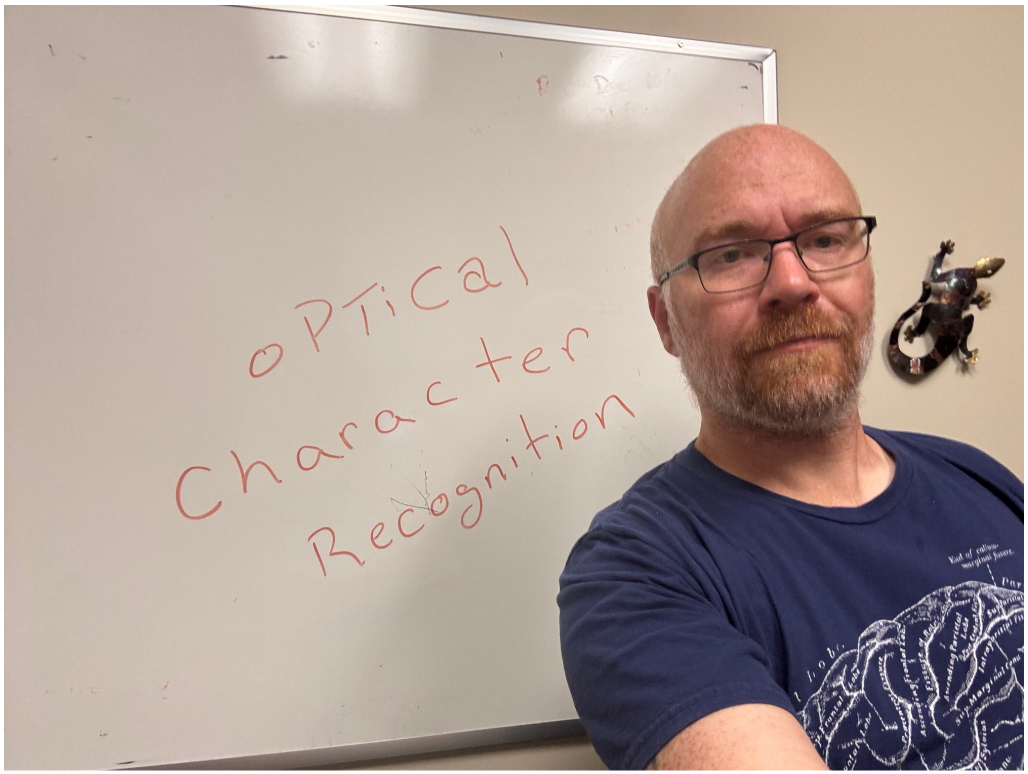



# Lecture 18

## 0PfiCa1 Chaca c ter Recojation

ECE 1390/2390

# Project Updates (10/30 & 11/4)

10/30

- BokehSwap
- DC not DC
- AutonomousDriving
- RepVision
- FaceEmojiSwap

11/4

- Team Sebastian, Timothy, Jake, & Tyler
- Shopkeepr
- The Riddlers
- Touchfree

## What do I expect?

- Progress update
- 10min informal presentation
- Can use your own laptop or give me slides

- Show off what you have done so far
- Have you tried things that didn’t work?
- What class lessons have you incorporated into the project? Any insights?
- Have your objectives changed?
- Has there been any unanticipated issues?

# Tesseract OCR Engine

[2] R.W. Smith, *The Extraction and Recognition of Text from Multimedia Document Images*, PhD Thesis, University of Bristol, November 1987.

