COMS4036A & COMS7050A
Computer Vision

Devon Jarvis, Nathan Michlo
Dr Richard Klein

Lab 1
Due Date: 16 August 2022 @ 10:00

1 Introduction

1.1 Meet Tino

Meet Quarantino. He's been working really hard to stay productive during the lockdown. He's been exercising, catching some sun, playing some music and solving puzzles. Tino enjoys the satisfaction of finishing a puzzle (See Figure 1) but doesn't always have the time in his busy schedule to work on them.

Tino needs your help!

Throughout the semester you will be working on the various components of a system to help Tino build puzzles more efficiently.

Many different problems need to be solved to build such a system, including segmentation, shape matching and texture matching. You will apply various computational, statistical and mathematical techniques from the course to solve each of these problems. There will be several hand-ins throughout the semester and you will be expected to provide code, solutions and sometimes reports on the problems in each phase. Finally, you should combine all of these components and see whether your system can correctly solve Tino's 42 piece puzzle. Tino also has a 1000 piece puzzle that he'd like some help with, so the challenge at the end of the course will be to see whose system gets the furthest.

Figure 1: Happy but tired.

1.2 Data

Over the last few weeks, Tino has worked tirelessly collecting data (Figure 2). For the initial stages, he has taken photos of every puzzle piece individually with a top-down view so that perspective correction won't be a problem (Figure 3). Ultimately, he'd prefer that he could just take 1 photo of all the pieces spread out on the table – but he knows that you should always solve the easier problem first.

Figure 2: Collecting Data.

Figure 3: Data

1.3 Segmentation

The first problem that we will need to solve will be segmentation. Where we have to separate the pixels that belong to the puzzle piece from those that belong to the background. This means that we need a label $\in \{0, 1\}$ for every pixel in the image. We call this label image a *mask*. An example of such a mask is shown in Figure 4 where white pixels represent the puzzle piece and black pixels represent the background. Tino has labelled two pieces very carefully (Figure 5).

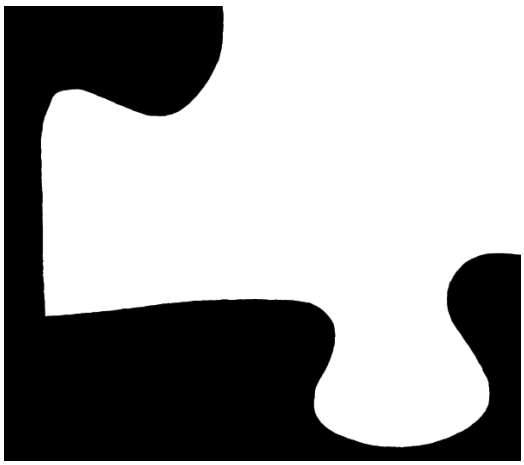

Figure 4: Portion of a mask.

Figure 5: A careful labelling process

Tino then approached the CV Class of 2020 and asked them to help label the remaining images. Each student in the class was assigned 3 images to label – this means that most images were labelled more than once. He felt that this would help us understand the quality of the labels that were received from the annotators. We'll consider this later in the course. These labelled images will form the training and validation sets.

Cleverly, Tino asked the class to save the masks in a lossless image format (PNG) which should stop the pixel values in the mask from becoming corrupted.

You will not get to see Tino's two, which will be held back as a test set for the labs that follow. A portion of your grade for this task will be allocated based on the accuracy of your segmentation algorithms on this unseen test set. So think very carefully about how to avoid overfitting throughout this series of labs.

Libraries

In this lab you need to get comfortable working with images. Tino strongly recommends using Python along with the libraries below inside a Jupyter Notebook:

- OpenCV: computer vision library (`import cv2`)
- NumPy: n-dimensional arrays and math (`import numpy as np`)
- SciPy: scientific computing and stats (`import scipy.stats`)
- scikit-image: image processing and color conversion (`import skimage`)
- ImageIO: easy image/video reading/writing (`import imageio`)
- mpmath: arbitrary precision floating point operations (`import mpmath`)
- matplotlib: plotting (`import matplotlib.pyplot as plt`)
- seaborn: matplotlib wrapper (`import seaborn as sns`)
- Python Image Library: alternative image processing (`import PIL`)

2 Instructions

You are given 3 images of *puzzle pieces* (`image-{35,83,110}.jpg`) along with corresponding *masks* (`mask-{35,83,110}.png`) which contains labels telling you which pixels belong to the puzzle piece and which pixels belong to the background. These are shown in Figure 6.

Figure 6: Puzzle Images and Masks

3 Reading and Displaying Images

Read in the images and masks. Use `matplotlib.pyplot.imshow` to view the 3 components of the images separately. Display the colour image. Does it look correct?

The colour of the images may look wrong because OpenCV uses a BGR colour space where `matplotlib`, while most other libraries use an RGB colour space. Try swapping the channel order using python slices (`img[:,:,::-1]`) or `np.moveaxis`.

Use the `skimage.color.rgb2gray` function to convert the 3 images into grayscale and display them.

Use the `skimage.color.rgb2hsv` function to convert the 3 images into the HSV colour space and display them.

4 Descriptive Statistics

For one of the 3 images (using `uint8` data type if you have read ahead):

1. What is the width of the image?
2. What is the height of the image?
3. How many pixels are in the image in total?
4. How many black pixels are there in the mask?

