HIPHOP

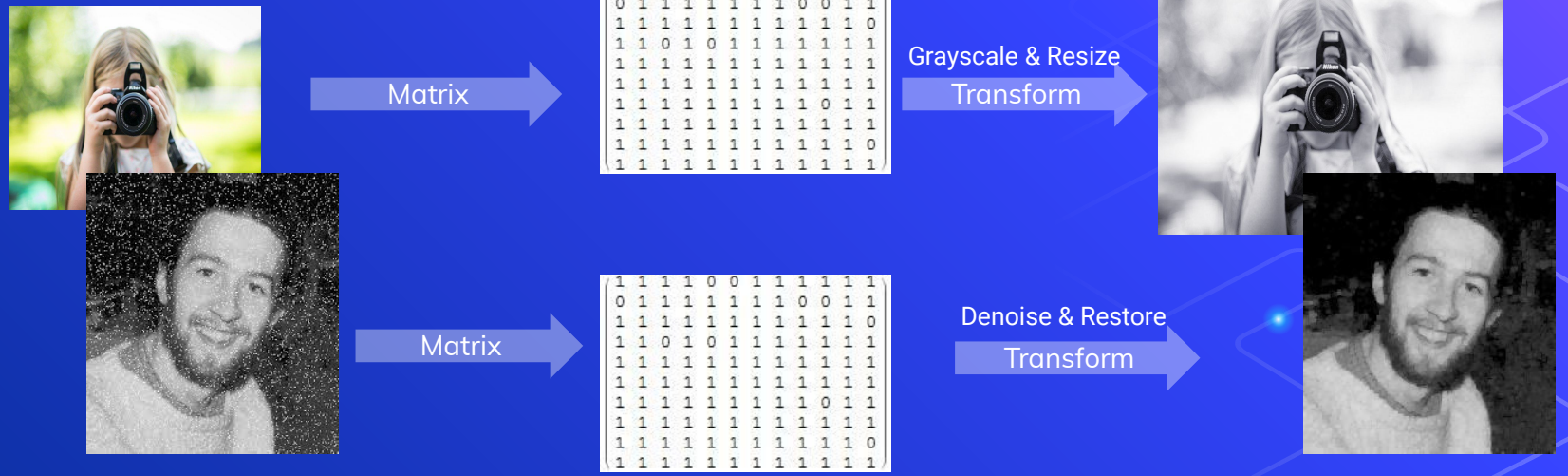
Handy Image Processing for Highly Over-caffeinated Programmers:

A DSL for Image Processing

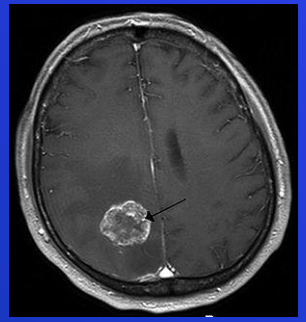
Kristen Kwong
Kalli Leung
Chrysen Park
Cindy (Yu-Hsin) Tu

Image Processing

Procedure of **converting an image into digital form** & **applying image transformations** on it.



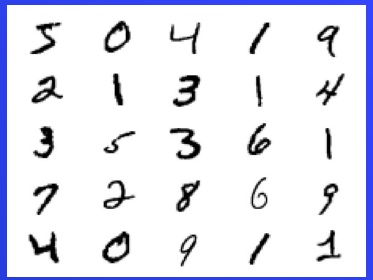
Applications



Biomedical Imaging

Ex. Contrast can be increased for better diagnosis.

Pictured above: MRI of brain tumor

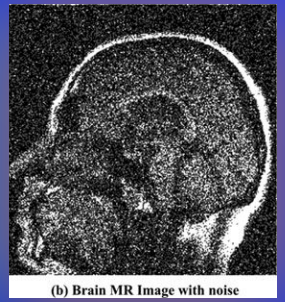


OCR (Optical Character Recognition)

Signature Recognition

Computer Vision: Edge Detection & Other Algorithms

(a) Brain MR Image without noise

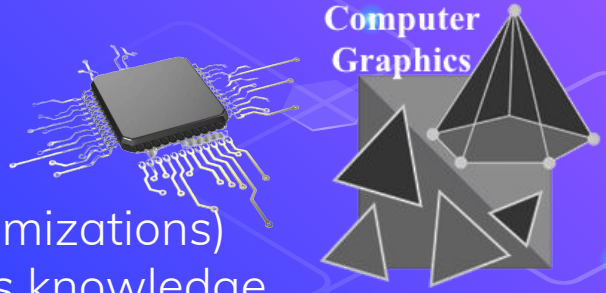


(b) Brain MR Image with noise

Machine Learning Preprocessing: Denoise Images, Filter

(ex. MRI images to detect brain tumors, old photos with static noise)

Image DSLs Now:



Hard to use:

- Focuses more on hardware (for runtime optimizations)
- Requires extensive computer vision/graphics knowledge

Impact of our Image DSL:

- Can perform image transformations easily without expert knowledge in computer imaging (abstract away complex parts)
- Good for non-domain experts to use:
  - Doctors, Medical Professionals
  - Students
  - Novice Programmers