- Developed by HP 1984-1994
- Purchased by Google 2006-2018
- Now open-source Apache-2.0 license

**Executable**

- https://github.com/tesseract-ocr

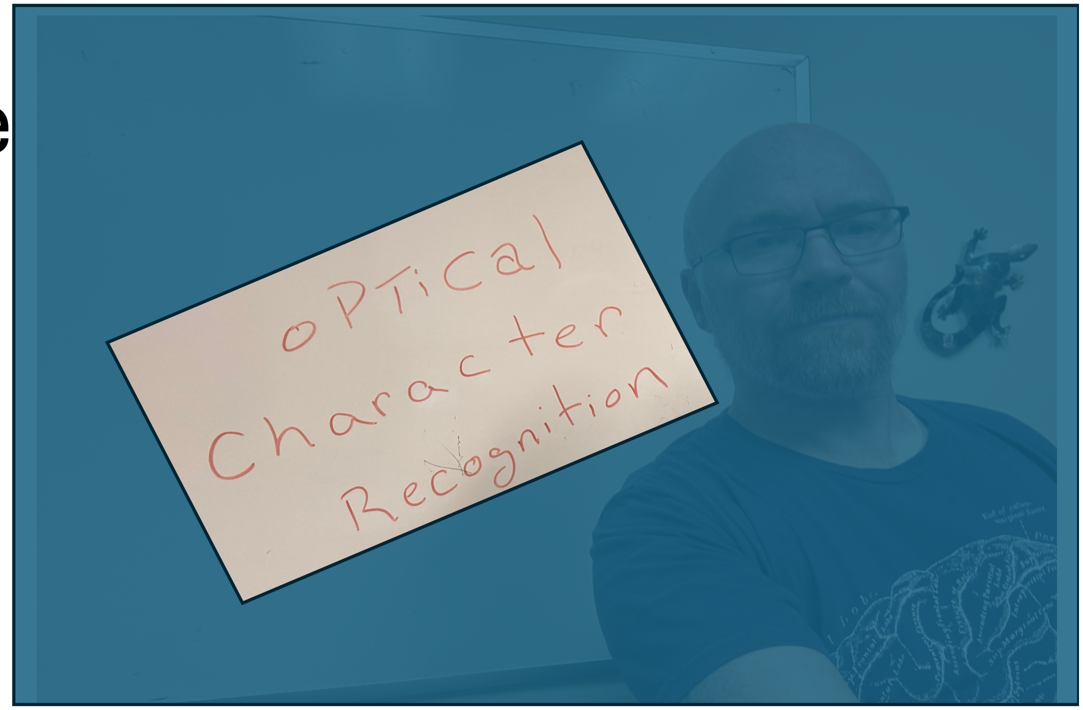
**Python Wrapper**

- https://pypi.org/project/pytesseract/

# Tesseract OCR Engine

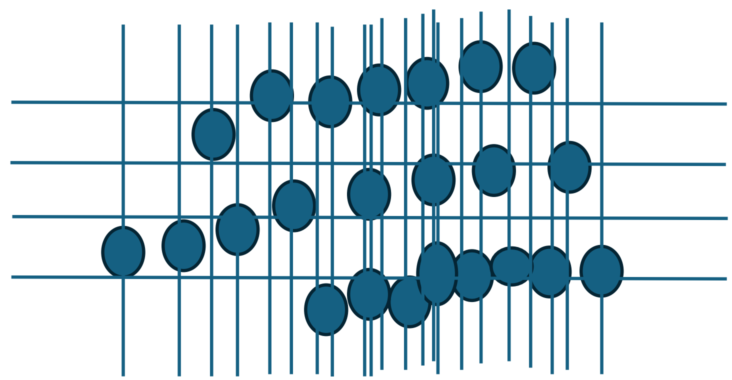
## Step 1. Line Finding

a) Find connected components in the image → "blobs"
b) Filter small/large blobs
c) Sort along horizontal axis
d) For left to right:
   If not blob overlaps row
      make new row
   Else
      expand vertical bounds of row
   end

