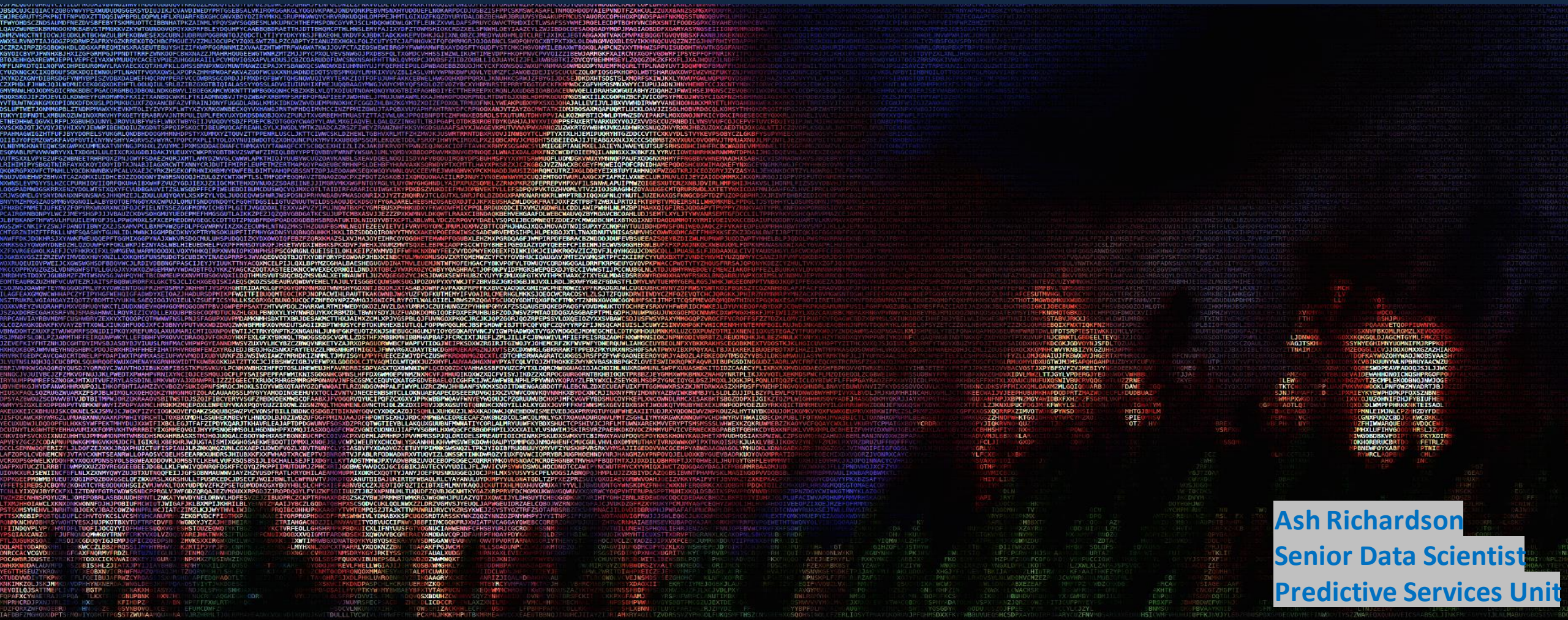
Character Recognition in < 100 Py Statements

BC Wildfire Service



Ash Richardson
Senior Data Scientist
Predictive Services Unit

Self-contained implementation "no" dependencies

"The syntactical nature of reality, the real secret of magic, is that the world is made of words. And if you know the words that the world is made of, you can make of it whatever you wish."

Terrence M (1946 – 2000) Writer, Ethnobotanist, Mystic, Philosopher (USA)

Welcome to Python online and DS CoP!

BRITISH COLUMBIA | BC Wildfire Service

- Everyone welcome
- Thank you for coming!
- Offer to do a talk or demo Please?
- Door is always open.. Feedback / input anytime pls.!
- Check out the code together session! / Experts on tap
- Please join channels and participate in discussions + events?

Overview

BRITISH COLUMBIA | BC Wildfire Service

- Explore Character Recognition -- a "core" Machine Learning task
  - Construct accessible implementation
  - Offer some baseline of ML algorithms complexity
    - "Keep it simple"
    - Flexible and scalable (to larger data)
    - Incremental springboard to support communicating other results later..
    - “base” Python with “no” dependencies
    - < 100 lines (ish)..
  - Stay as non-technical as possible
  - Keep Emphasis on discussion & relationships & connections..

Plan

- Some context
  - Why explicit, self-contained?
    - Control
    - Predictable, reliable
    - Want progs to run for 10, 20, 30+ years?
- ML Approach Dive in…
  - **1) Segment**
    - **"Flood-fill" segmentation**
  - **2) Classify**
    - **Earth Mover Distance (EMD) (ish)..**
- Visualize 1) and 2)
  - Iterate on some examples


esa

6th ADVANCED COURSE ON RADAR POLARIMETRY

This is to certify that

RICHARDSON Ashlin David

participated and completed successfully the 6th Advanced Course on Radar Polarimetry

10–14 May 2021 | ONLINE EVENT

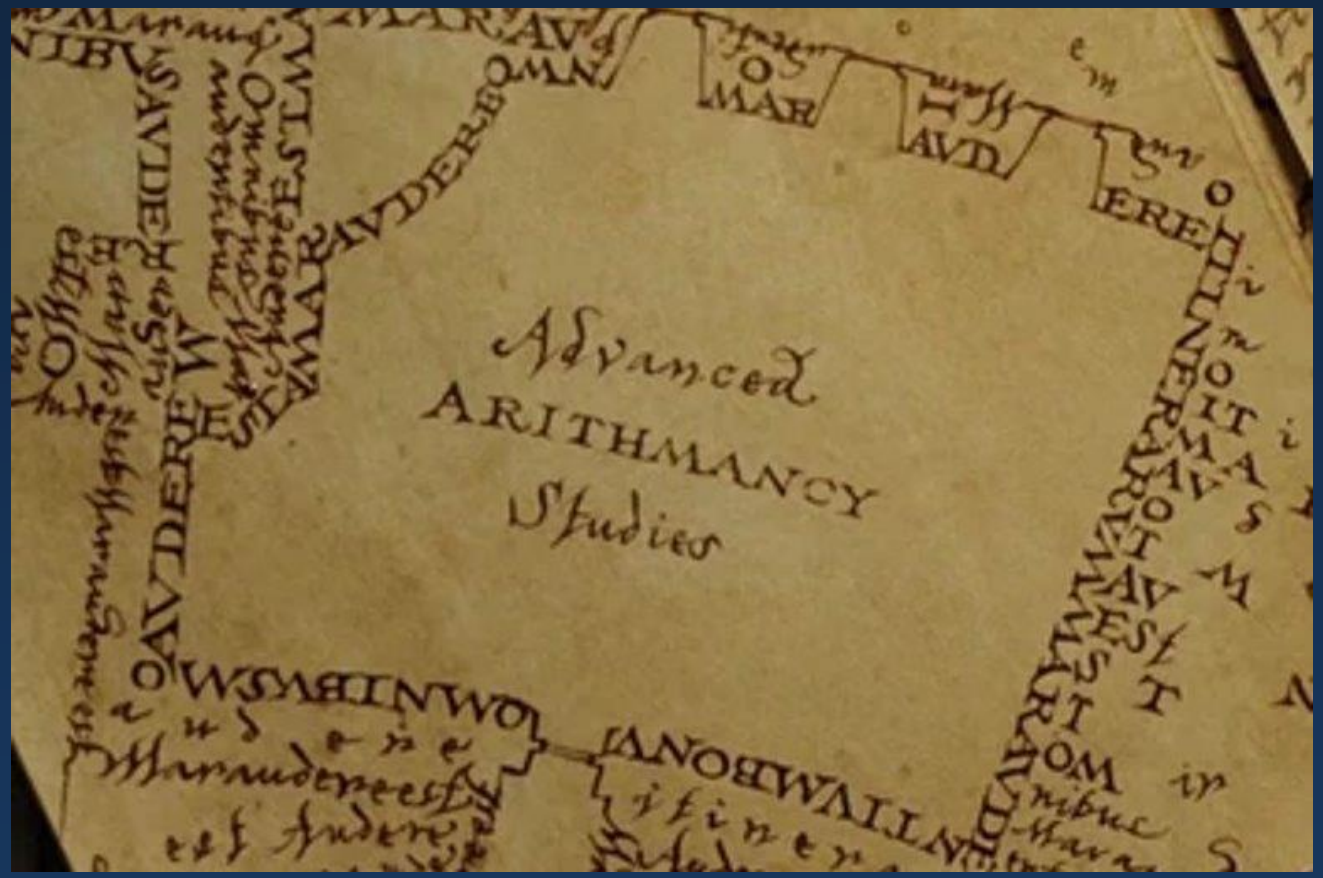
Dr. Francesco Sarti
ESA - European Space Agency
Scientific Coordinator of the Education and Training Activities
Directorate of Earth Observation Programmes

Prof. Eric Pottier
University of Rennes 1
Head Director, Institute of Electronics
and Telecommunications of Rennes

What kind of magic?

BC Wildfire Service

Building maps with symbols..



(English... transport planning for statistical predictive analytics..)

Transfiguration for **Arithmantic divination?**

CURRICULUM

MANDATORY CLASSES

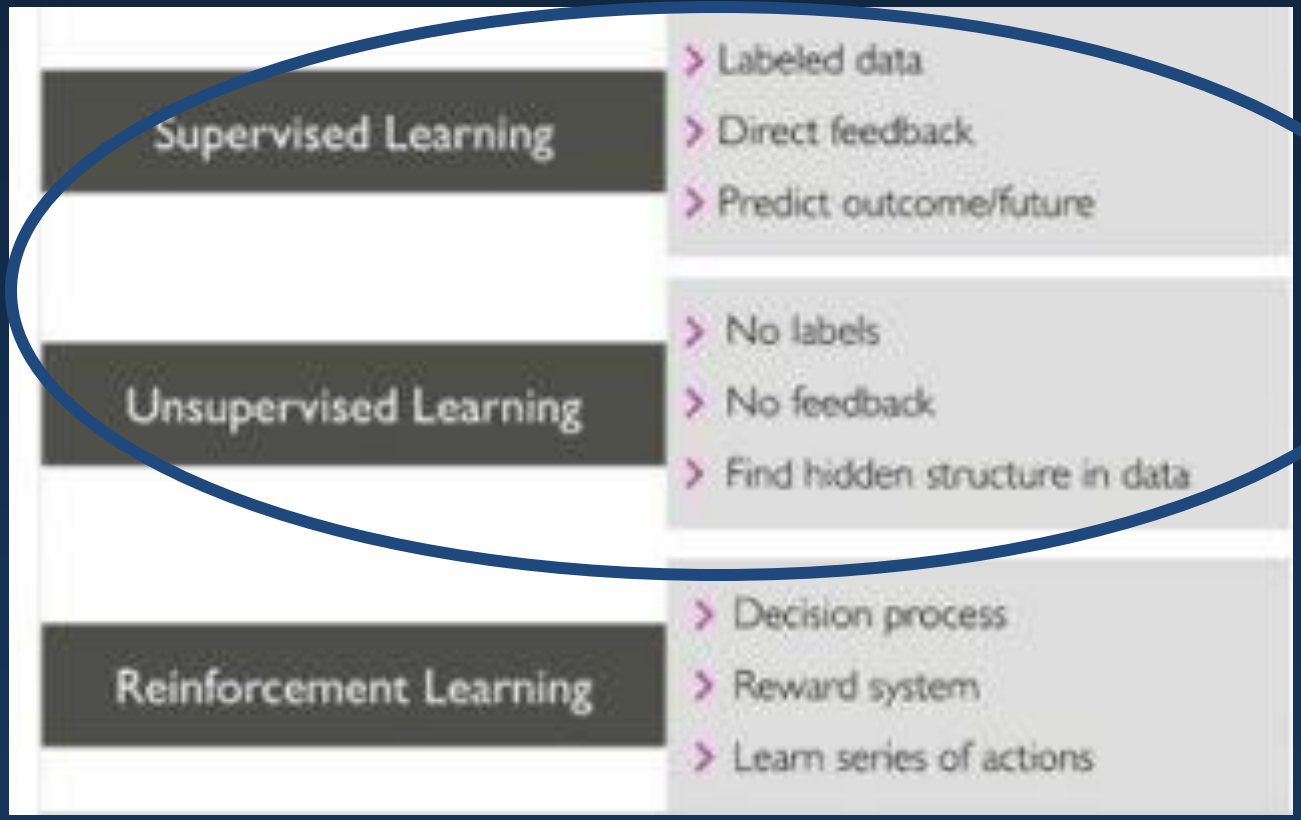
- TRANSFIGURATION
- DEFENCE AGAINST THE DARK ARTS
- CHARMS
- POTIONS
- ASTRONOMY
- HISTORY OF MAGIC
- HERBOLOGY
- FLYING (FIRST YEARS ONLY)

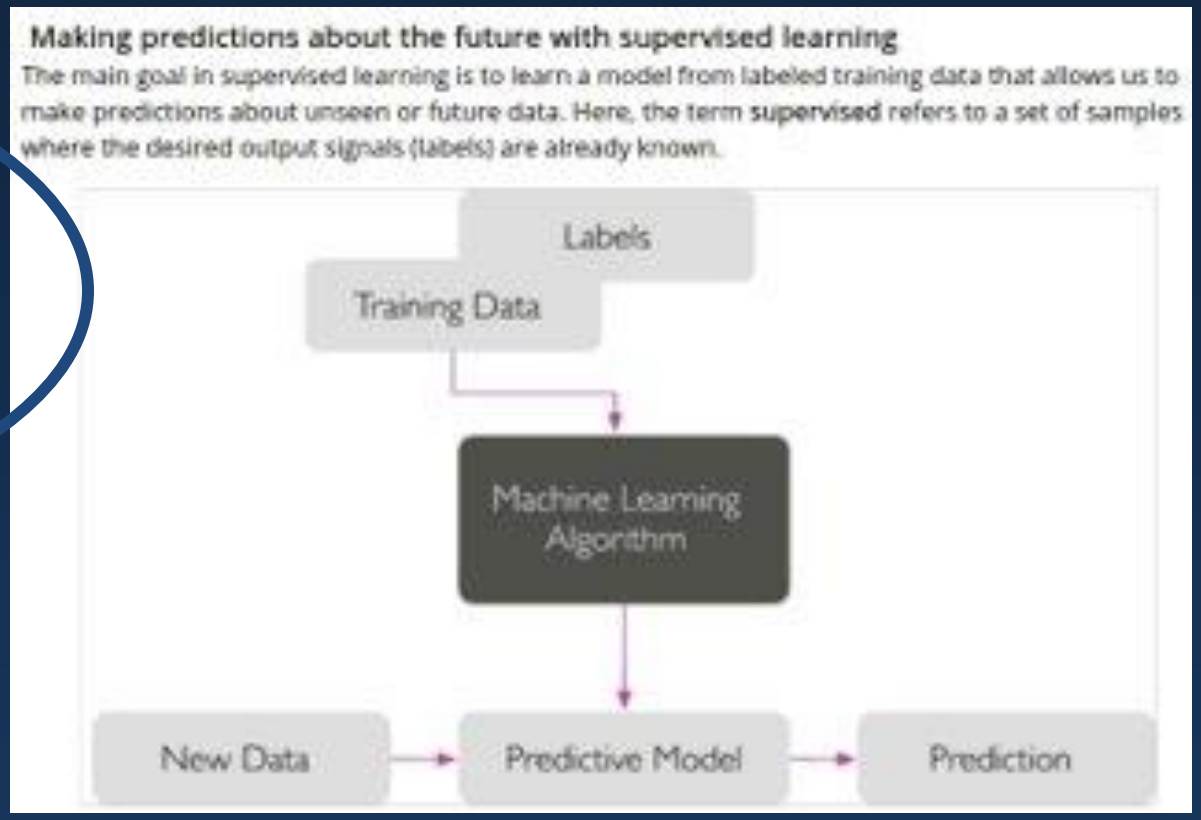
ELECTIVE CLASSES (THIRDS YEARS & UP CHOOSE TWO)

- ALCHEMY
- APPARITION
- ARITHMANCY
- ANCIENT RUNES
- CARE OF MAGICAL CREATURES
- DIVINATION
- MUGGLE STUDIES
- MUSIC

Character recognition, what kind of ML?