5. How many white pixels are there in the mask?

Using the grayscale version of the image...

6. What is the minimum pixel value in the image?
7. What is the maximum pixel value in the image?
8. What are the minimum and maximum pixel values of the puzzle pixels?
9. What are the minimum and maximum pixel values of the background pixels?
10. What is the mean pixel intensity in the image?
11. What is the mean brightness of the puzzle pixels?
12. What is the mean brightness of the background pixels?
13. What is the variance in the grayscale intensities for puzzle pixels?
14. What is the variance in the grayscale intensities for background pixels?

Using the seaborn library...

15. Display a histogram of the red pixel intensities in the image
16. Display a histogram of the green pixel intensities in the image
17. Display a histogram of the blue pixel intensities in the image
18. Repeat the previous 3 steps for one of the mask images
19. Display a histogram of the pixel intensities of all pixels in the image (across all channels)
20. Display a histogram of the pixel intensities of the pixels in the grayscale image.
21. Display the relevant histograms of the channels in the HSV image.
22. Re-plot the histograms above with Kernel Density Estimates plotted as well.

5 Bonus

1. Perform a contrast stretch of the image.
2. Perform a histogram equalisation of the image.

See how the histograms for grayscale and HSV are affected by the above operations. How are the mean and variance of the different pixels affected by the operations above?

6 Background Classifier

Be wary of the datatype (dtype) of your images being processed. Up until now, you should have been working with images with a datatype of `uint8`, however, silent errors can occur in this or later labs if you don't cast an image to `float32` from `uint8` before further processing.

Hint: `cv2.filter2D` or simple addition or subtraction are notable examples of where things can go wrong on `uint8` images. These operations will usually not automatically cast your images and integer overflow errors (Google this!) can occur.

While integer images have values in the range $[0, 255]$, float images should have values in the range $[0, 1]$. A direct cast `ndarray.astype` between data types will not preserve this relationship, you also need to divide or multiply by 255 (or even scale or truncate values if images have been processed). Have a look at `skimage.img_as_float` to automate this process when an image is read in (We recommend you nearly always do this).

1. Write a function that applies a convolution to an image with a kernel/filter, K, that is input as a parameter to the function. Pad the image so that the height and width of the image are the same before and after the convolution is applied.
2. Apply the Vertical Prewitt, Horizontal Prewitt and Laplacian filters to one of your images (use RGB values for the image, the filter should be applied to each channel separately). We will use this one image to train a background classifier.
3. Using the output from Question 2 as well as the RGB and HSV pixel values of the image (15 features in total) calculate the mean (15-dimensional vector) and covariance (15×15 matrix) of these features. The mean and covariance matrix from Question 3 are the learned parameters of a multivariate Normal distribution. We can use it to make inferences about whether the pixels from a new image belong to the background.
4. For a second image (our validation data) obtain the same features as those which were used to train our model in Question 3. Now find the value of the probability density function of the new data points being background pixels (*HINT:* `scipy.stats.multivariate\_normal.pdf()`). Note that the value of the PDF is not the probability of the pixel being a background pixel – recall that to get the probability from a PDF you need to integrate between two values. If the PDF value of a data point coming from the background of the image is above some threshold (Θ) classify it as being background. If not it belongs to the foreground. Think about how to find a good value of Θ, what threshold value gives your model the best performance? Compare your model's predictions to the mask for this image (you will have to invert black and white pixels in the image) and calculate $accuracy = 1 - \frac{1}{N}\sum_{i=1}^{N} |predict_i - test\_mask_i|$.
5. Repeat the process from Question 3 (training) and 4 (inference) except now apply the two Prewitt filters and the Laplacian filter to the HSV representation of the image. Does this improve the accuracy of the model on the validation data? For a final attempt, repeat the training and inference without the RGB pixels altogether.
6. Use the model from above which gave the best accuracy. Apply it to your third (test) image. What is its accuracy? What else could be done to improve the model's performance?
7. Analyse the performance of the previous models in detail:[1]
   (a) Calculate the confusion matrices for both models on the test image.
   (b) Based on the confusion matrices, calculate the Cohen's Kappa for both models on the test image.
   (c) Calculate the intersection-over-union of the predicted mask.
   (d) Plot the Receiver Operating Characteristic (ROC) curve and Precision-Recall curves by varying the threshold (Θ) on the probability density of the background model. How could you use this graph to find an optimal threshold?

[1] Look up how to calculate and interpret the relevant metrics if necessary.

7 Submission

You should submit a Jupter Notebook containing all the results from the questions above. They should be commented and clearly labelled (marks will be deducted if they are not).

You may work in groups of 3 for these labs. Make sure that you are able to accomplish all of the tasks individually though.

The notebook should contain the following:

1. The values for all the questions in Section 4.
2. The corrected images from Section 5.
3. The mean and covariance from Question 6.3.
4. Image containing the probability density maps from Question 6.4 and 6.5 on the validation images, as well as the accuracies. Apply a contrast stretch to the density maps so that the dynamic range of the image is $[0, 1]$.
5. The probability density map and accuracy on your test image. Apply a contrast stretch to the density maps so that the dynamic range of the image is $[0, 1]$.
6. All performance metrics and a *brief* interpretation of these results.