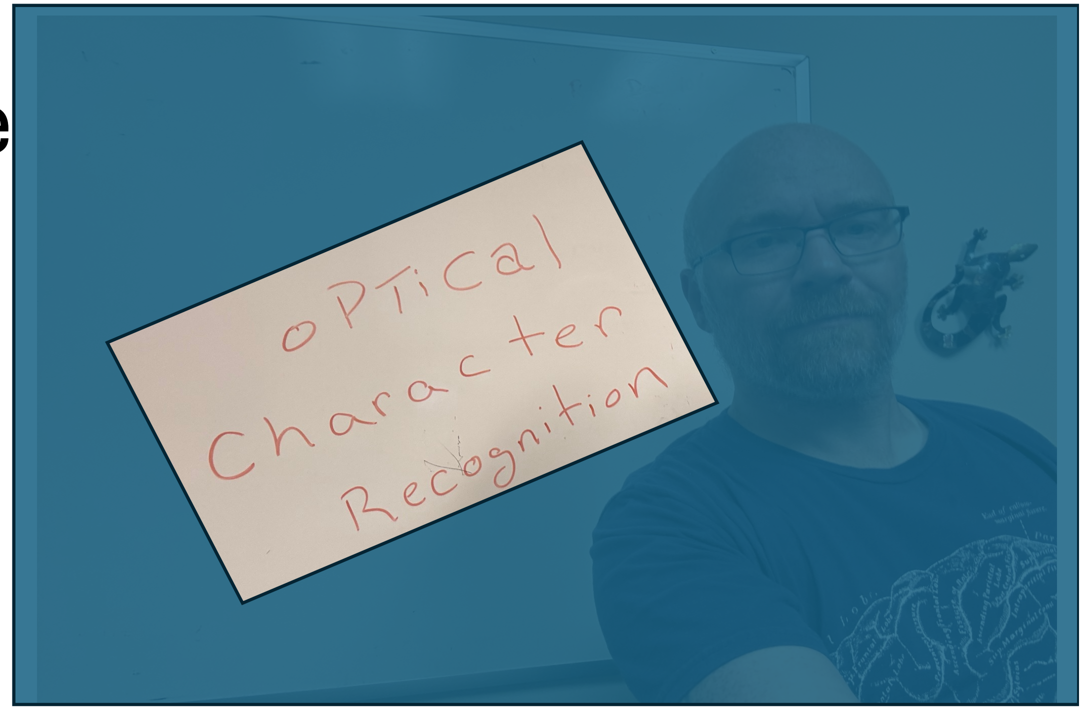



Row -1

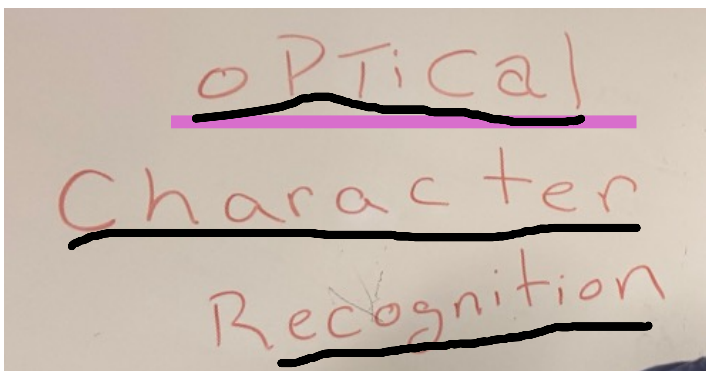
Row 0



# Tesseract OCR Engine




**Step 2. Baseline Fitting**



