COMS4036A & COMS7050A
Computer Vision

Devon Jarvis, Nathan Michlo
Prof. Richard Klein

Lab 1
Due Date: 8 August 2024 @ 17:00

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 loves 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.

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).

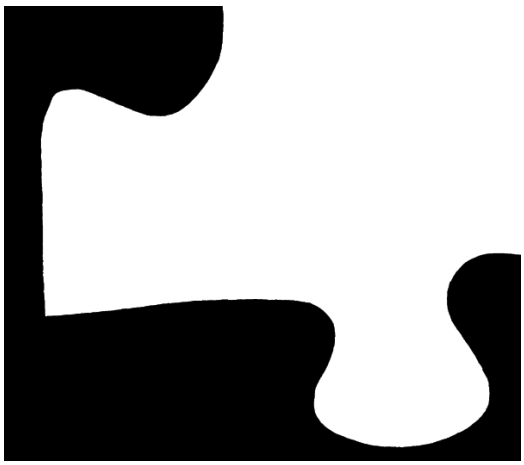

Figure 4: Portion of a mask.

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 prevent 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. Therefore, 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 inside a Juypter Notebook, along with the libraries:

- 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.

(a) Train (b) Validation/Dev (c) Test

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 the training image, `image-35` (using `uint8` data type if you have read ahead):

1. What is the width and height of the image?
2. How many white pixels are there in the mask?

Using the grayscale version of the image from Section 3...

3. What is the maximum pixel value in the image?

4. What is the maximum pixel value across all the *puzzle* pixels?
5. What is the mean pixel intensity in the image?
6. What is the mean brightness of the puzzle pixels?
7. What is the mean brightness of the background pixels?
8. What is the variance in the grayscale intensities for puzzle pixels?
9. What is the variance in the grayscale intensities for background pixels?

Using the seaborn library...

10. Display a histogram of the red pixel intensities in the image.
11. Display a histogram of the green pixel intensities in the image.
12. Display a histogram of the blue pixel intensities in the image.
13. Repeat the previous 3 steps for the mask image, what do you notice?
14. Display a histogram of the pixel intensities of the pixels in the grayscale image.
15. Display the relevant histograms of the channels in the HSV image.
16. Re-plot the histograms above with Kernel Density Estimates plotted as well.

5 Background Classifier

Be wary of the datatype of your images being processed. Up until now, you should have been working with images with a data type 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.

While integer images have values in the range $\{0, 1, \ldots, 255\}$, float images should have values in the range $[0, 1]$. You can cast the numpy array to a float (`ndarray.astype`) and then divide by 255, or use functions such as `skimage.img_as_float` to convert the images to the right data type.

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 can occur and are extremely difficult to find.

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 the training image (use RGB values for the image, the filter should be applied to each channel separately).
3. What is the difference between the output of your convolution function from question 1, and the output of `cv2.filter2D` when you apply the filters above?

   Hint: See the open cv documentation: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#filter2d
4. Using the output from Question 2 as well as the RGB and HSV pixel values of the image (15 features in total: $[\text{Prewitt}_{VR}, \text{Prewitt}_{HR}, \text{Prewitt}_{VG}, \text{Prewitt}_{HG}, \text{Prewitt}_{VB}, \text{Prewitt}_{HB}, \text{Laplacian}_{R}, \text{Laplacian}_{G}, \text{Laplacian}_{B}, R, G, B, H, S, V]$) calculate the mean ($\vec{\mu} \in R^{15}$) and covariance matrix ($\Sigma \in R^{15\times15}$) of these features. The mean and covariance matrix can be used as the learned

parameters of a multivariate Normal distribution representing the likelihood of the pixel data that we see, i.e. $P(\vec{x})$.

If we calculate these values separately over the puzzle piece pixels, $P(\vec{x}|\text{fg})$, and background pixels, $P(\vec{x}|bg)$), We can then use Bayes' rule to calculate the posterior probability that a pixel belongs to the foreground or background:

$$P(\text{fg}|\vec{x}) = \frac{P(\vec{x}|\text{fg})P(\text{fg})}{P(\vec{x}|\text{fg})P(\text{fg}) + P(\vec{x}|\text{bg})P(\text{bg})} \tag{1}$$

$$= \frac{\text{Norm}_{\vec{x}}[\vec{\mu}_{\text{fg}}, \Sigma_{\text{fg}}] \cdot \frac{N_{\text{fg}}}{N}}{\text{Norm}_{\vec{x}}[\vec{\mu}_{\text{fg}}, \Sigma_{\text{fg}}] \cdot \frac{N_{\text{fg}}}{N} + \text{Norm}_{\vec{x}}[\vec{\mu}_{\text{bg}}, \Sigma_{\text{bg}}] \cdot \frac{N_{\text{bg}}}{N}} \tag{2}$$