BC Wildfire Service





From "KDNuggets" blog

Supervised:
Get samples of labelled data,
try to label new data and get the right answer..

https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

Flow chart

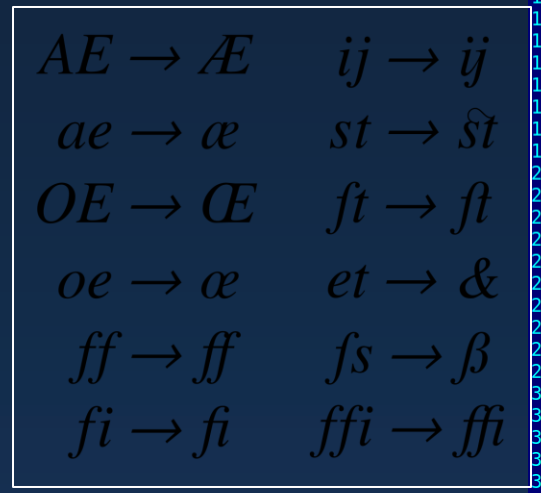
- 0) Rendering
  - Generate "raw binary" truth & test data.. LaTeX disable ligatures

- 1) Segmentation (flood fill)
  - Cut up truth & test images (read them with no driver)
- 2) Predict ("Earth mover" style distance)
  - Compare test data to training data
    - Find "nearest" truth data

0) Rendering..

- 'pdflatex'
  - Ligatures off
  - Space the characters out slightly
- 'convert' (imagemagick)
  - Convert pdf to bmp
- 'gdal_translate'
  - Convert bmp to "raw" binary

Result: beautiful images of text that Can be read without a driver..



render.py

```
''' render some text in latex, converting the typeset result to "raw binar
..which can be read without a driver'''
import os

def run(c): # run something at terminal and wait to finish
    print(c)
    a = os.system(c)

def render(my_text, name):
    run('mkdir -p ' + name)

    # use "Computer Modern" font by Donald Knuth # insert Knuth quotes..
    open(name + '.tex', 'wb').write(('\n'.join(['\\documentclass{letter}',
                                                '\\usepackage{lmodern}',
                                                '\\usepackage[T1]{fontenc}
                                                '\\usepackage{xcolor}',
                                                '\\usepackage[tracking=tru
                                                '\\DisableLigatures{encodi
                                                '\\pagenumbering{gobble}',
                                                '\\begin{document}',
                                                '\\color{blue}'] +  # blue
                                               ['\\textls{'] + my_text +

    if not os.path.exists(name + '.bin'): # delete train.bin to start from
        run('pdflatex ' + name + '.tex') # render with LaTeX
        run('convert -background white -density 333 ' + name + '.pdf ' + n
        run('gdal_translate -of ENVI -ot Float32 ' + name + '.bmp ' + name

        if os.path.exists(name + '.hdr'):
            d = open(name + '.hdr').read() + 'band names = {red,\ngreen,\n
            open(name + '.hdr','wb').write(d.encode())

truth = [] # truth characters

if __name__ == "__main__":

    def chars(i, j):  # add chars between ascii codes i, j to truth data
        global truth
        my_chars = [chr(x) for x in range(i, j)]
        truth += my_chars
        print("my_chars", my_chars)
        return ' '.join(my_chars)

    if not os.path.exists('truth.bin'):
        render([chars(48, 58) + '\n', chars(65, 91) + '\n', chars(97, 123)
                'truth')  # designate as truth

        print('+w truth_chars.txt')
        open('truth_chars.txt', 'wb').write((''.join(truth)).encode())  #

    if not os.path.exists('test.bin'):
        print("render test data..")
        # render(["hello world"], 'test')

        render(["Through three cheese trees\\ \\\\",
                "three free fleas flew\\ \\\\",
                "While these fleas flew\\ \\\\",
                "freezy breeze blew\\ \\\\",
                "Freezy breeze made\\ \\\\",
                "these three trees freeze\\ \\\\",
                "Freezy trees made\\ \\\\",
                "these trees cheese freeze\\ \\\\",
                "Thats what made these\\ \\\\",
                "three free fleas sneeze\\ \\\\"],
                'test')
```

How to read image data without driver

BC Wildfire Service

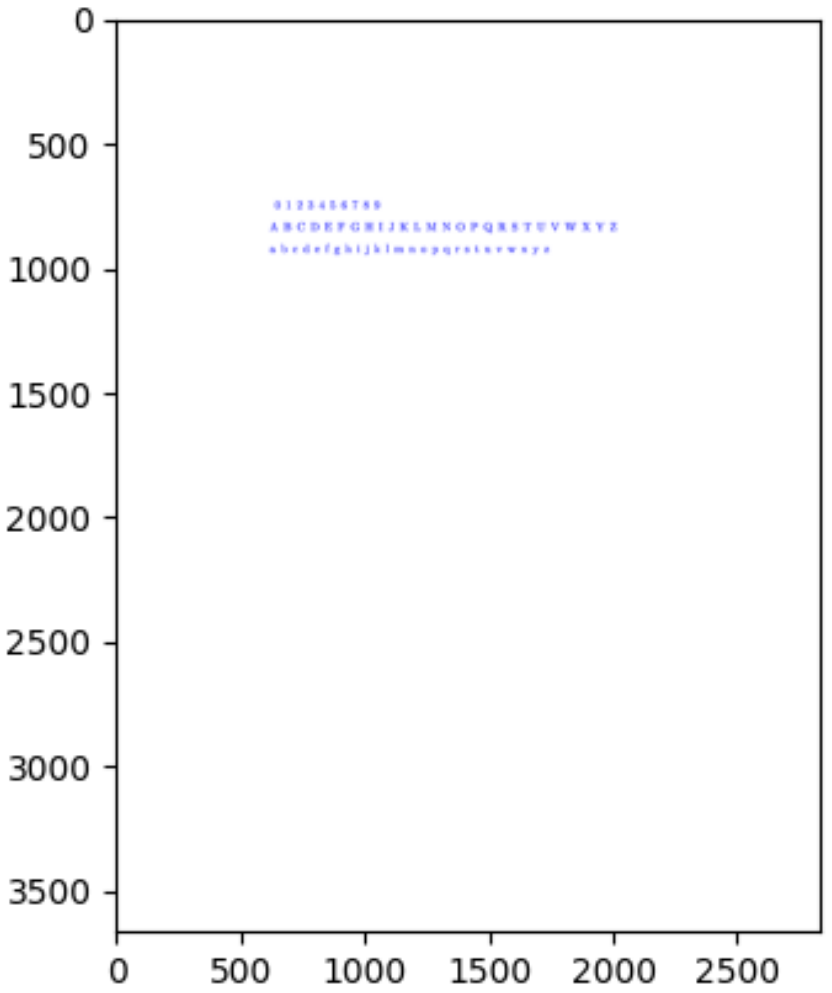
```
def read_hdr(hdr):  # read the image dimensions
    cols, rows, bands = 0, 0, 0
    for line in open(hdr).readlines():
        chunks = line.strip().split('=')
        try:  # pull off two chunks delimited by '='
            f, g = [x.strip() for x in chunks[0:2]]
            if f == 'samples':
                cols = g
            if f == 'lines':
                rows = g
            if f == 'bands':
                bands = g
        except:
            pass
    return [int(x) for x in [cols, rows, bands]]  # string to int


def read_float(fn):  # read the raw binary file
    return np.fromfile(fn, dtype=np.float32) / 255.  # put data in range [0, 1]


'''pixel @ (row, col) = (i, j):
npx = nrow * ncol # number of pixels in image
red value: dat[          i * ncol + j]
grn value: dat[    npx + i * ncol + j]
blu value: dat[2 * npx + i * ncol + j]'''
```

truth.hdr

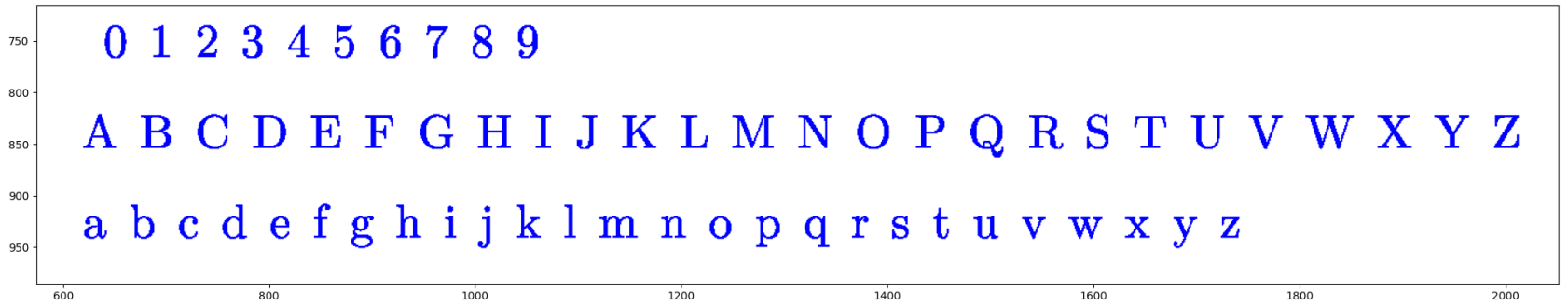
ENVI
samples = 2831
lines = 3663
bands = 3
header offset = 0
file type = ENVI Standard
data type = 4
interleave = bsq
byte order = 0
band names = {red,
green,
blue}

"flat file" of image world? Lots of people prefer geotiffs..

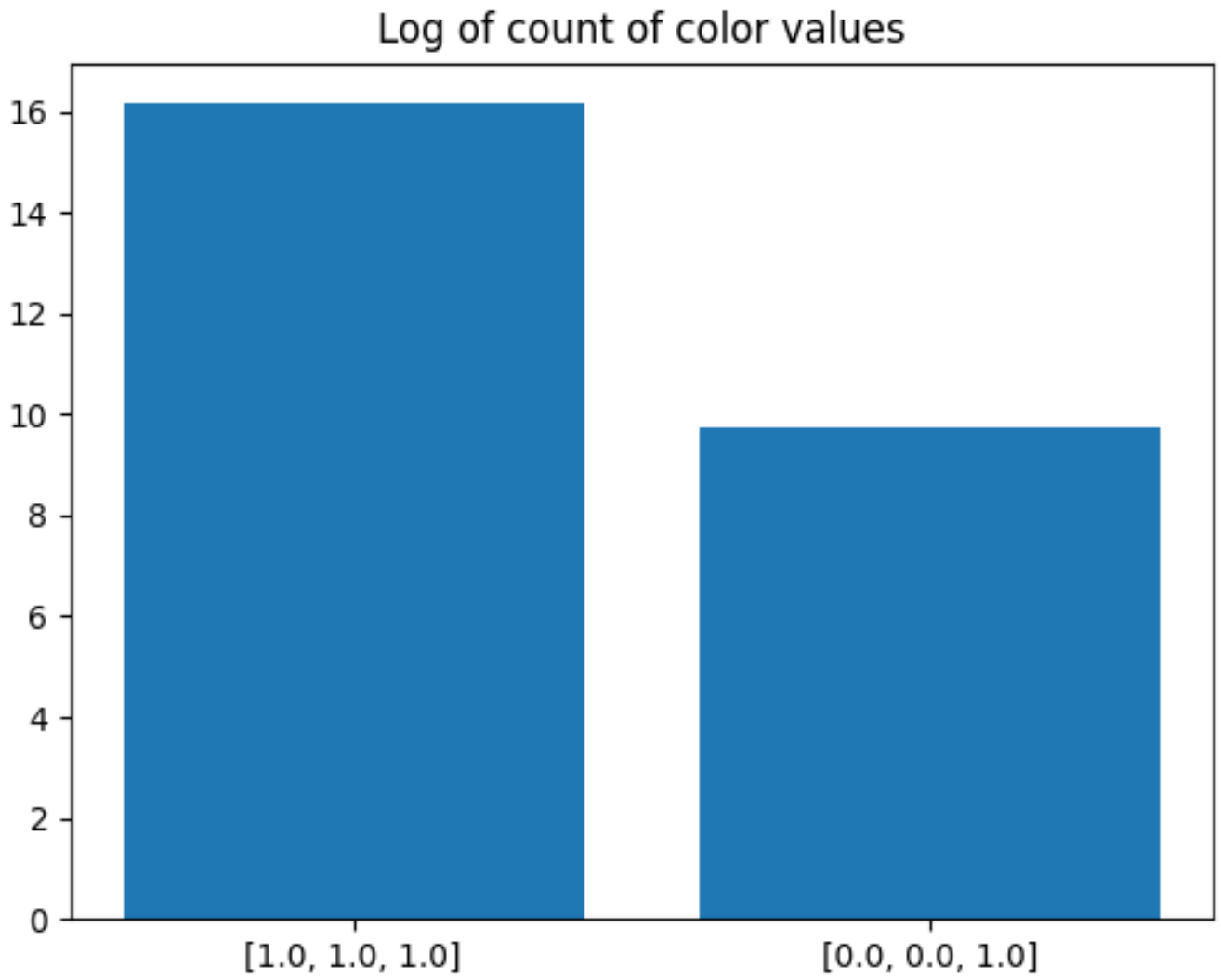
Truth



Truth


0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
750
800
850
900
950
600
800
1000
1200
1400
1600
1800
2000

Truth

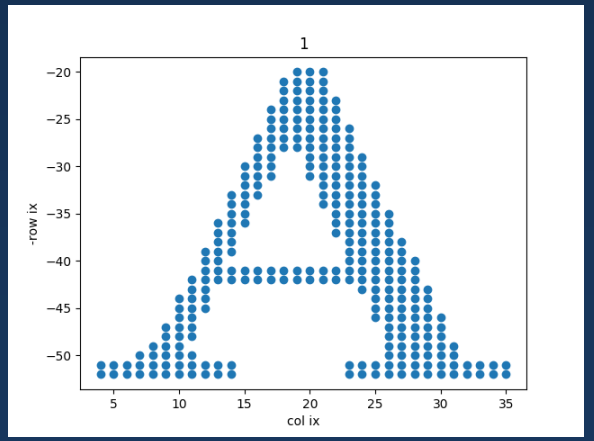


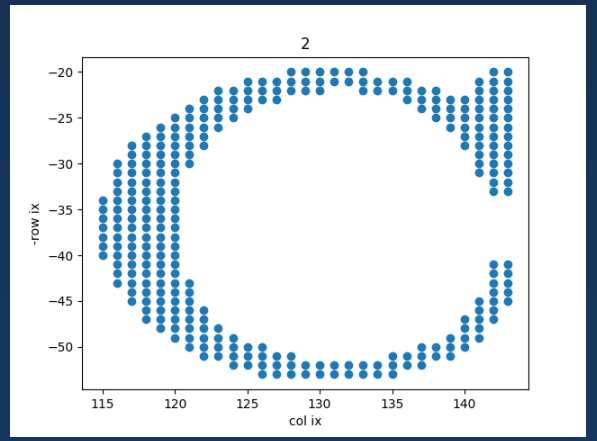
[1,1,1] is white
[0,0,1] is blue

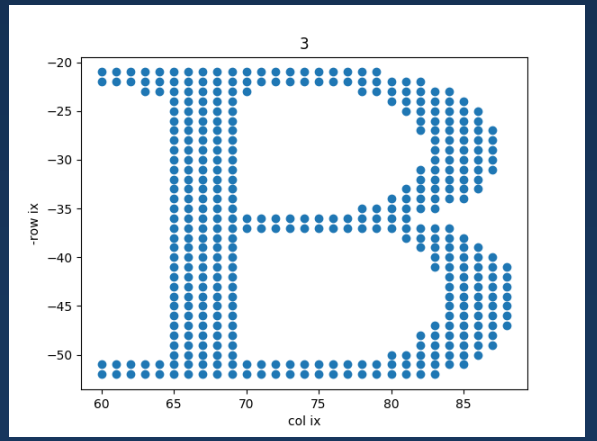
Flood fill


```
# Recap the last step (render.py does this for us)
gdal_translate -of ENVI -ot Float32 abc.bmp abc.bin
Input file size is 156, 75
0...10...20...30...40...50...60...70...80...90...100 - done.
```

```
$ python
>>> from image import image
>>> a = image("abc.bin")
>>> a.segment()
```