Our DSL: hiphop-lang

Simple to use software-oriented image processing DSL

Provides the following functionalities:

- File operations (Open, save with file paths & file names)
- Simple Image Transformations
  - Colour Filter
  - Blur
  - Grayscale
  - Scale
  - Crop
- Macros for pipelines (group common image transforms into a saved macro)
- Other additional transforms (ML-preprocessing steps)

Image Processing Steps

1. Import image into the application
2. Analyse, manipulate, improve, enhance
3. Resulting image is exported

hiphop-lang usage

1 Import image into the application

```
open “filename” as var
```

Common image processing functions are built in and simplified to easily understood commands

2 Analyse, manipulate, improve, enhance

```
apply blur 10 to var
```

File operations are abstracted away for ease-of-use

3 Resulting image is exported

```
save var as “new-filename”
```

EBNF

```
<HHE> ::= open <filename> as <id>
        | save <id> as <filename>
        | apply <func> to <id>
        | apply-all [<funcs>] to <id>
        | save-macro [<funcs>] as <id>

<filename> ::= "<literal>"

<funcs> ::= <func>
          | <func>, <funcs>

<func> ::= <id> <nums>

<literal> ::= STRING
```

open filename as id: opens file specified at filename as id

save id as filename: saves id from the program at filename

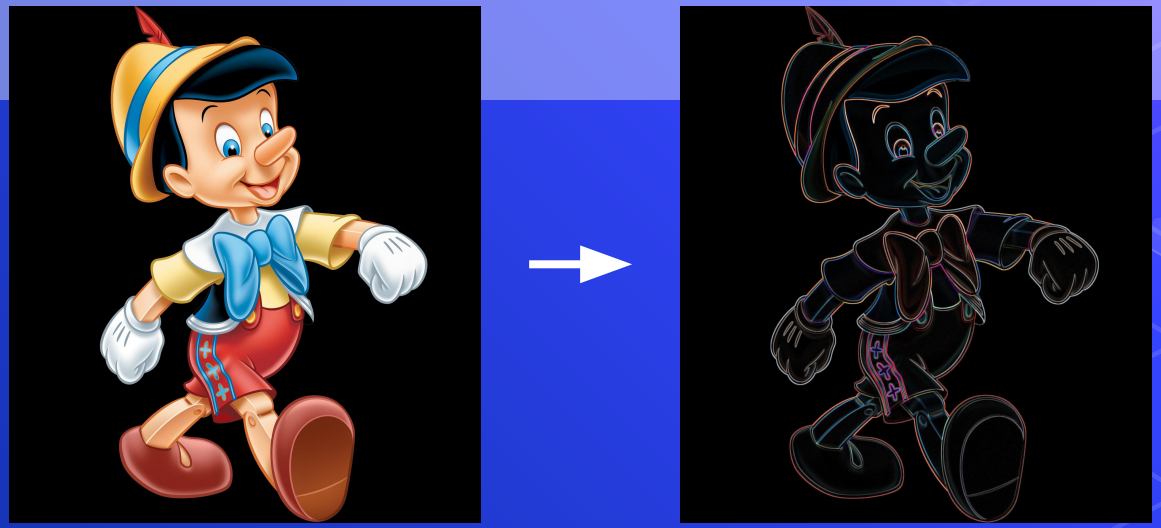
apply func to id: applies specified function to given id within the program

apply-all funcs to id: applies all functions specified to given id

save-macro funcs as id: allows saving a chain of functions as a new id in the program

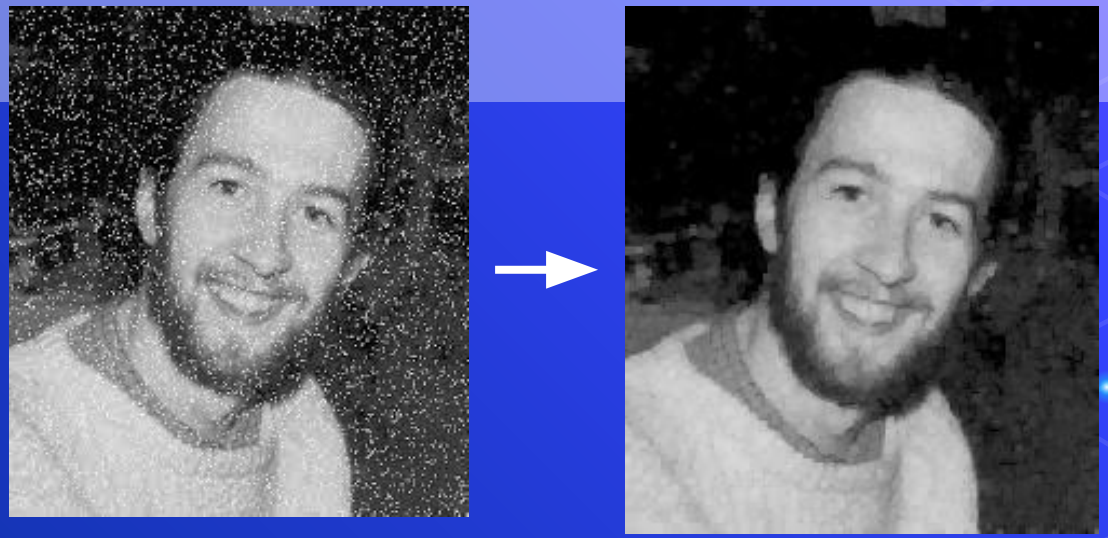
Example HIPHOP program: Edge Detection Preprocessing

```
open "pinocchio.jpg" as pic
apply outline 5 to pic
save pic as "pinocchio-outline.jpg"
```