$$= \frac{\text{Norm}_{\vec{x}}[\vec{\mu}_{\text{fg}}, \Sigma_{\text{fg}}] \cdot N_{\text{fg}}}{\text{Norm}_{\vec{x}}[\vec{\mu}_{\text{fg}}, \Sigma_{\text{fg}}] \cdot N_{\text{fg}} + \text{Norm}_{\vec{x}}[\vec{\mu}_{\text{bg}}, \Sigma_{\text{bg}}] \cdot N_{\text{bg}}} \tag{3}$$

where $\vec{\mu}_{\text{fg}}, \vec{\mu}_{\text{bg}}$, and $\Sigma_{\text{fg}}, \Sigma_{\text{bg}}$ are the mean and covariance parameters calculated on the foreground and background pixels respectively. N_{fg} and N_{bg} are the total number of foreground and background pixels respectively, and N is the total number of pixels.

We can calculate this probability for each pixel in the image and classify pixels as foreground or background based on some threshold using $P(\text{fg}|\vec{x}) \geq \Theta$.

Hint: Use `scipy.stats.multivariate_normal.pdf`

5. Calculate the feature vectors for each pixel in our validation image. Apply Equation 3 to each pixel using the parameters we calculated on our training image. Calculate the accuracy, precision, recall, F1 score and confusion matrix of the model on the validation image. How do these values change if you adjust the probability threshold, Θ? Plot the ROC curve, ensure that you use at least 10 threshold values. What is the area under the curve (AUC) on our validation image?

6. There is a segmentation specific performance metric called the Intersection-over-Union (IoU). This measures the proportion of correctly classified pixels out of the true positives, false positives, and false negatives. When the predicted mask and the true mask are perfectly aligned, the intersection and union are the same and the score is 1. At the performance degrades, the intersection decreases, the union increases, or both, ultimately decreasing the score to 0. This accounts for various different types of errors that a segmentation model can make. We will revisit this when we study object detection and segmentation in more detail.

   Calculate the IoU score over the validation image as well.

7. Perform feature selection to pick the most important features for your model. This helps make the model efficient and effective, especially with high-dimensional data. Using too many features leads to increased data requirements (curse of dimensionality), higher computational costs, and can increase overfitting.

   Try at least three different sets of features. For each set, train the model (using Equation 3) with the training image. Then, calculate all the performance metrics from Questions 5 and 6 on the validation image.

8. Pick the best model from your feature selection process and briefly justify your choice.

9. Apply the model to the test image and calculate the performance metrics one last time.

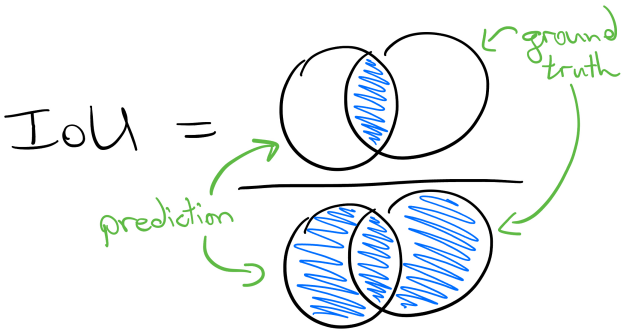

Figure 7: Intersection over Union

10. What do you notice about the model and its performance? Where does the model tend to make errors? What else could be done to improve the model's performance?

6 Submission

You should submit a Jupyter Notebook containing all the results from the questions above. The notebook should be separated into sections and numbered and clearly (marks will be deducted for poor presentation).

You may work in groups of 3 for these labs, but make sure that you understand are able to accomplish all of the tasks individually. Only one person in your group needs to submit the lab. You should create your group using the "Lab Groups" link on Moodle.

The notebook should contain the following:

1. The values for all the questions in Section 4.
2. The means and covariance matrices from Question 5.4, as well as the pixel counts used in Equation 3.
3. Image containing the original model outputs per pixel on the validation image. In other words, $P(\text{fg}|X)$ where $X \in R^{w \times h}$ is the validation image.
4. All the performance metrics applied to the original model.
5. A clear, brief summary of the feature selection and performance evaluation used to select your final model.
6. The probability density map and performance metrics on your test image.
7. Final observations of the model and model performance.

7 Conclusion

This lab provided an introduction to working with images in python. By the end, you should be able to read and display images; manipulate their datatypes; analyse and plot various properties of the images; implement and apply various linear filters as handcrafted features, and use them to train a basic statistical classifier; calculate and interpret various classification and segmentation performance metrics to evaluate models and perform basic feature selection; and interpret and analyse the results.