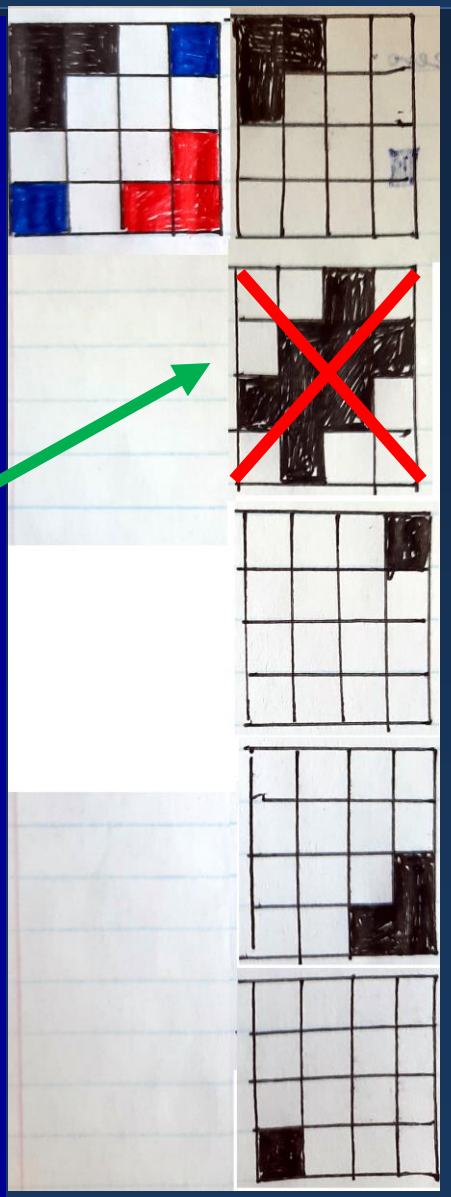
Point cloud representations

Why does B come last?

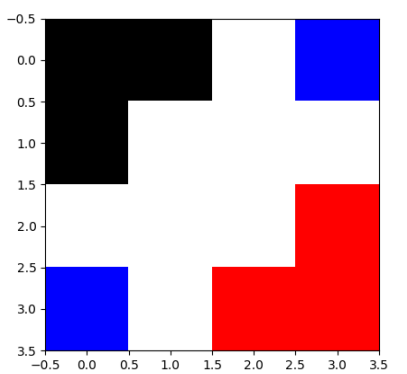
Step 2. Flood fill...

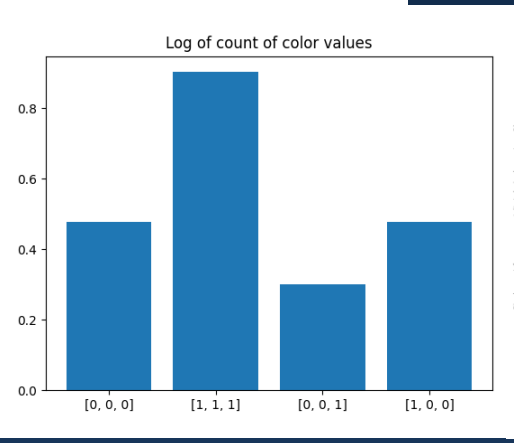
BC Wildfire Service

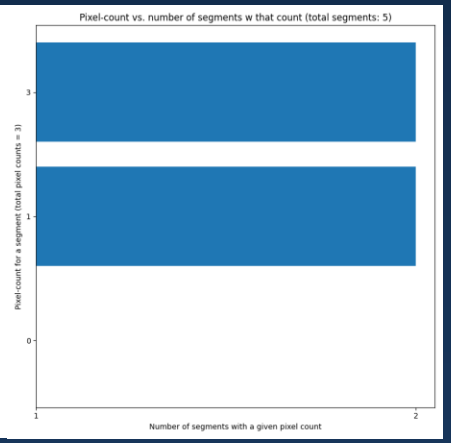
```
'''flood fill segmentation'''
def flood(img, i, j, my_label=None, my_color=None):
    if i >= img.rows or j >= img.cols or i < 0 or j < 0:
        return  # out of bounds
    ix = i * img.cols + j  # linear index of (i, j)

    if img.labels[ix] > 0:
        return  # stop: already labelled

    c_s = str(img.rgb[ix])
    if c_s == img.max_color:
        return  # stop: ignore background "background subtraction"
    if my_color and my_color != c_s:
        return  # stop: different colour than at invocation chain start

    if my_label:
        img.labels[ix] = my_label
    else:
        img.labels[ix] = img.next_label
        img.next_label += 1

    for di in [-1, 0, 1]:
        for dj in [-1, 0, 1]:
            if not (di == 0 and dj == 0):
                flood(img, i + di, j + dj, img.labels[ix], c_s)

```

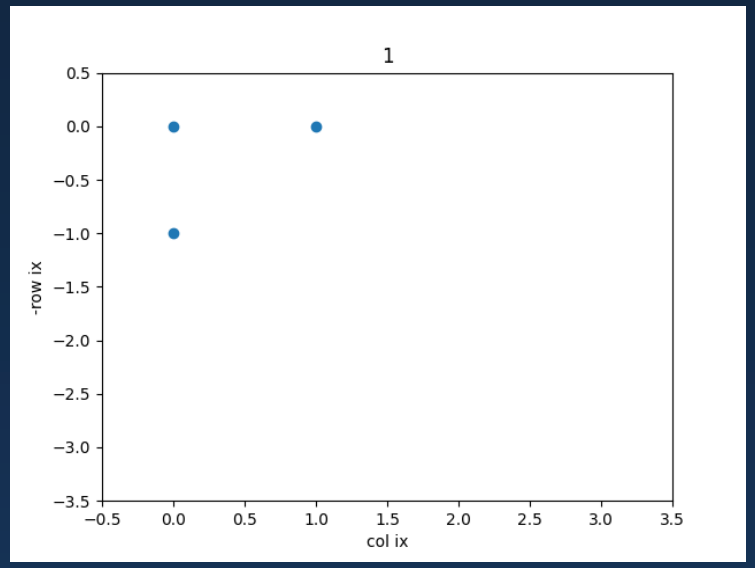


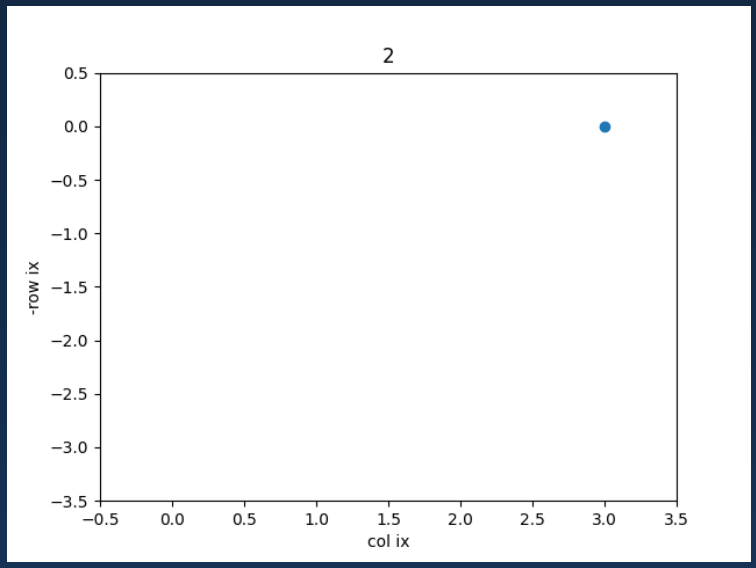
Step 2. Flood fill...

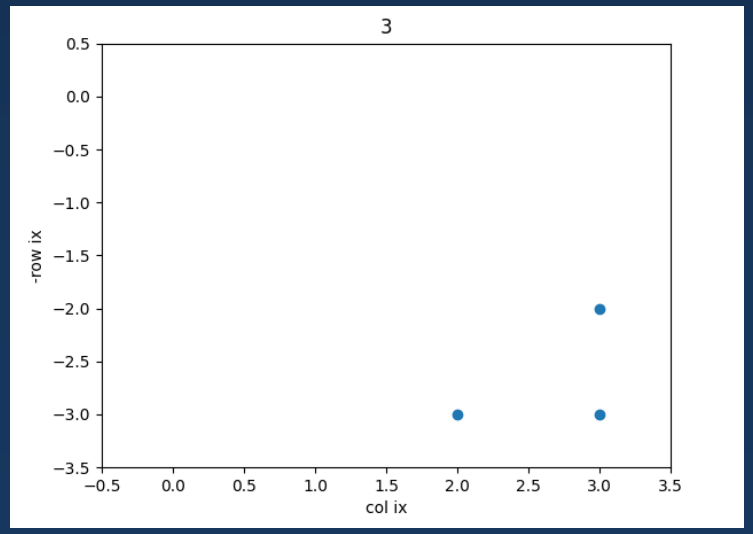


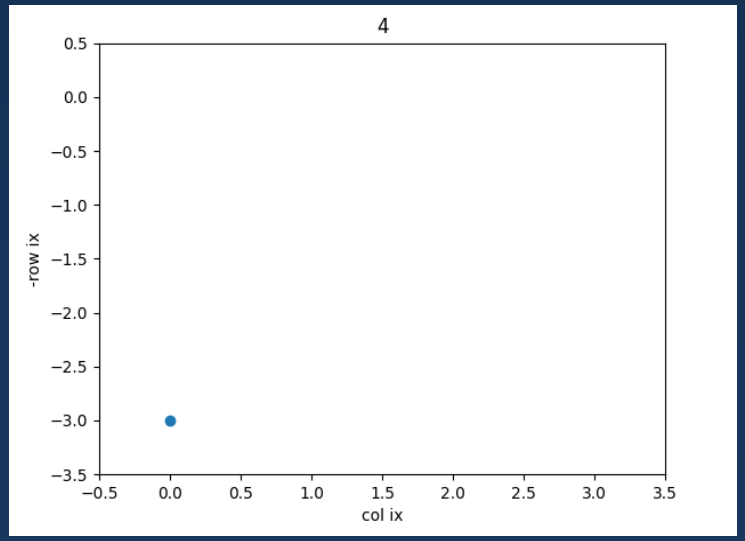












Why this order?

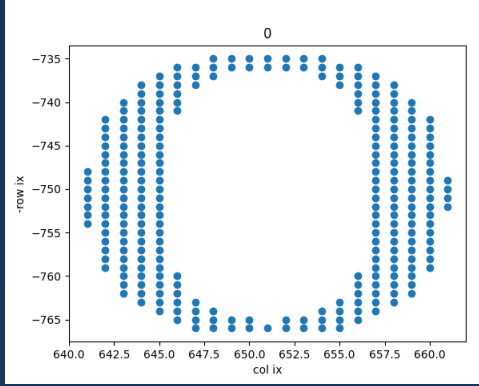
Flood fill segmentation

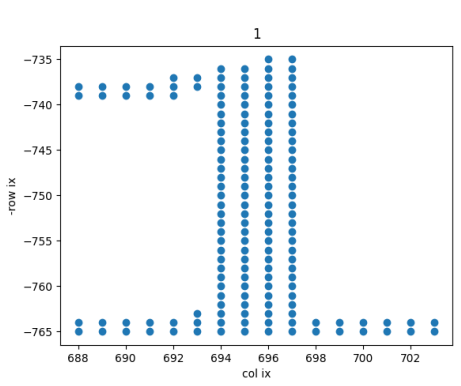
- "Smart scissors"
  - not "very" smart.. Auto scissors!
  - Grabs "Connected areas of uniform color"...
- Glyphs saved..
  - Truth/*.p
  - Truth/*.png (Check filenames match plots!)
  - Test/*.p
  - Test/*.png

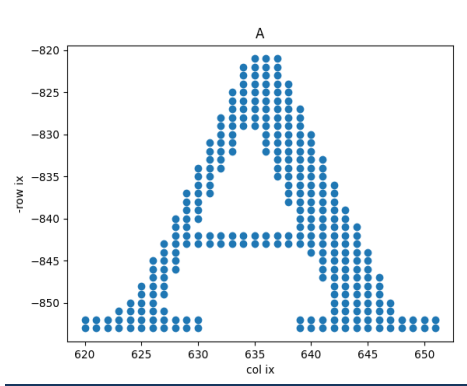
segment.py

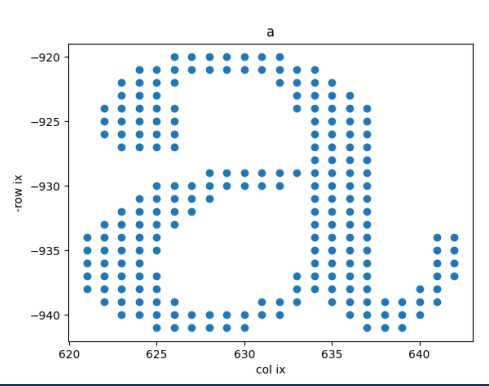
```
1 from image import image
2
3 # segment the truth data
4 truth = image('truth.bin')
5 truth.segment([745, 838, 932])
6
7 # segment the test data
8 test = image('test.bin')
9 test.segment()
```

why?