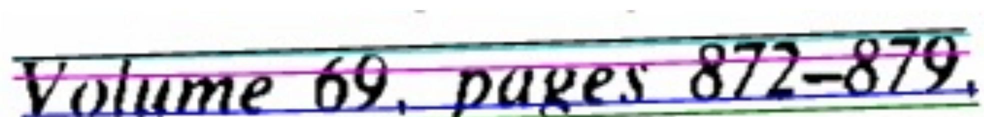
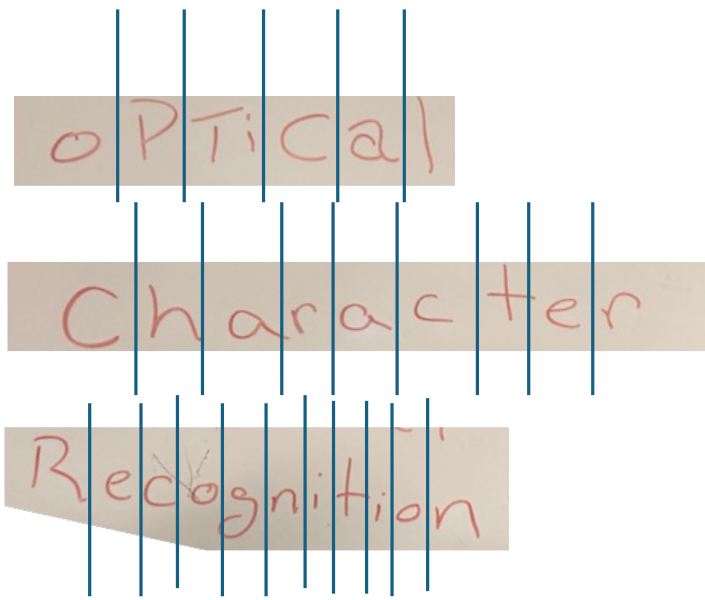
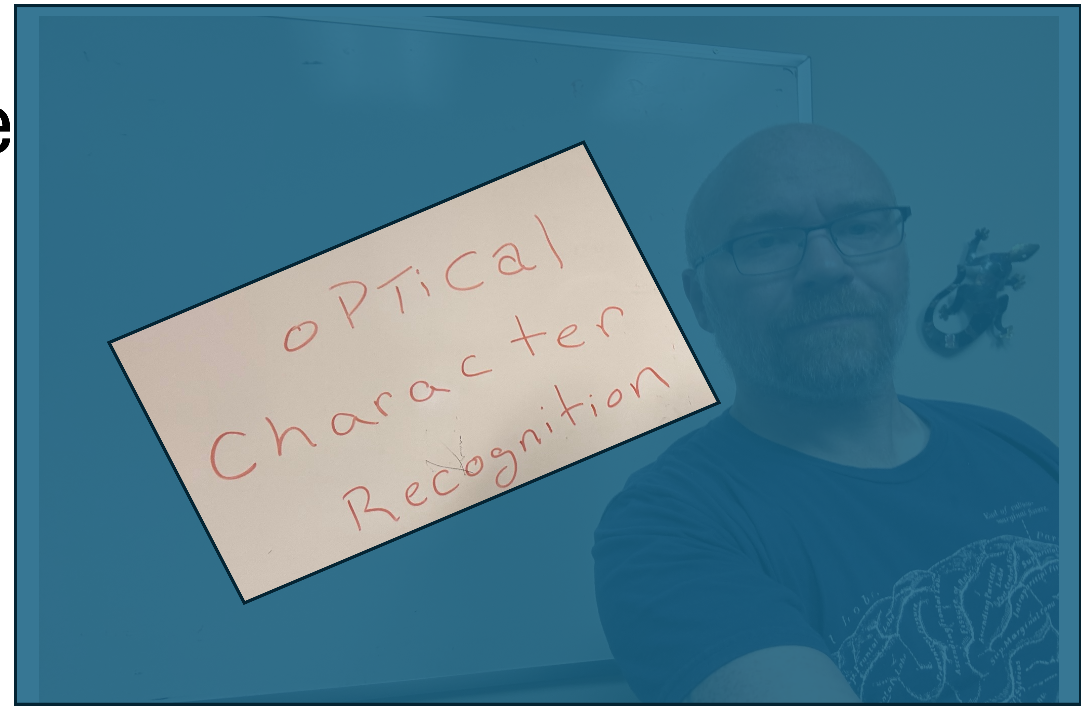




Fig. 1. An example of a curved fitted baseline.