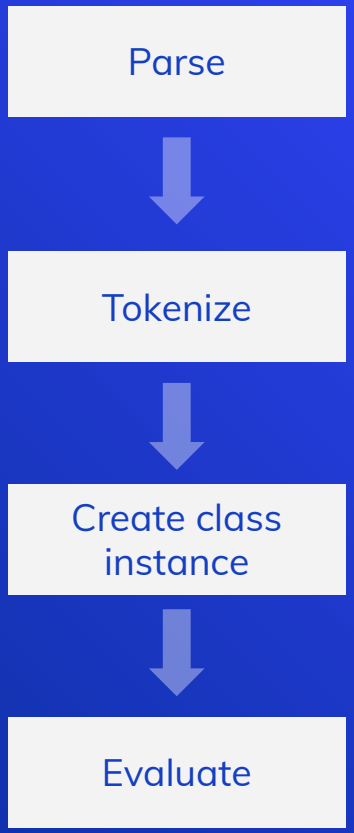


Example HIPHOP program: Denoising an Image*

```
open "man.jpg" as target
apply erode 3 to target
apply dilate 3 to target
save target as "man-output.jpg"
```



*For more info on how images can be denoised, see: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html




```
apply-all [blur 10, grayscale] to img
```

Parse each line of the program, checking for syntax errors.

```
<funcs> = blur 10, grayscale
<id> = img
```

Using regex, we obtain the variables of each command.

```
expr = apply_all_expr(funcs,id)
```

We create an instance of a class representation of each type of expression with the given parameters.

```
expr.evaluate()
```

We call evaluate, which uses the parameters from tokenization to perform the appropriate functionality.

Example Image Transform

```
def filtercolor(id, lowR, lowG, lowB, highR, highG, highB):

    img = saved_vars.get_var(id)

    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # define range of color in HSV
    lower_range = np.array([lowB, lowG, lowR])
    upper_range = np.array([highB, highG, highR])

    # Threshold the HSV image to get only specified colors
    mask = cv2.inRange(hsv, lower_range, upper_range)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(img ,img, mask=mask)
    saved_vars.add_var(id, res)
    return
```

We use OpenCV and numpy to implement our functions for evaluating apply expressions.

Function	Arguments	Functionality	Example Usage
blur	scale	Blurs the image by averaging the image with a kernel of scale*scale size	`blur 10`
grayscale		Turns the image into black and white	`grayscale`
erode	scale	Erodes the image with a kernel of scale*scale size with full of ones	`erode 3`
dilate	scale	Dilates the image with a kernel of scale*scale size with full of ones	`dilate 3`
outline	scale	Removes the noise in image by eroding and dilating the image with a kernel of scale*scale size with full of ones	`outline 5`
filtercolor	lowR, lowG, lowB, highR, highG, highB	Filters the image so only color between [lowB, lowG, lowR] and [highB, highG, highR] returns and turns rest of image black	`filtercolor 50 50 110 255 255 130`
scale	x, y	Scales the image to x*original width and y*original height	`scale 0.5 0.3`
crop	widthlow, widthhigh, heightlow, heighthigh	Crops the image with specified range, where the range of image is [-1, 1] for width and height with 0 at center Applying example on the right on an image with width 200 and height 100 would return a new image with pixels width ranged [50, 150] and height ranged [25, 75] of original image	`crop -0.5 0.5 -0.5 0.5`

Try it out via our Jupyter notebook:

https://tinyurl.com/hiphop-lang*

(run in playground - top left corner)

*Google sign-in required to view notebook

Or clone our repo to run it locally:

git clone https://github.com/kristenkwong/hiphop-lang.git

Want to learn more?

- hiphop-lang on Github:
  https://github.com/kristenkwong/hiphop-lang/blob/master/README.md
- Digital Image Processing:
  https://en.wikipedia.org/wiki/Digital_image_processing
- Image Processing Applications:
  http://www.ijesi.org/papers/NCIOT-2018/Volume-1/9.%2046-51.pdf
  https://pdfs.semanticscholar.org/7656/d3db8962a5a75d162842065319155db73af8.pdf
- Image Processing with OpenCV & Python:
  https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
- Other image processing DSLs (hardware oriented):
  RIPL: http://dx.doi.org/10.1017/S0956707006296
  IPOL: http://www.thinkmind.org/download.php?articleid=icons_2015_3_20_40047