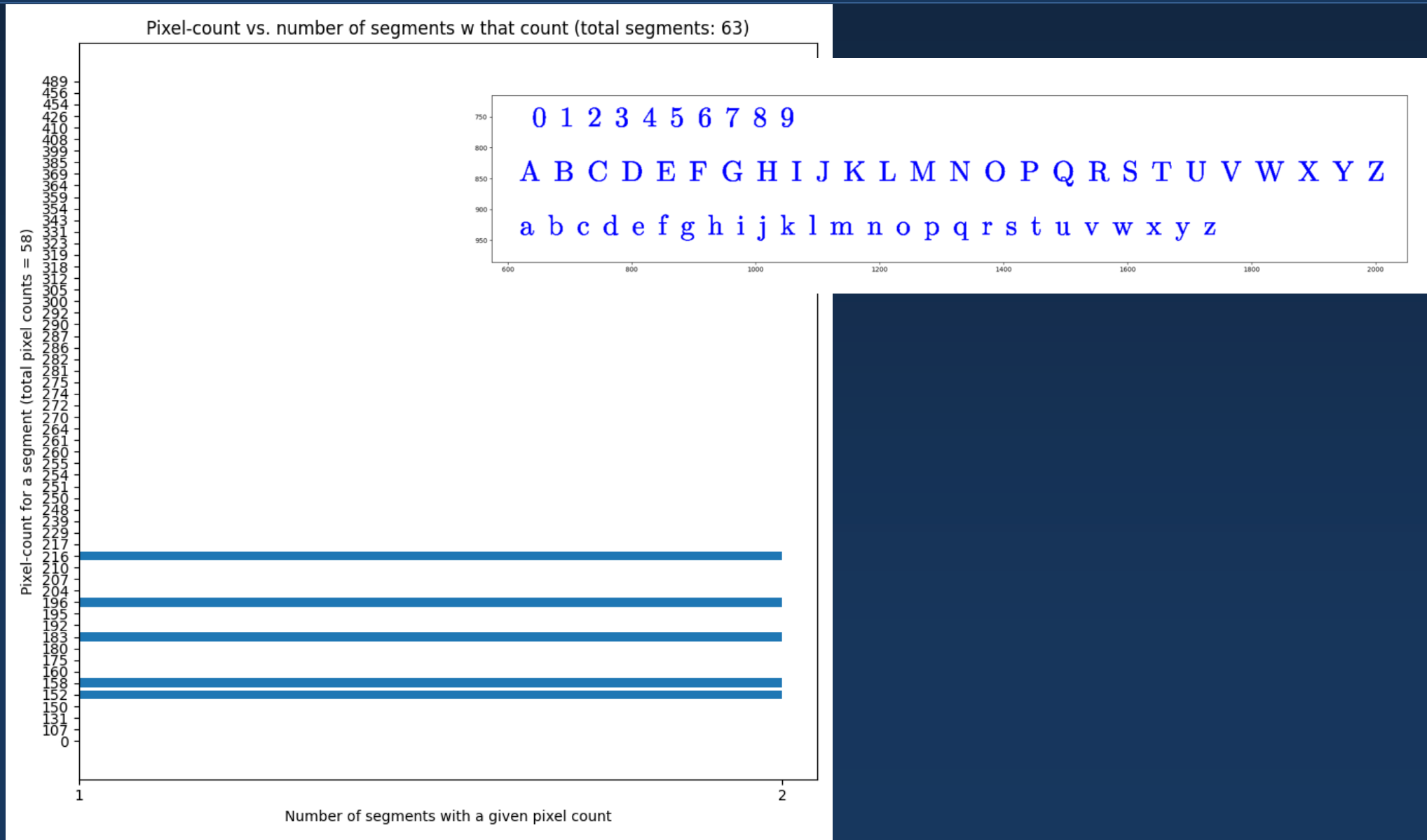






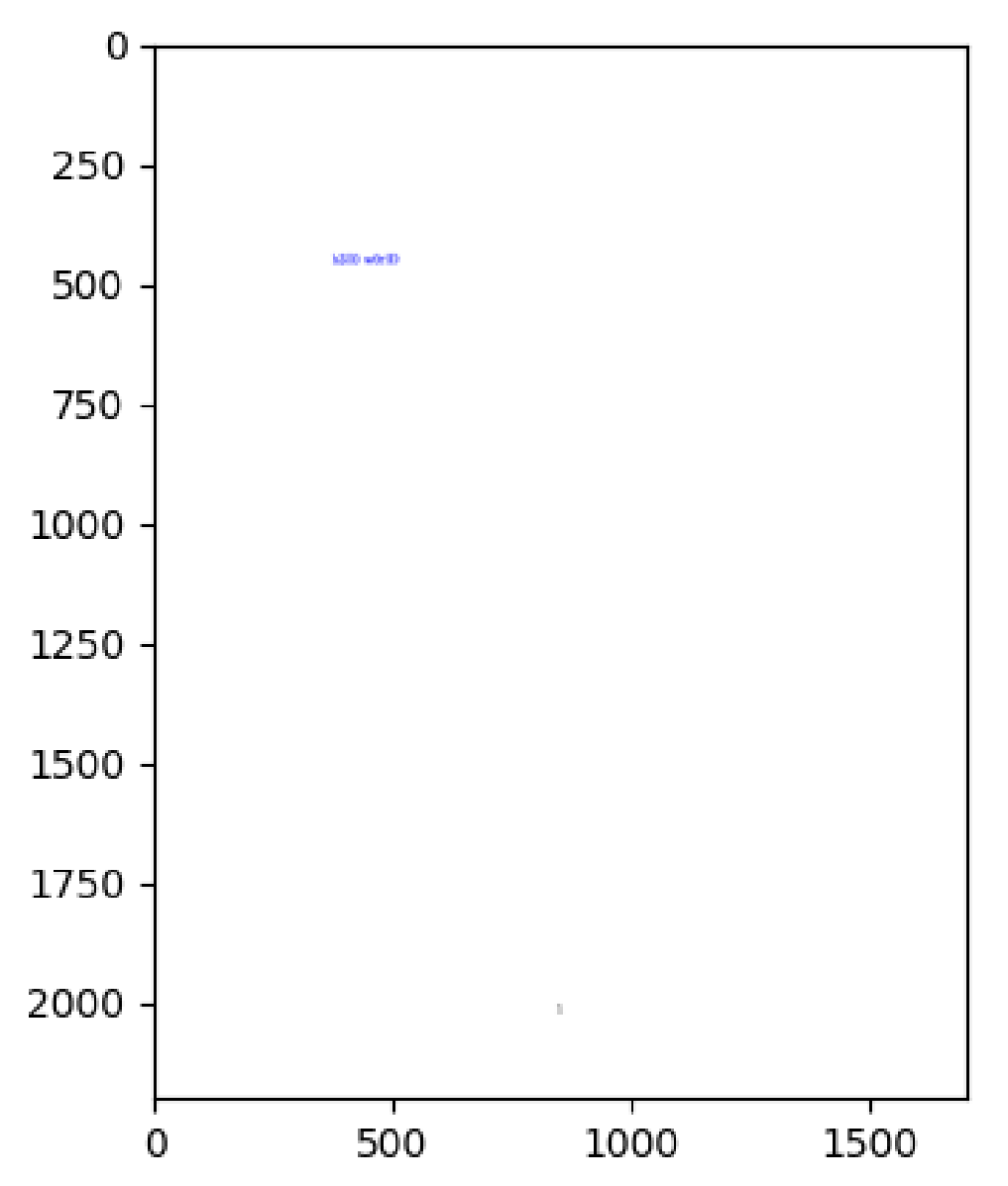


Truth

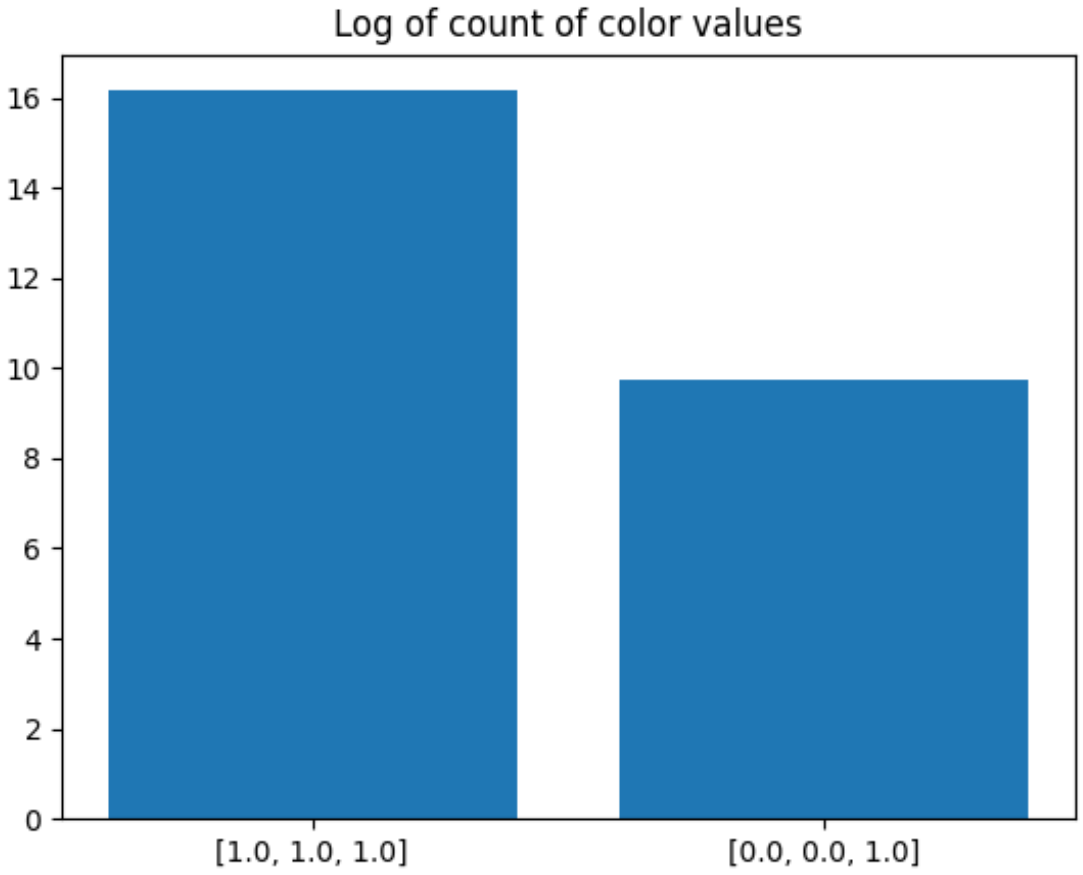


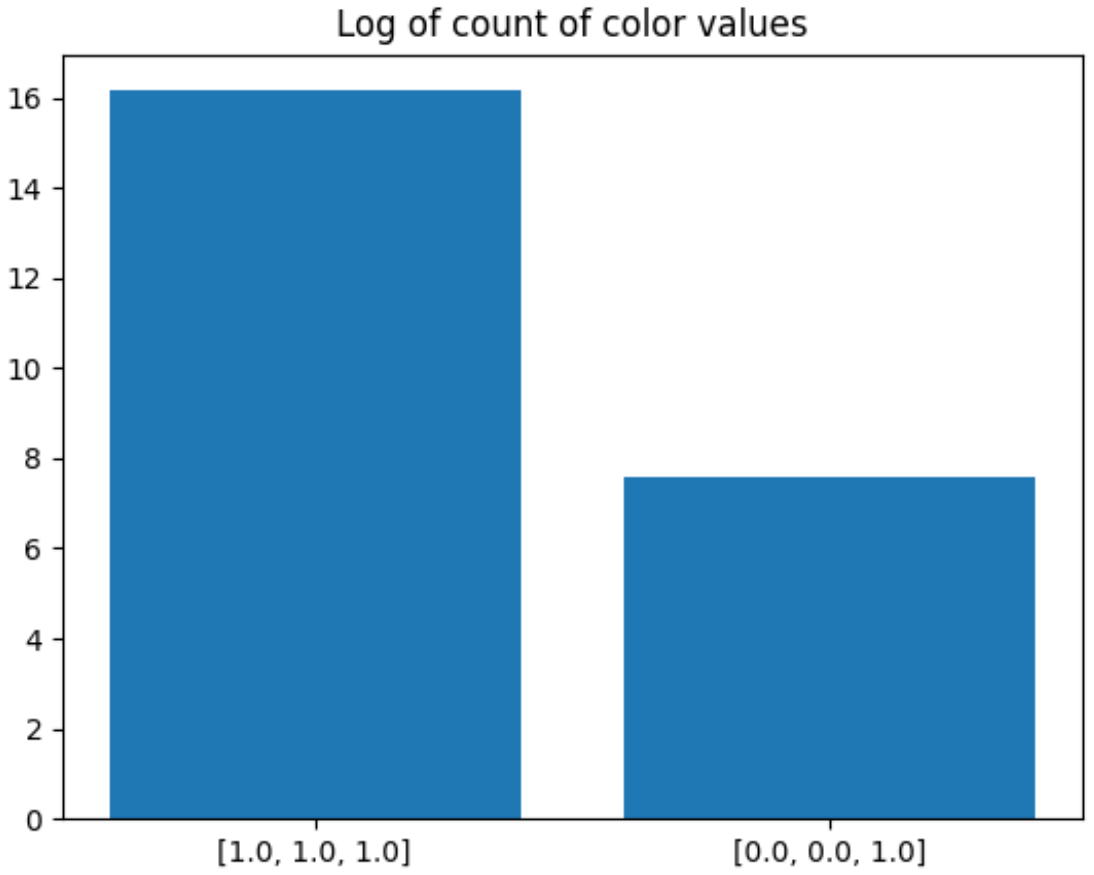
Most "segments" have Unique pixel-counts..
..except for a few that double up