# Tesseract OCR Engine

**Step 3. Pitch Detection and Chopping**



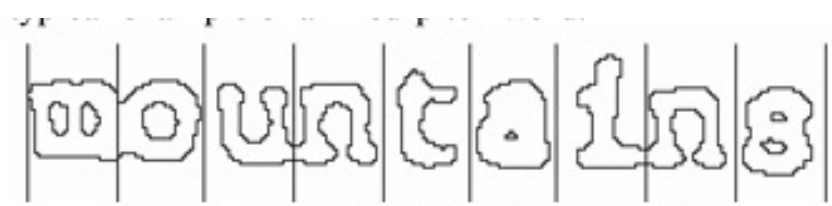



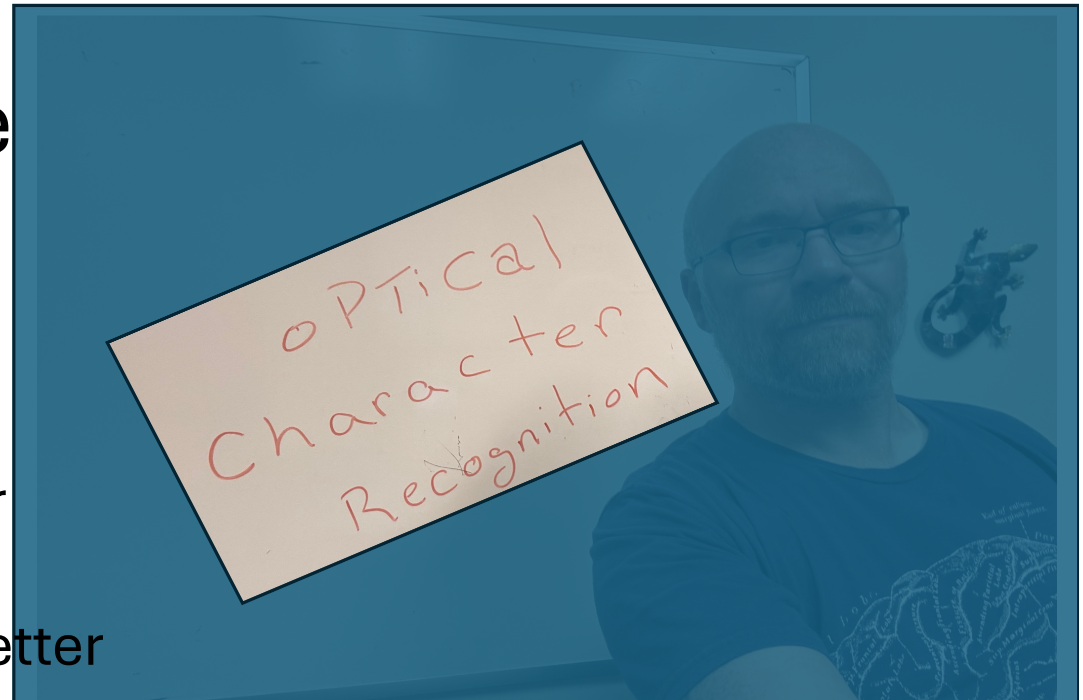
Fig. 2. A fixed-pitch chopped word.

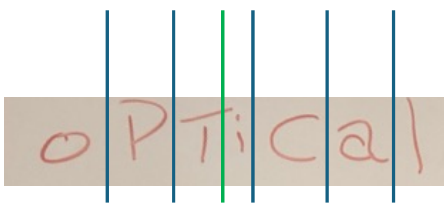
# Tesseract OCR Engine

**Loop.**

Classify blobs to letters

if blob doesn't match any letter
cut the blob
if cut blob doesn't match any letter
associate broken segments



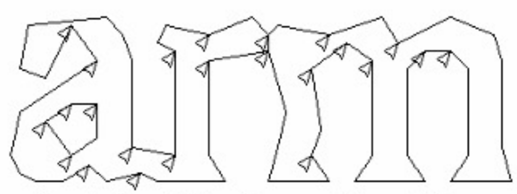



Fig. 4. Candidate chop points and chop.

