11e8-97db-0242ac1c0002.jpg: 0.01, 0.02
4c472e18-e402-11e8-97db-0242ac1c0002.jpg: 0.27, 0.54
4252e46c-e40f-4543-91ab-031917d46c5c.jpg: 0.29, 0.54
da236f3a-8c82-4c64-9a7d-9b950fd8b47e.jpg: 0.51, 0.54
af5626a5-feb4-4b2a-8c57-152e443ea3dc.jpg: 0.34, 0.54
5fad3947-76d7-4352-9329-4a92f898dd59.jpg: 0.33, 0.54
917FB1CC-E132-497F-A463-46C61A84BB40.jpg: 0.36, 0.54
C03527E3-ADED-423E-9F11-D797EA35E286.jpg: 0.0, 0.0
54108996-6DA8-48F9-93DF-7ABB92F64E03.jpg: 0.7, 0.0
feb2c029-b89c-4ce5-b208-db2114516a40.jpg: 0.52, 0.0
4c47ee66-e402-11e8-97db-0242ac1c0002.jpg: 0.31, 0.63
2c376c66-9823-4874-869e-1e7f5c54ec7b.jpg: 0.0, 0.0
865a1e90-7ad2-4ceb-b2a1-50b07875c5c7.jpg: 0.33, 0.0
4c484f8c-e402-11e8-97db-0242ac1c0002.jpg: 0.22, 0.43
4c4a0dd6-e402-11e8-97db-0242ac1c0002.jpg: 0.31, 0.62
d60153c0-db67-11e8-9658-0242ac1c0002.jpg: 0.36, 0.62
a3a73edd-1483-4413-addb-9a7264b5d853.jpg: 0.4, 0.62
d60f3ecc-db67-11e8-9658-0242ac1c0002.jpg: 0.24, 0.49
d6072ec6-db67-11e8-9658-0242ac1c0002.jpg: 0.13, 0.49
7e9f5818-4425-4d8a-808a-4673d96fa250.jpg: 0.0, 0.0
5CBCC5AC-B638-4DFA-AA7B-2464FADFF2F7.jpg: 0.33, 0.0
d63890ec-db67-11e8-9658-0242ac1c0002.jpg: 0.25, 0.0
d60a5f06-db67-11e8-9658-0242ac1c0002.jpg: 0.28, 0.0
d6358550-db67-11e8-9658-0242ac1c0002.jpg: 0.64, 0.0
d633f320-db67-11e8-9658-0242ac1c0002.jpg: 0.31, 0.0
d62b0cd8-db67-11e8-9658-0242ac1c0002.jpg: 0.37, 0.0
1eecab90-1a92-43a7-b952-0204384e1fae.jpg: 0.01, 0.01
4c46b91a-e402-11e8-97db-0242ac1c0002.jpg: 0.11, 0.01
869CDA2E-8251-4880-89D6-9409CBC416F3.jpg: 0.52, 0.01
edff1324-0b02-4471-b449-14cea27f0094.jpg: 0.47, 0.01

41d83dbb-7c39-4d91-979e-eec5ff71b265.jpg: 0.32, 0.01
9ABD7CB9-7617-4A66-8892-B335FDB0E89F.jpg: 0.0, 0.0
d6303a64-db67-11e8-9658-0242ac1c0002.jpg: 0.0, 0.0
cc25f453-04ca-4c24-92bc-f26c5a94e78c.jpg: 0.14, 0.29
964c11f9-fbe3-420e-8674-9bb356cb5d3f.jpg: 0.2, 0.29
4c48acb6-e402-11e8-97db-0242ac1c0002.jpg: 0.06, 0.29
d639e532-db67-11e8-9658-0242ac1c0002.jpg: 0.48, 0.29
d6321726-db67-11e8-9658-0242ac1c0002.jpg: 0.13, 0.29
F6F9B3E6-FA7B-4DAC-B08C-1AD19BC43A76.jpg: 0.67, 0.29
4c49b502-e402-11e8-97db-0242ac1c0002.jpg: 0.2, 0.4
d61d6de4-db67-11e8-9658-0242ac1c0002.jpg: 0.15, 0.4
average iou: 0.2822623540417669, average dice: 0.2639658800221994