Test: h3ll0 w0rld



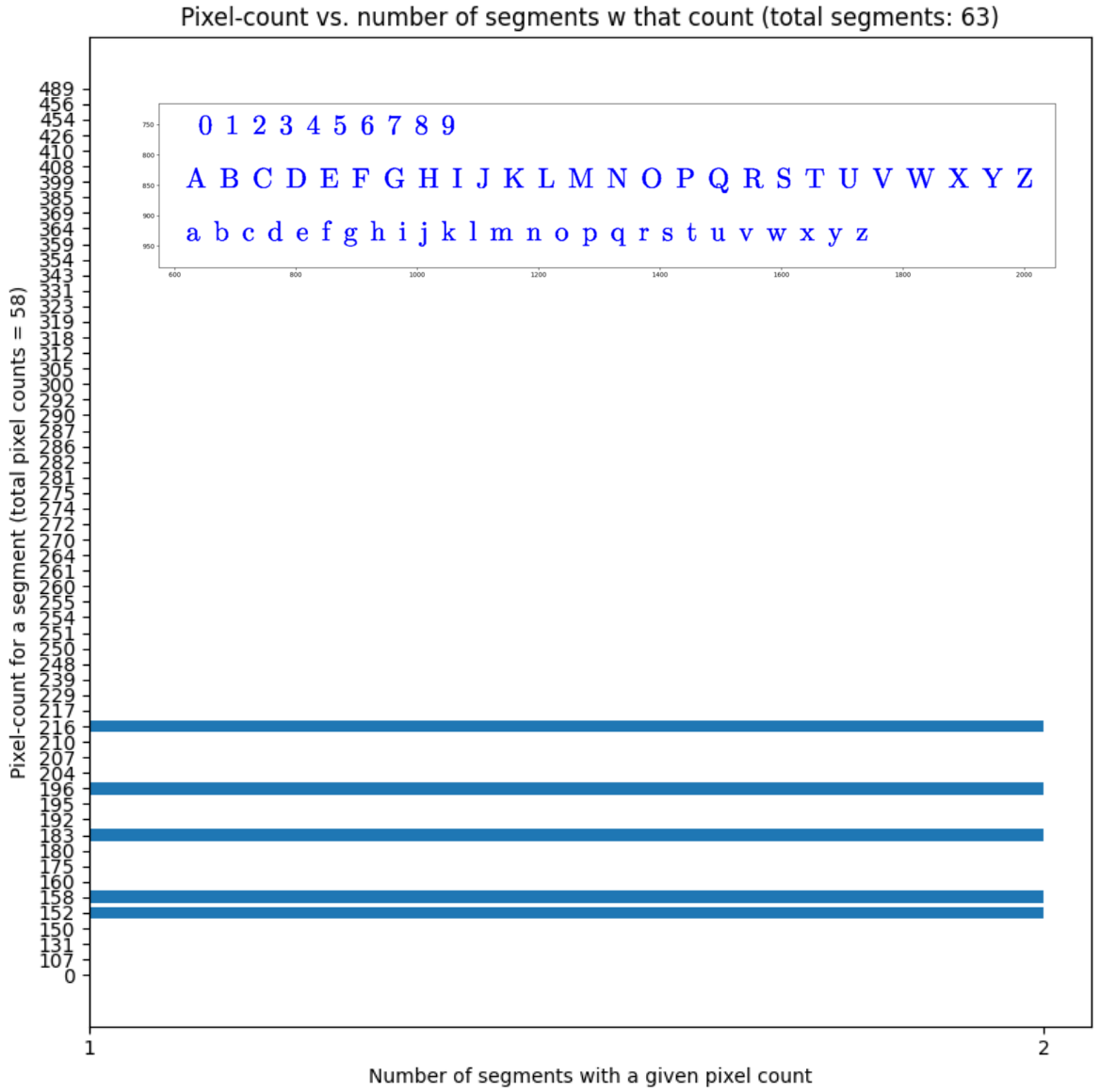
Truth vs Test

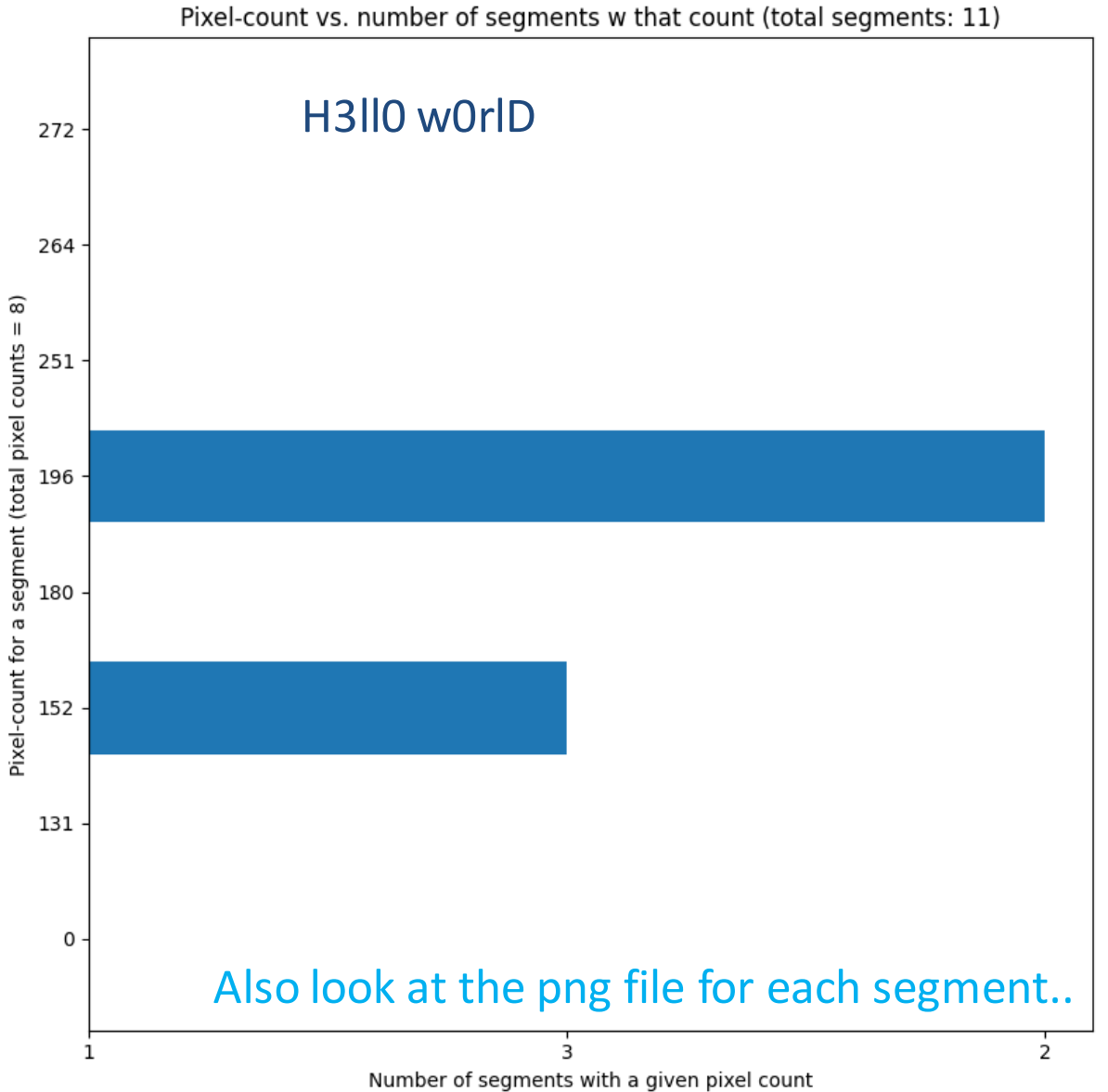




Truth vs Test

11 = hello world + bg





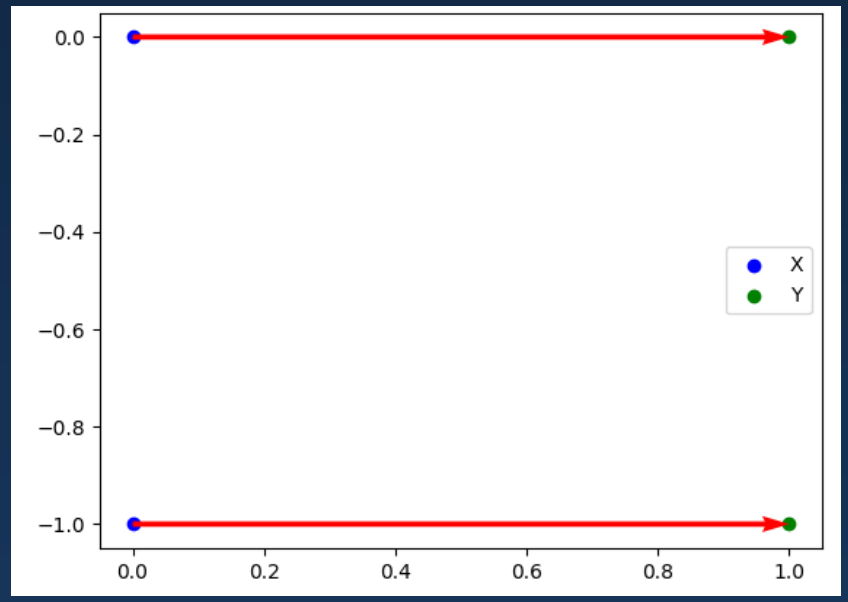
Prediction: Earth-mover Distance (EMD)

BC Wildfire Service

Traffic plan for moving a pile of stuff... initial and ending formation!

- We used "point" representation!
- Traffic plan = "where to move each piece"
  - Diagram for an "optimal" maneuver!

Example:
"move X to Y"
not a lot of combinations...here's one:




```
if __name__ == '__main__':
    a = [[0,1], [0,0]] # [x1, x2], [y1, y2]
    b = [[0,1], [1,1]] # [x1, x2], [y1, y2]
    d, arrows, subdist = dist(a, b)
    dist_plot(a, b, d, arrows, subdist, 0, 1)
```

Method:

1) Compare all possible moves
"distance matrix"

2) Choose shortest potential moves first, cross off the items that we add to our plan..

The other option (X shape) would not be optimal..
..it is optimal to move the points straight across!
Dmat: 1, 1, sqrt(2), sqrt(2)

https://en.wikipedia.org/wiki/Earth_mover%27s_distance

https://en.wikipedia.org/wiki/Wasserstein_metric

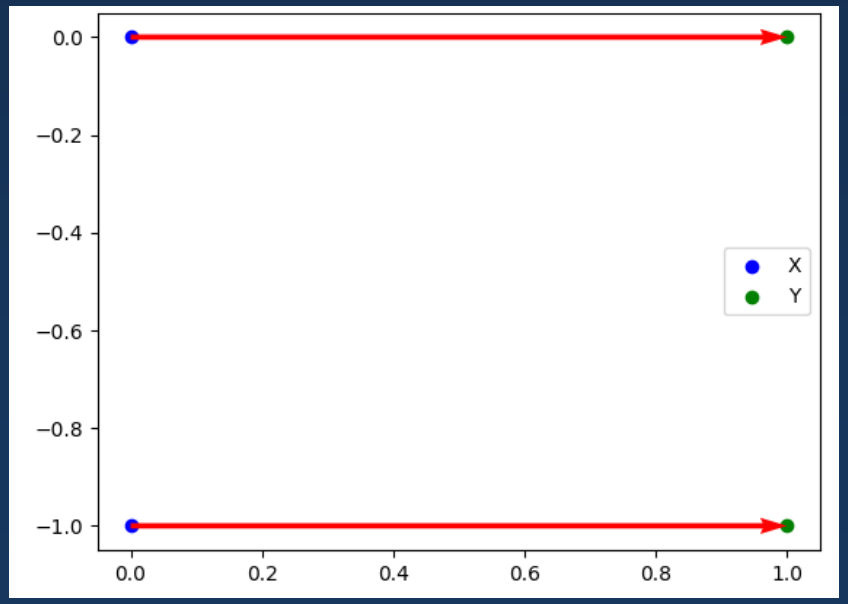
Prediction: Earth-mover Distance (EMD)

BC Wildfire Service

Method:

1) Compare all possible moves
   "distance matrix"
2) Choose shortest potential moves first,
   (for things that aren't moved already)

Add to plan and cross those items off as moved...

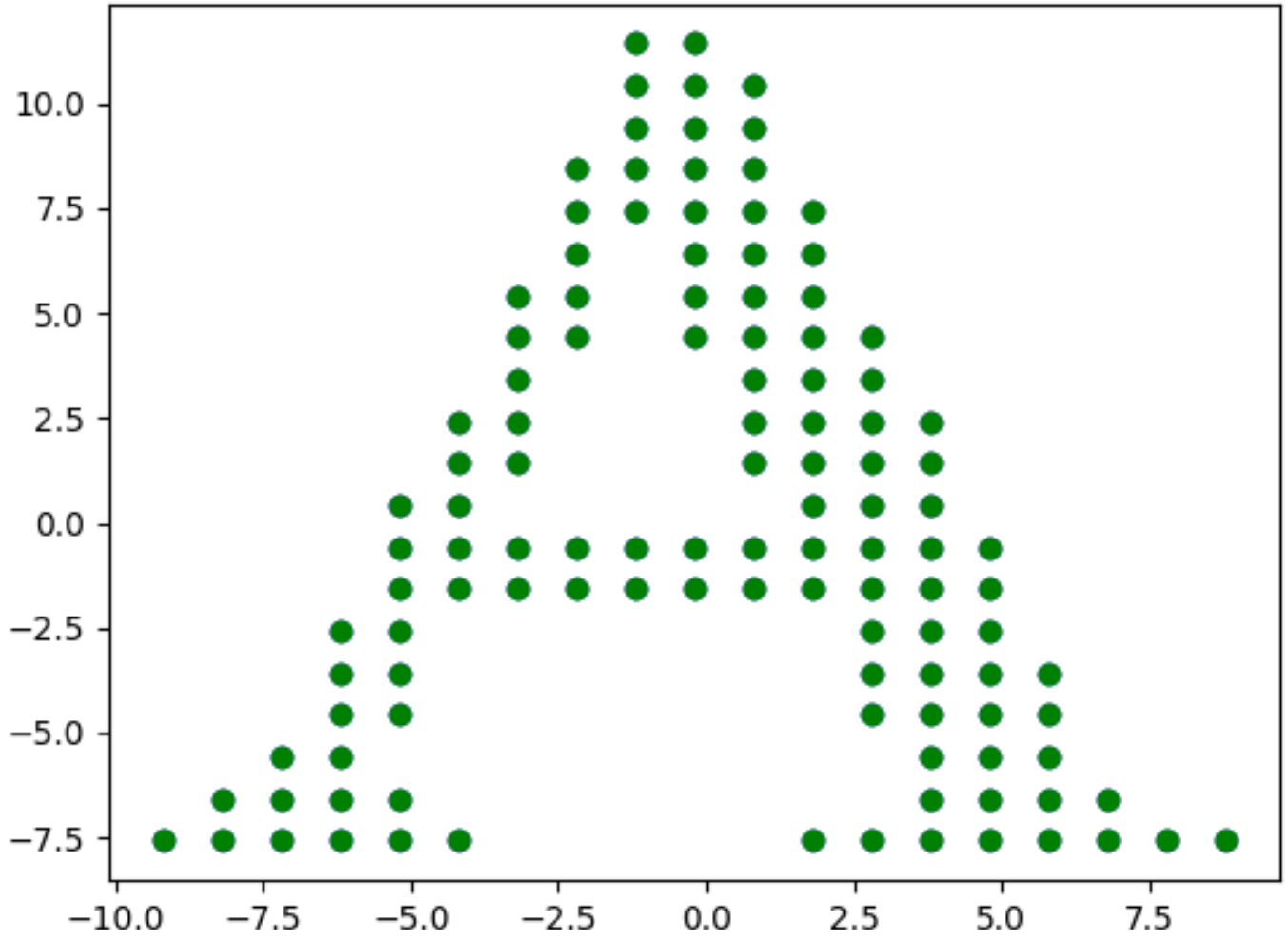


Result: Compare Shapes!

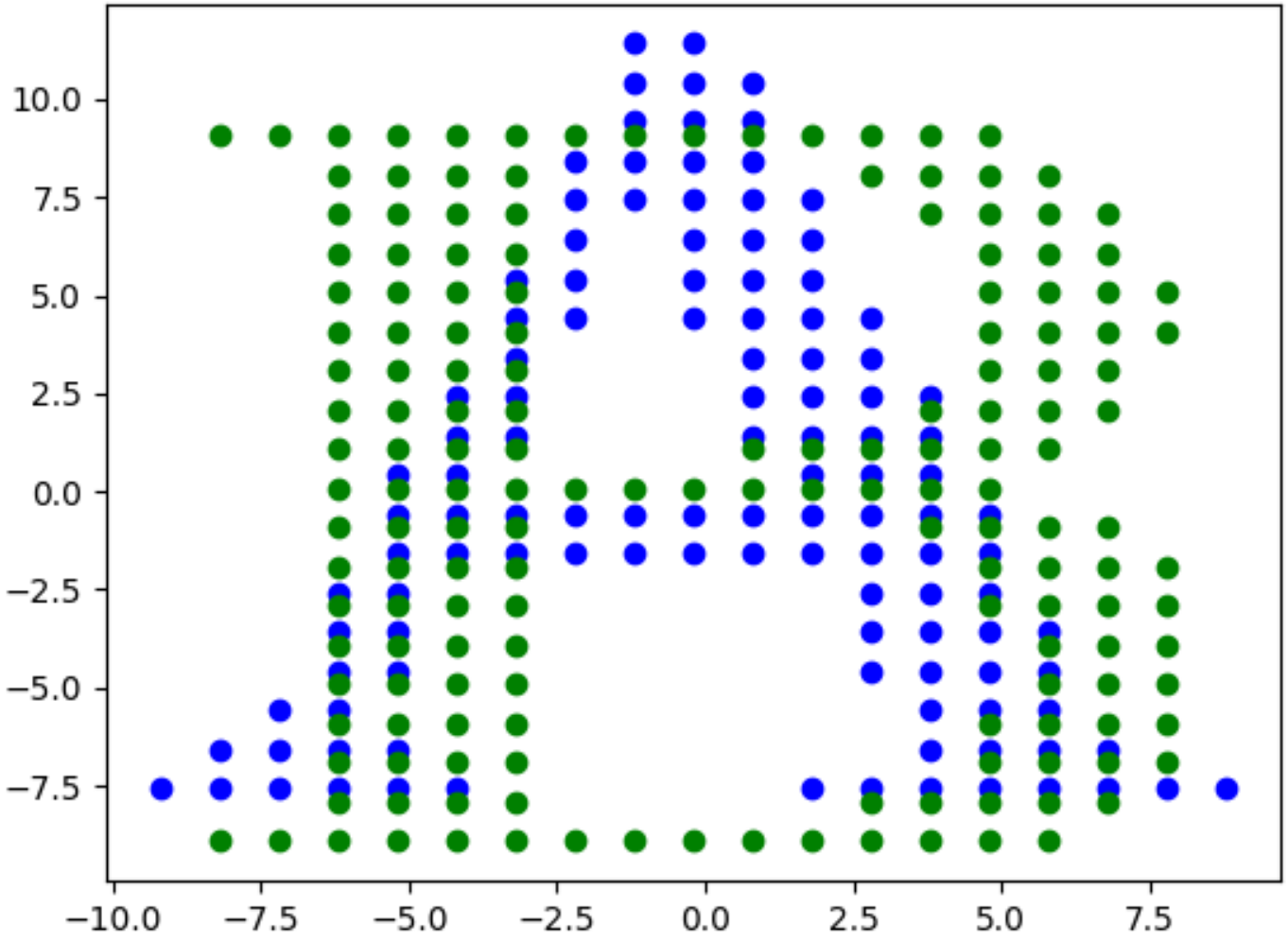
```
def dist(X, Y):
    rho, [x1, y1], [x2, y2] = 0, X, Y  # compare two point-sets
    i_n, j_n = len(x1), len(x2)
    subdist, arrows = [], []  # for visualization only

    # element-wise slots for inclusion of each element in traffic plan
    i_f, j_f = [False for i in range(i_n)], [False for i in range(j_n)]

    dm = [[abs(x1[i] - x2[j]) + abs(y1[i] - y2[j]), i, j]
          for i in range(i_n) for j in range(j_n)]  # distance matrix
    dm.sort()  # sorted distance matrix

    for k in range(len(dm)):  # for each dmat element
        d, i, j = dm[k]  # dist btwn points, indices of points compared

        if not i_f[i] and not j_f[j]:  # if slots open for both elements:

            # add term to distance and close the slots
            i_f[i], i_n, j_f[j], j_n, rho = [True,
                                             i_n - 1,
                                             True,
                                             j_n - 1,
                                             rho + d]

            if d > 0:  # record stuff for visualization
                arrows.append([[x1[i], y1[i]], [x2[j], y2[j]]])
                subdist.append(d)

        if i_n * j_n == 0:
            break  # ran out of slots for X or Y: stop comparing..

    rho /= min(float(len(x1)), float(len(x2)))  # divide by the number of slots
    return rho, arrows, subdist
```