Fig. 5. An easily recognized word.

# Limitations

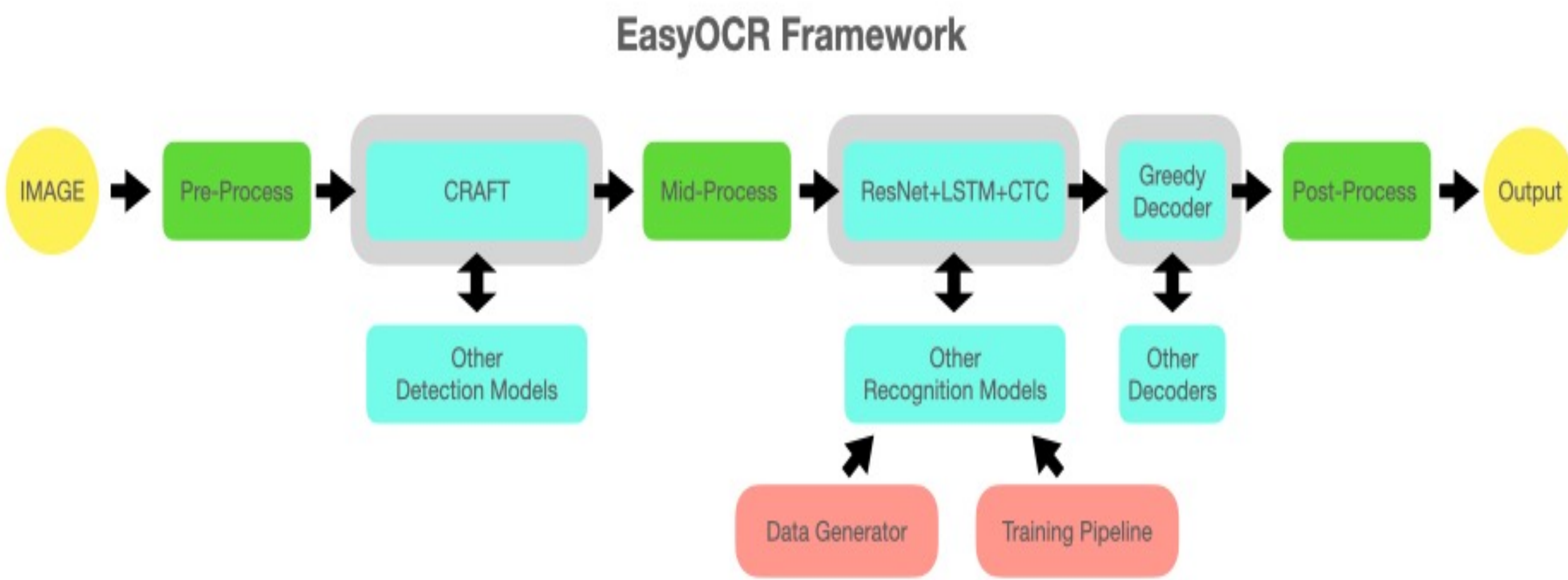
- Works best for printed characters (uniform spacing, uniform height, stylized lines)
- Can handle oblique lines (or curves in baseline; book bindings)
- Does not work with other stuff is in the image