Prediction: Earth-mover Distance

BRITISH COLUMBIA | BC Wildfire Service

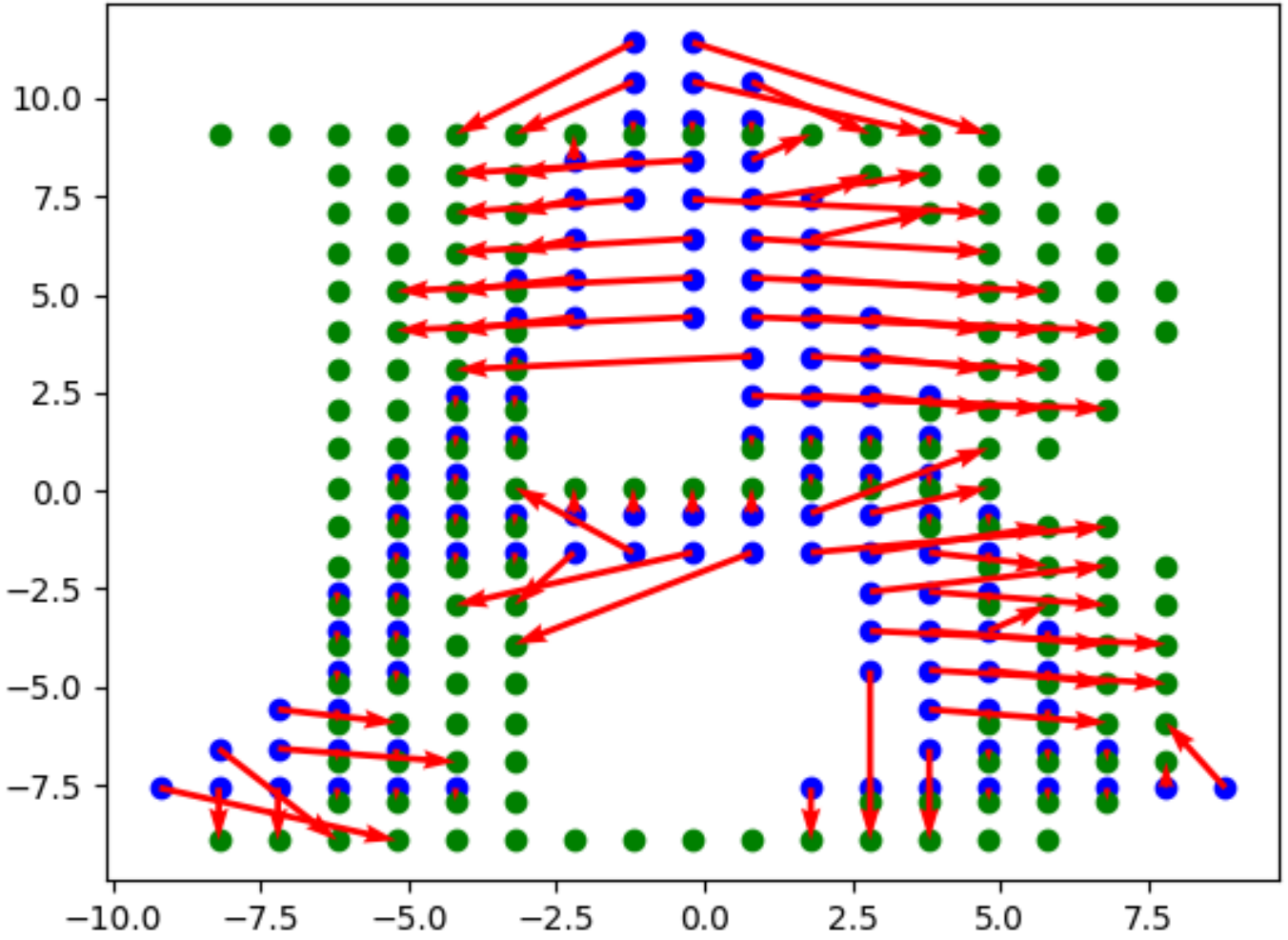


Prediction: Earth-mover Distance



Compare A with B!

Prediction: Earth-mover Distance

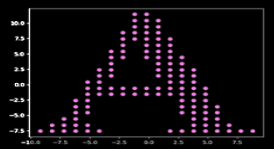
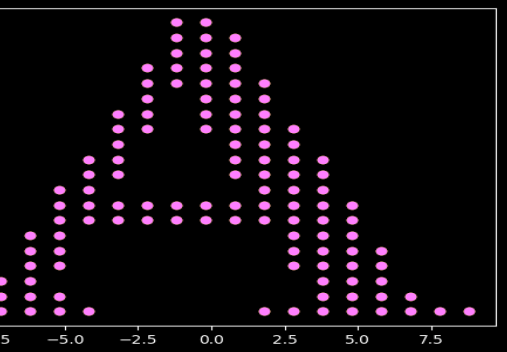
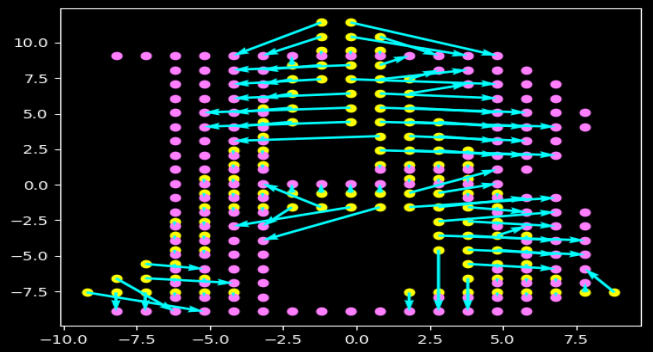
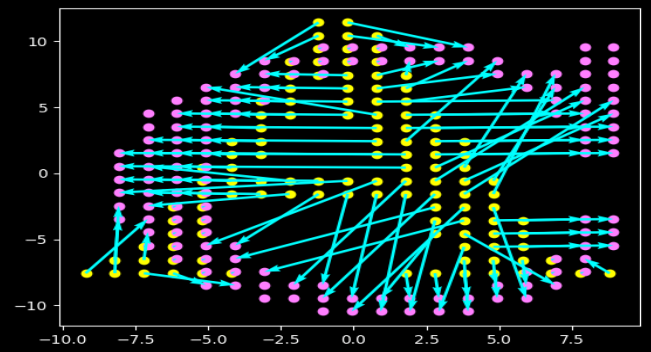
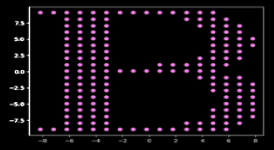
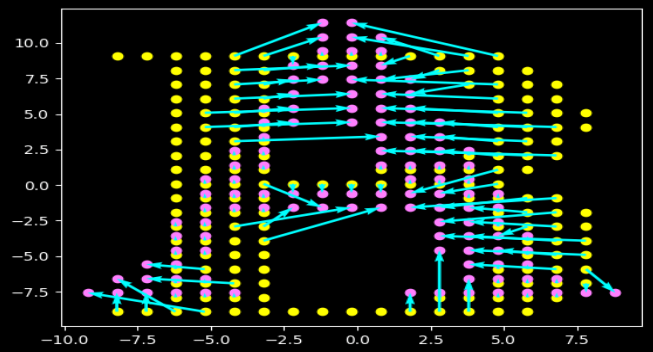
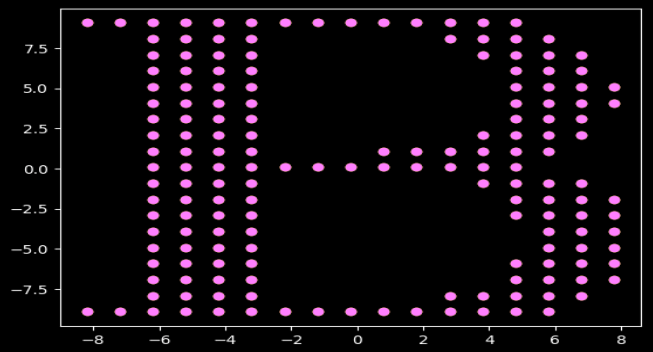
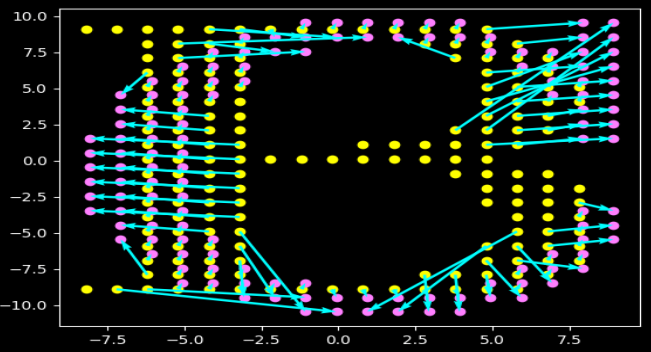
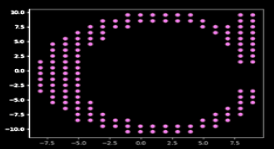


Red:
"Transport Plan":

Diagram that says what to move where

Idea: similar figures take less energy to transform...
...why? Some tolerance for the figures not being exactly the same

Look at relationships between letters..
...matrix!

How to predict / classify with EMD?

Same as you would with other distance...

Find closest (least distance) "truth" data-point for each test data-point...
..label the test point accordingly!

"If our test item looks more like an A than any other letters, label it A"

"Pseudocode":
1) For each test data (letter):
2) Compare with each truth item (alphabet etc),
3) find the closest match
(according to EMD "efficient morphing").

4) Label accordingly... "predict"

Run in parallel!!!!! Because...

predict.py

```
def predict_i(pi):  # for pi in range(len(test_points)):
    print("predict_i", pi)
    # print(test_files[pi])
    p = normalize(test_points[pi])  # didn't normalize before to get absolute centroid

    min_d, min_i = sys.float_info.max, None  # closest truth value

    for i in range(len(truth_points)):
        t = normalize(truth_points[i])
        d, arrows, subdist = dist(p, t)
        print("rho", d, "i", i)

        if pi == 0:
            dist_plot(p, t, d, arrows, subdist, pi, i)

        if d < min_d:  # found a better match
            min_d, min_i = d, i

        if min_d == 0.:  # perfect match, stop looking..
            break

    print("min_i", min_i)
    prediction = truth_labels[min_i]
    result = [test_centroids[pi], prediction]
    print(result)
    return(result)
    print("point", pi, "of", len(test_points))



print("truth_points", truth_points)

use_parfor = True
if use_parfor:
    predictions = parfor(predict_i, range(len(test_points)))
else:
    predictions = [predict_i(pi) for pi in range(len(test_points))]

print("prediction", predictions)

plt.figure()
for p in predictions:
    plt.plot(p[0][1], -p[0][0])
    plt.text(p[0][1], -p[0][0], p[1])
plt.show()
plt.savefig("prediction.png")
```

Prediction for h3ll0 w0rlD..

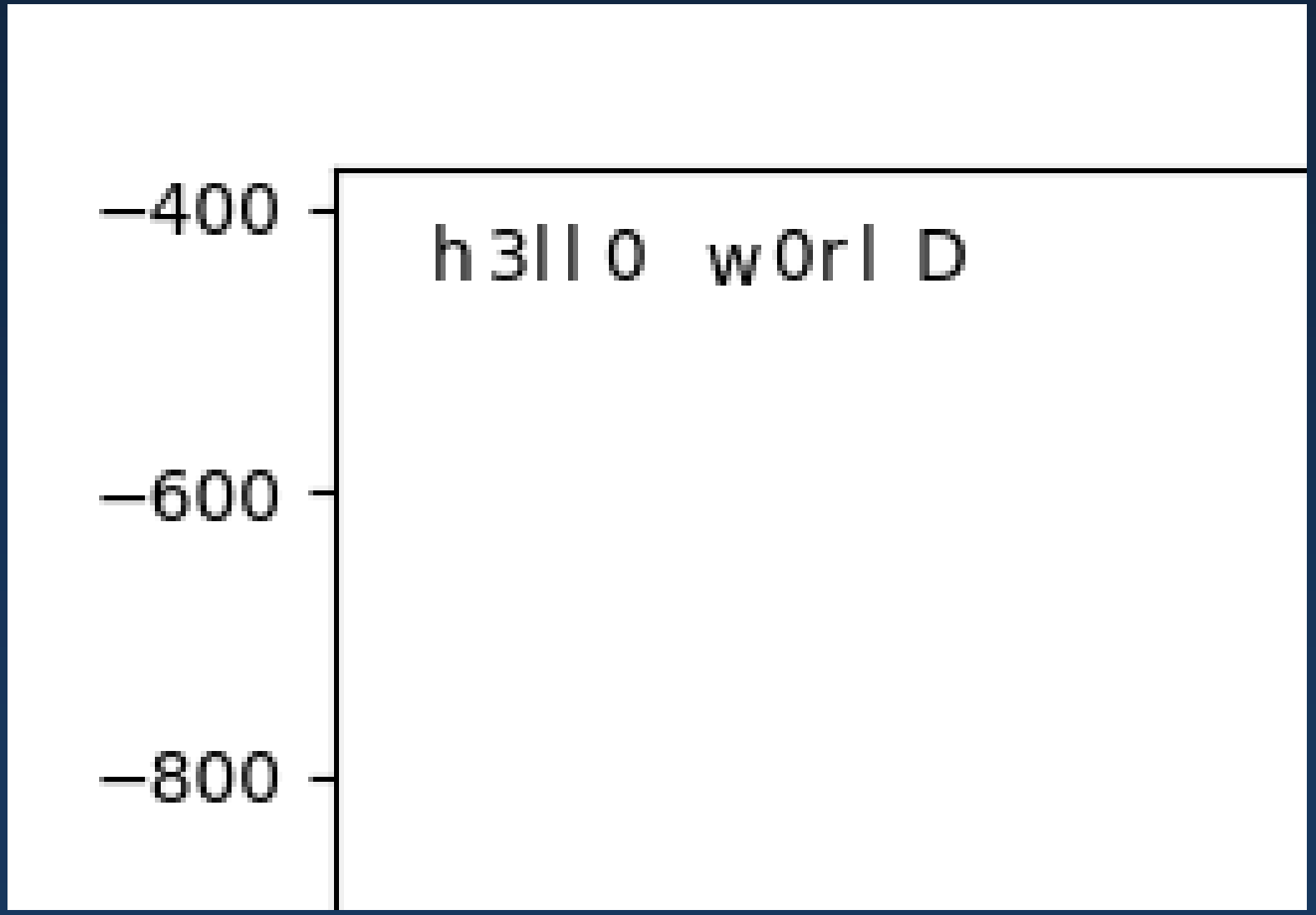
truth ['h', '3', 'l', 'l', '0', 'w', '0', 'r', 'l', 'D']

prediction ['3', 'D', 'l', '0', 'h', 'r', '0', 'l', 'l', 'w']

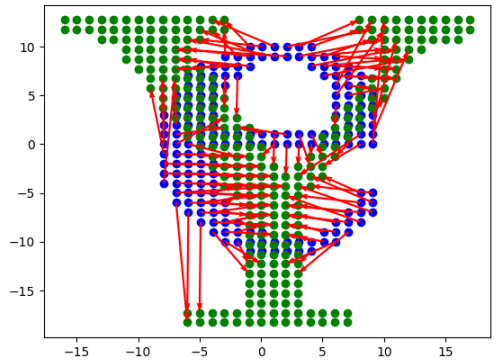
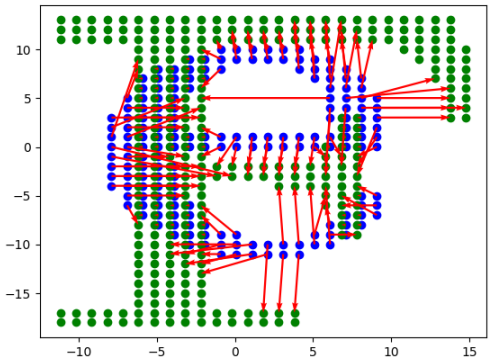
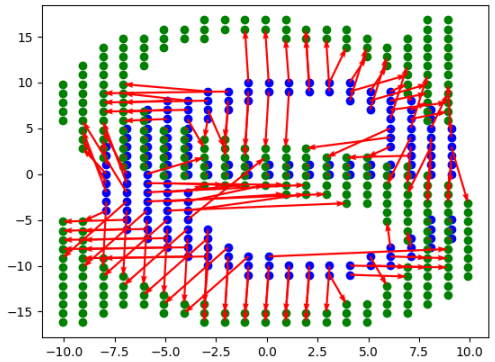
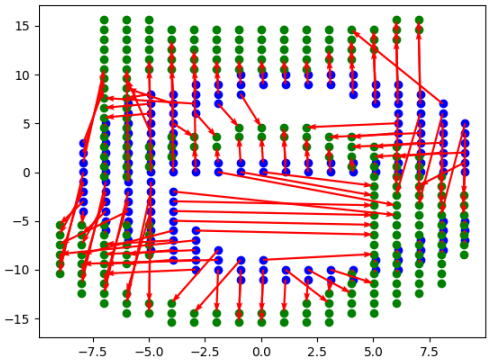
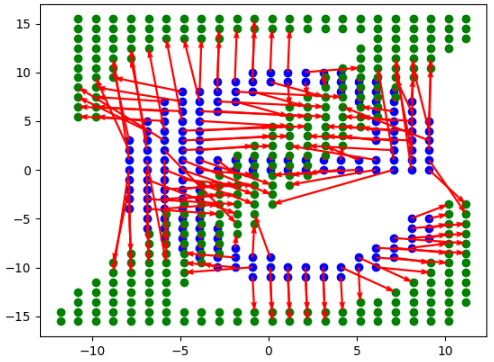
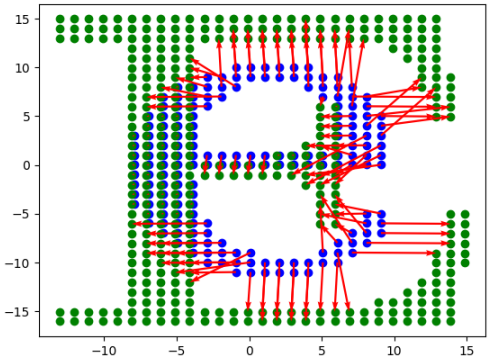
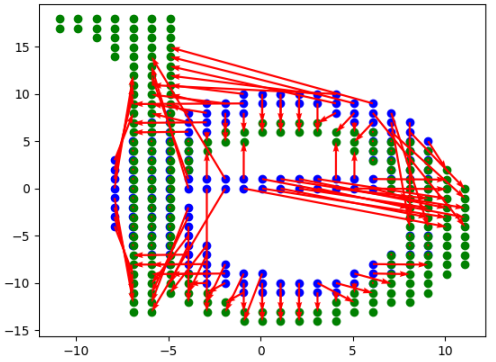
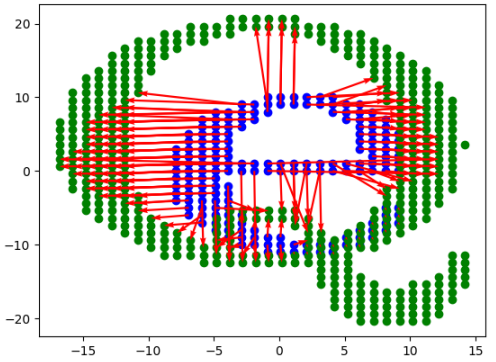
The letters aren't the same height...
….scanning right-left top-bottom doesn’t pull them in order.

- For truth data set scan lines (in the segmentation code...)
- For predictions, plot the predicted value at the centroid location

Predictions: plot at centroid locations..

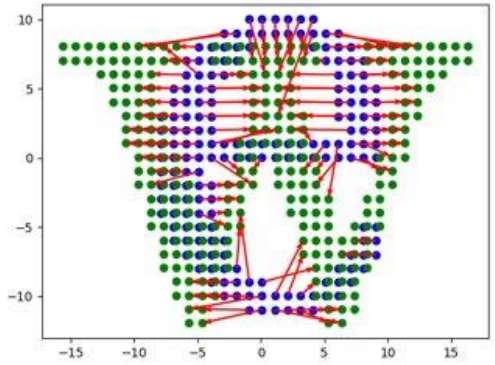
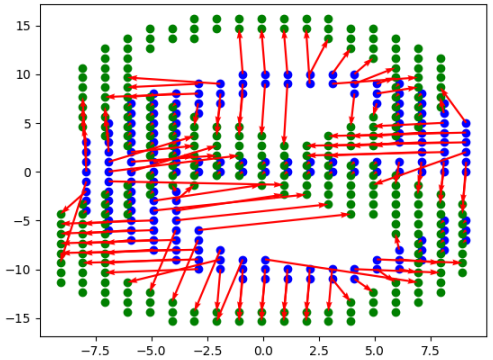
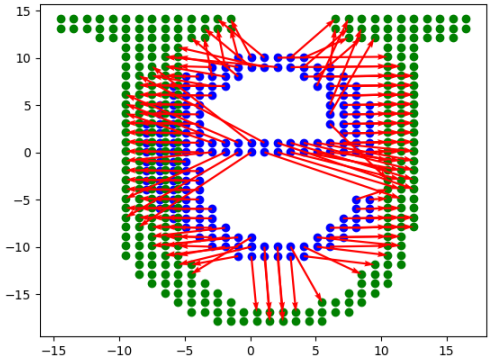
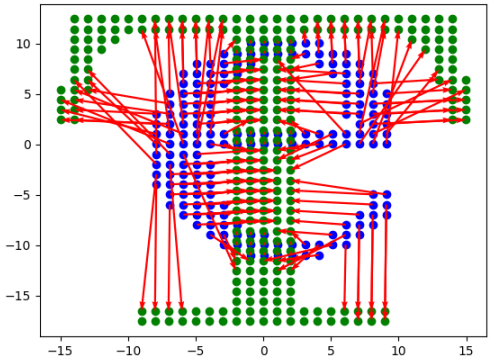
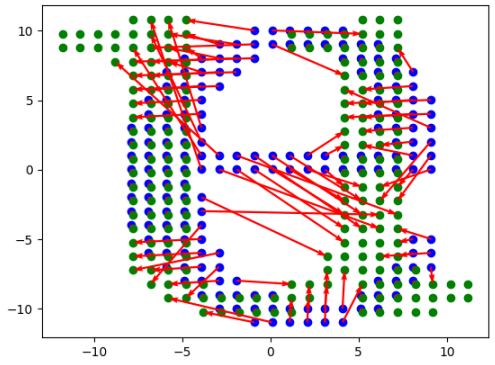
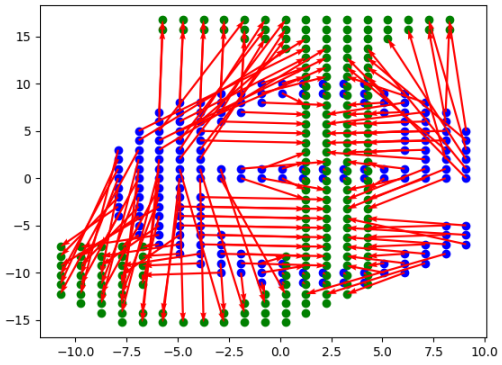
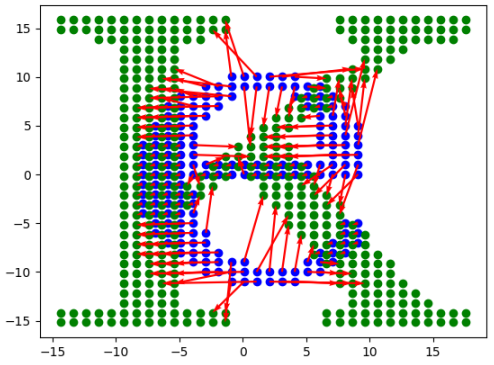
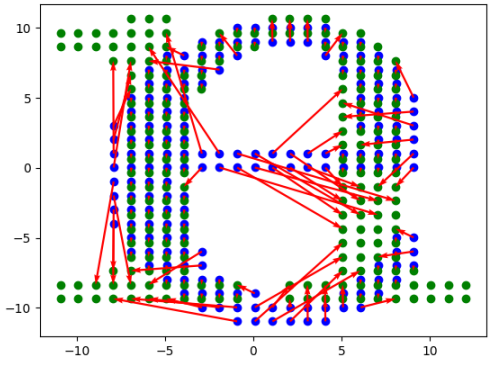


Prediction: example... Match onto "e"..

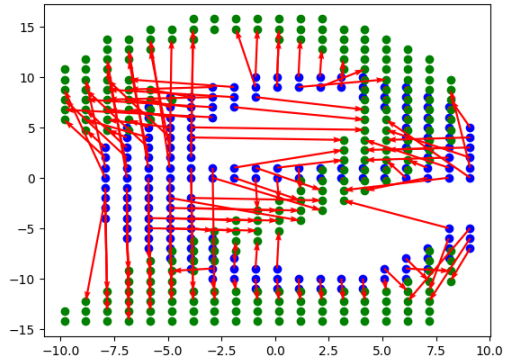
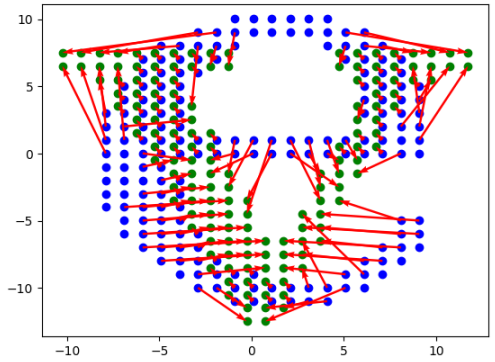
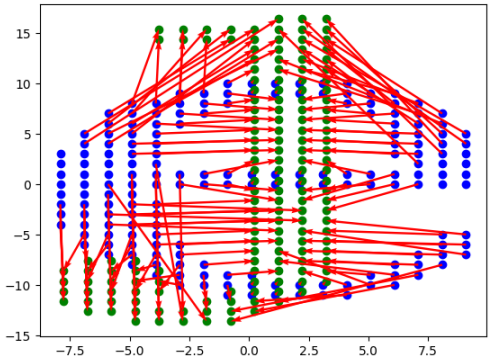
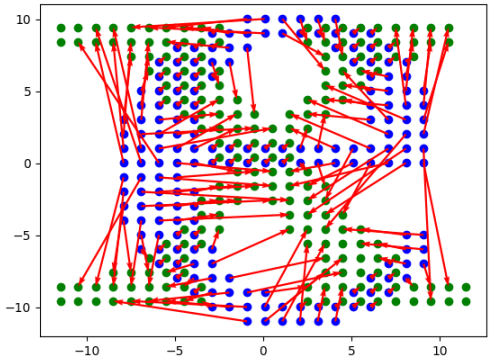
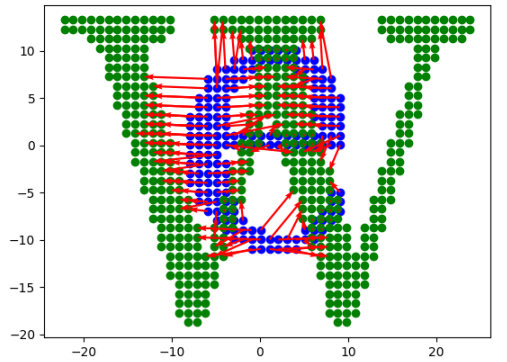
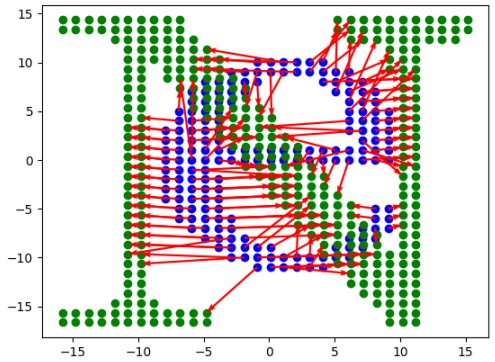
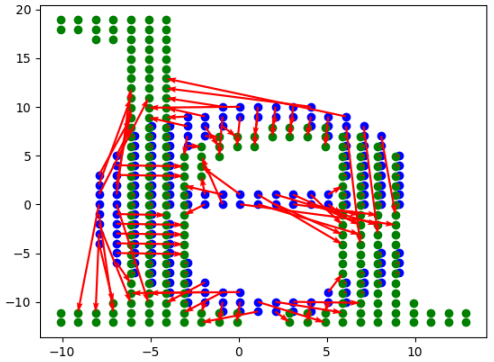
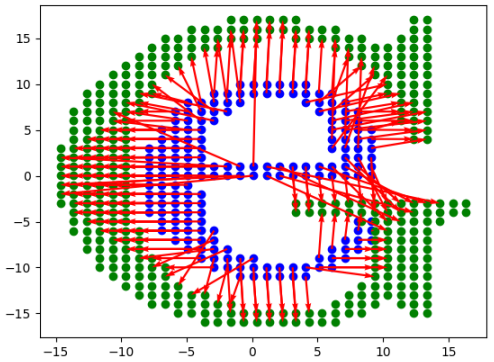
Prediction: example... Match onto "e"..

BRITISH COLUMBIA | BC Wildfire Service

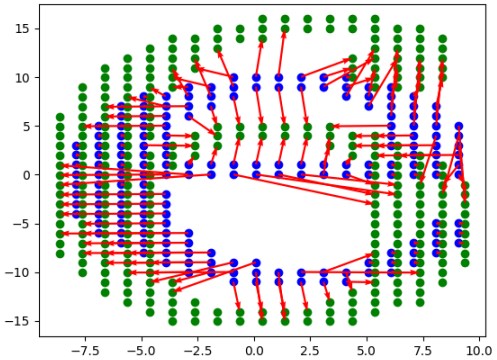









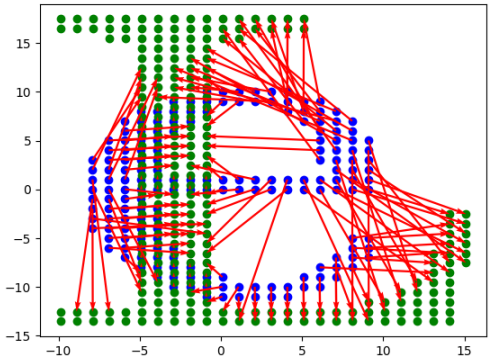
Prediction: example... Match onto "e"..

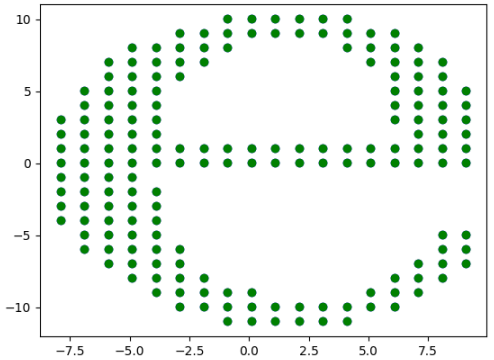
BRITISH COLUMBIA | BC Wildfire Service

Prediction: example... Match onto "e"..







Perfect overlap..

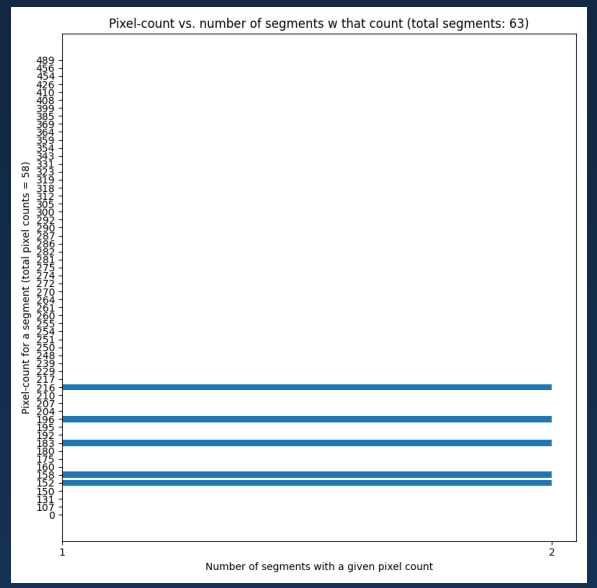
..Got a good match, can stop now!

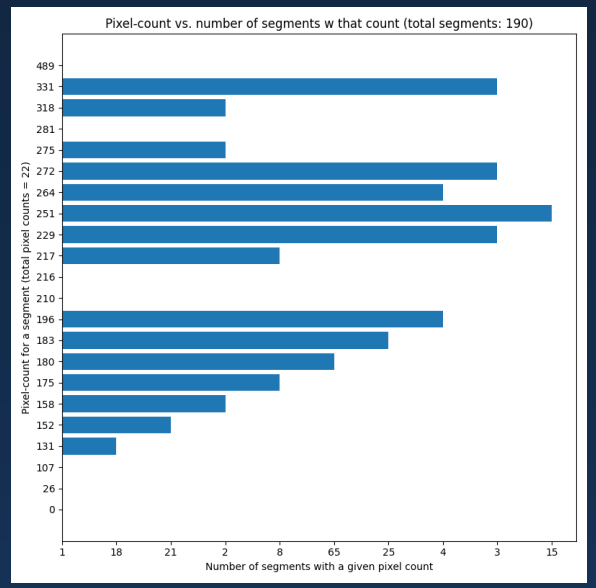
Good news:

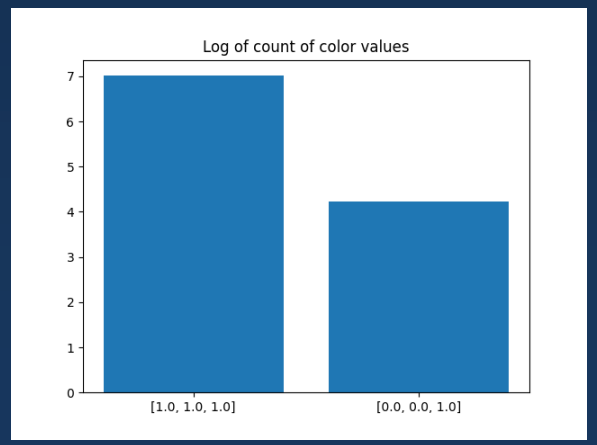
This will still work if the "e" we were matching was a slightly different shape from the "e" in the "truth" set....

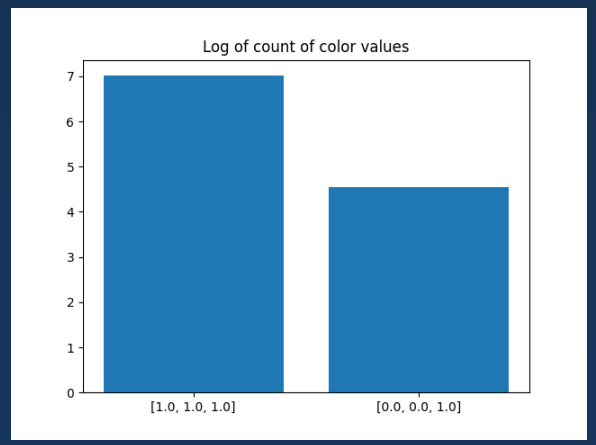
Scaling up..

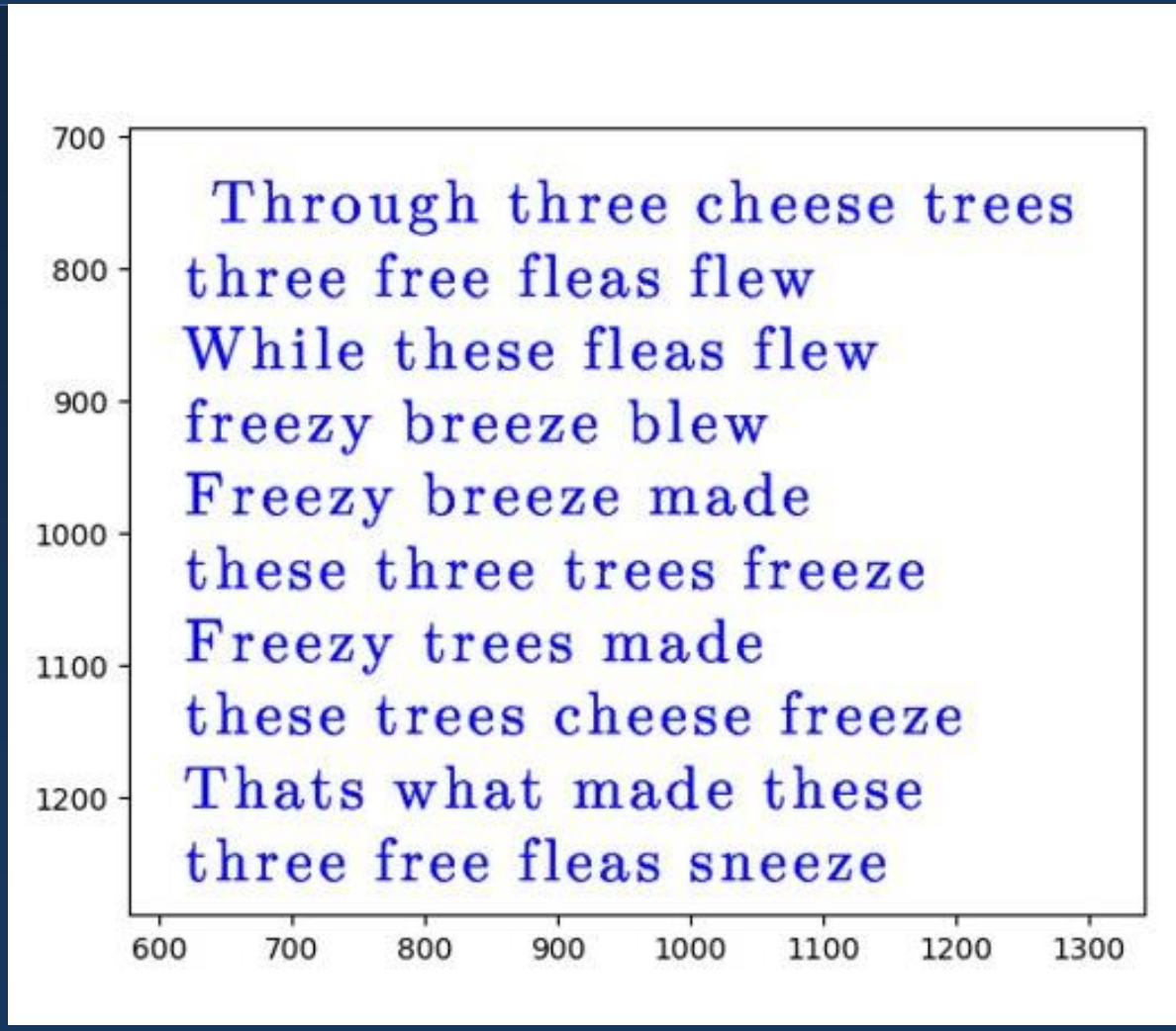
BC Wildfire Service



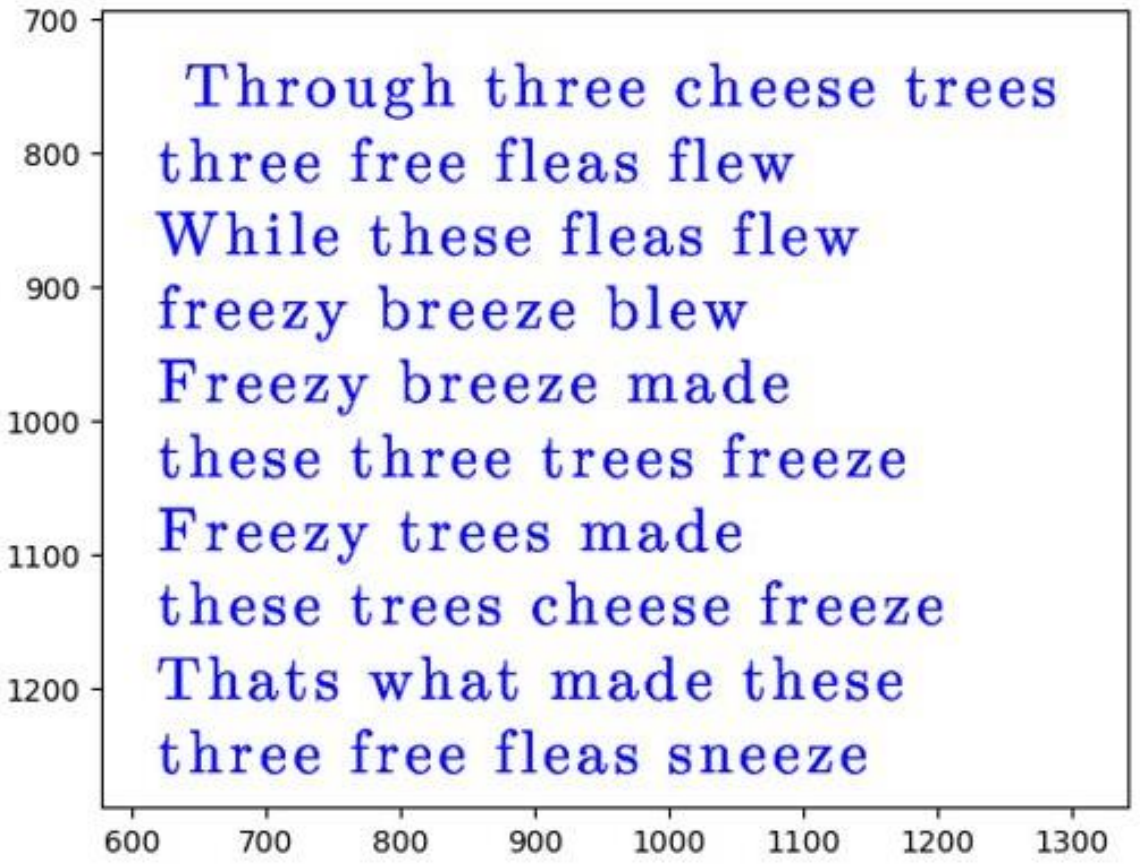




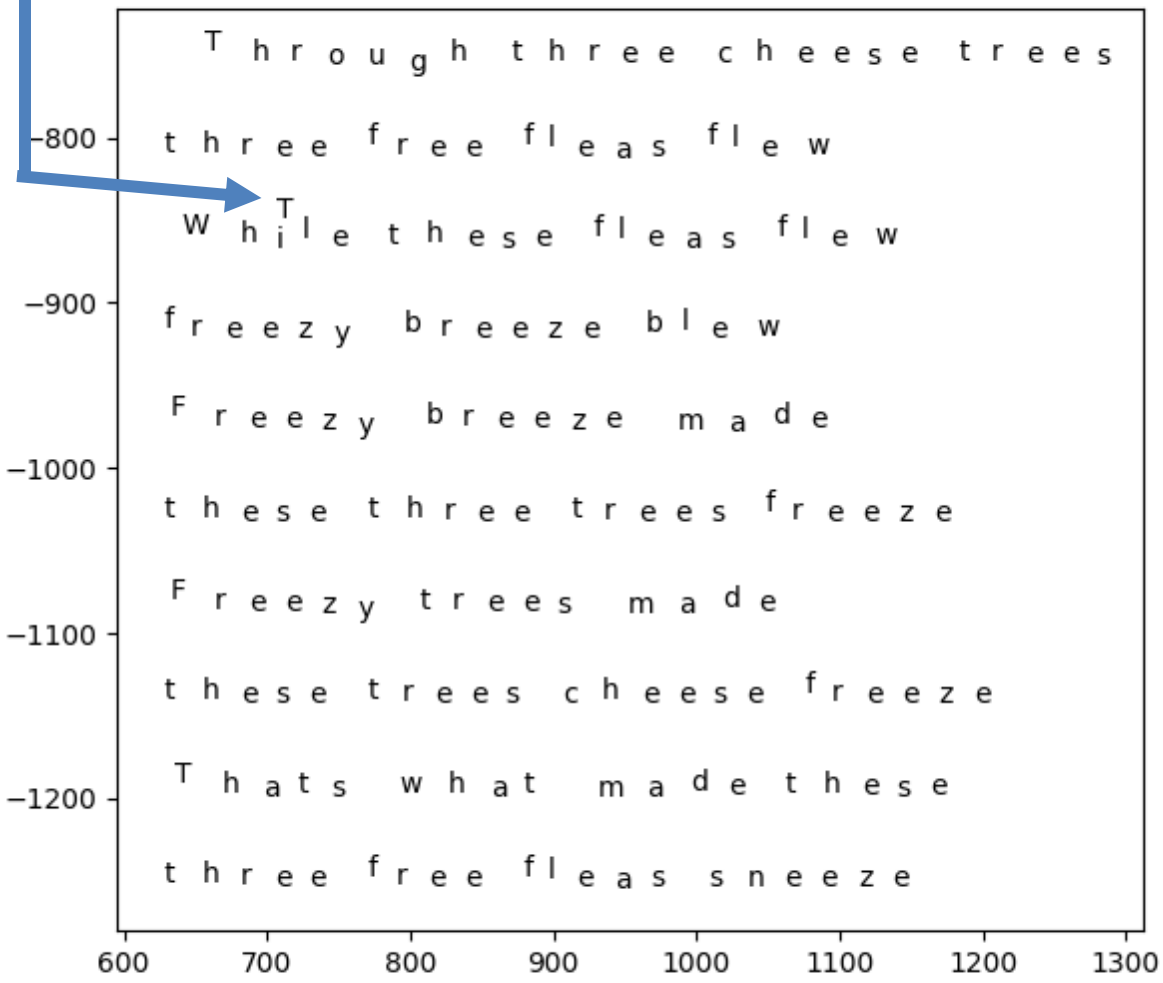




Prediction



What's that?



Need a segment size threshold..

Earth Mover + Predictive Models.. Applications

BRITISH COLUMBIA | BC Wildfire Service

- Compare patterns / shapes!
  - By morphing them
  - Makes sense for many kinds of data..
- Image classification.. Sensor fusion for environment monitoring
  - Land
  - Atmosphere
  - LiDAR, Multispectral, Radar, Hyperspectral etc...(or combos)..
- Resource allocation / distribution..
  - Cost efficiency for maneuvers (planes, helicopters etc.)...
    - route finding for groups, rather than just individuals!

PyTesseract example!!!

BC Wildfire Service

Very short script for OCR using OpenCV and Tesseract (thanks Ken Fullerton for the share!)

# text recognition	
	import cv2
	import pytesseract
	# read image
	im = cv2.imread('./test3.jpg')
	# configurations
	config = ('-l eng --oem 1 --psm 3')
	# pytessercat
	text = pytesseract.image_to_string(im, config=config)
	# print text
	text = text.split('\n')
	text

https://www.analyticsvidhya.com/blog/2020/05/build-your-own-ocr-google-tesseract-opencv/

Questions?

Ashlin.Richardson@gov.bc.ca