# EasyOCR

- https://github.com/JaidedAI/EasyOCR
- pip install easyocr

- Trained on 80+ languages

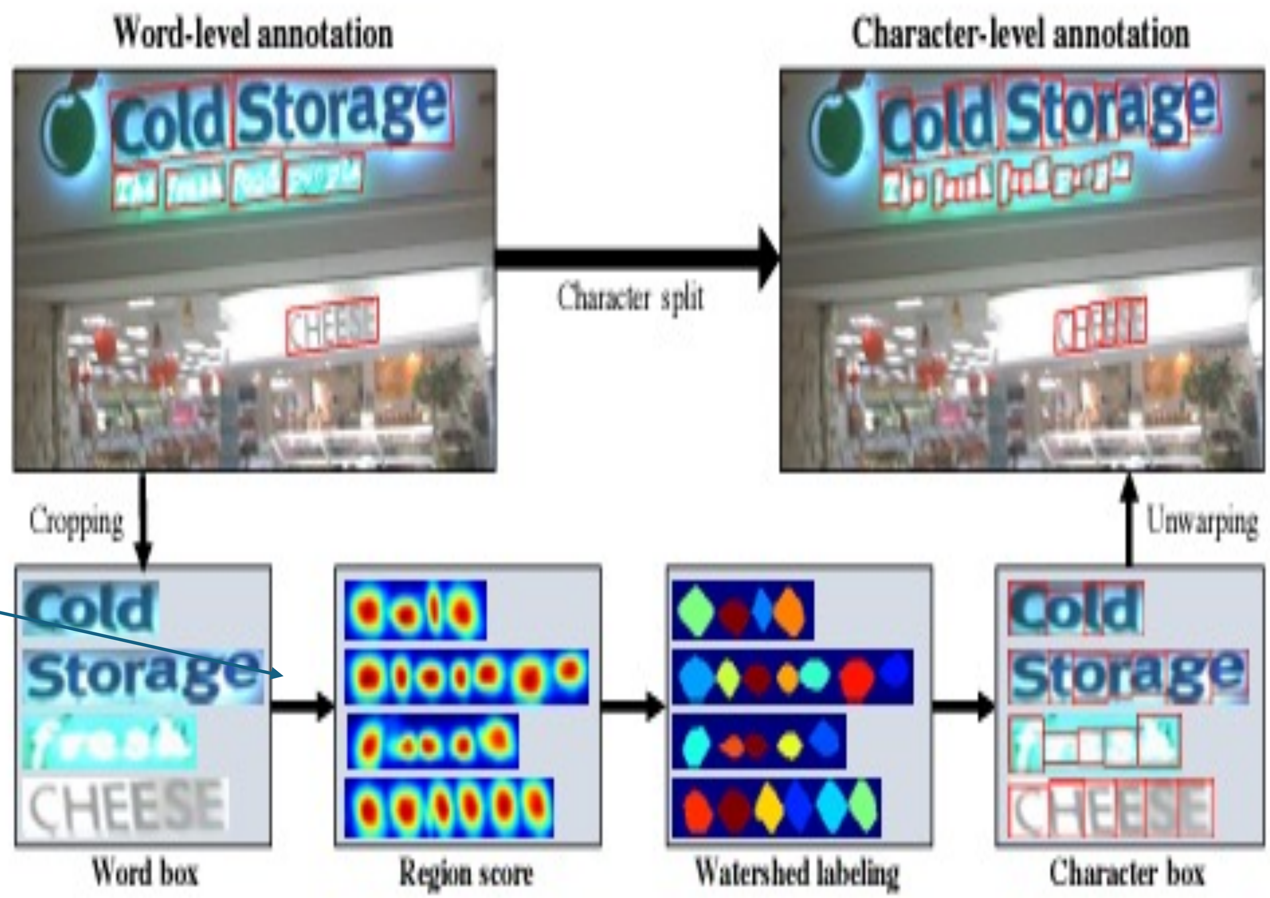
# EasyOCR


EasyOCR Framework
IMAGE
Pre-Process
CRAFT
Mid-Process
ResNet+LSTM+CTC
Greedy Decoder
Post-Process
Output
Other Detection Models
Other Recognition Models
Other Decoders
Data Generator
Training Pipeline

# EasyOCR

**CRAFT Detection**
(Character Region Awareness for Text)

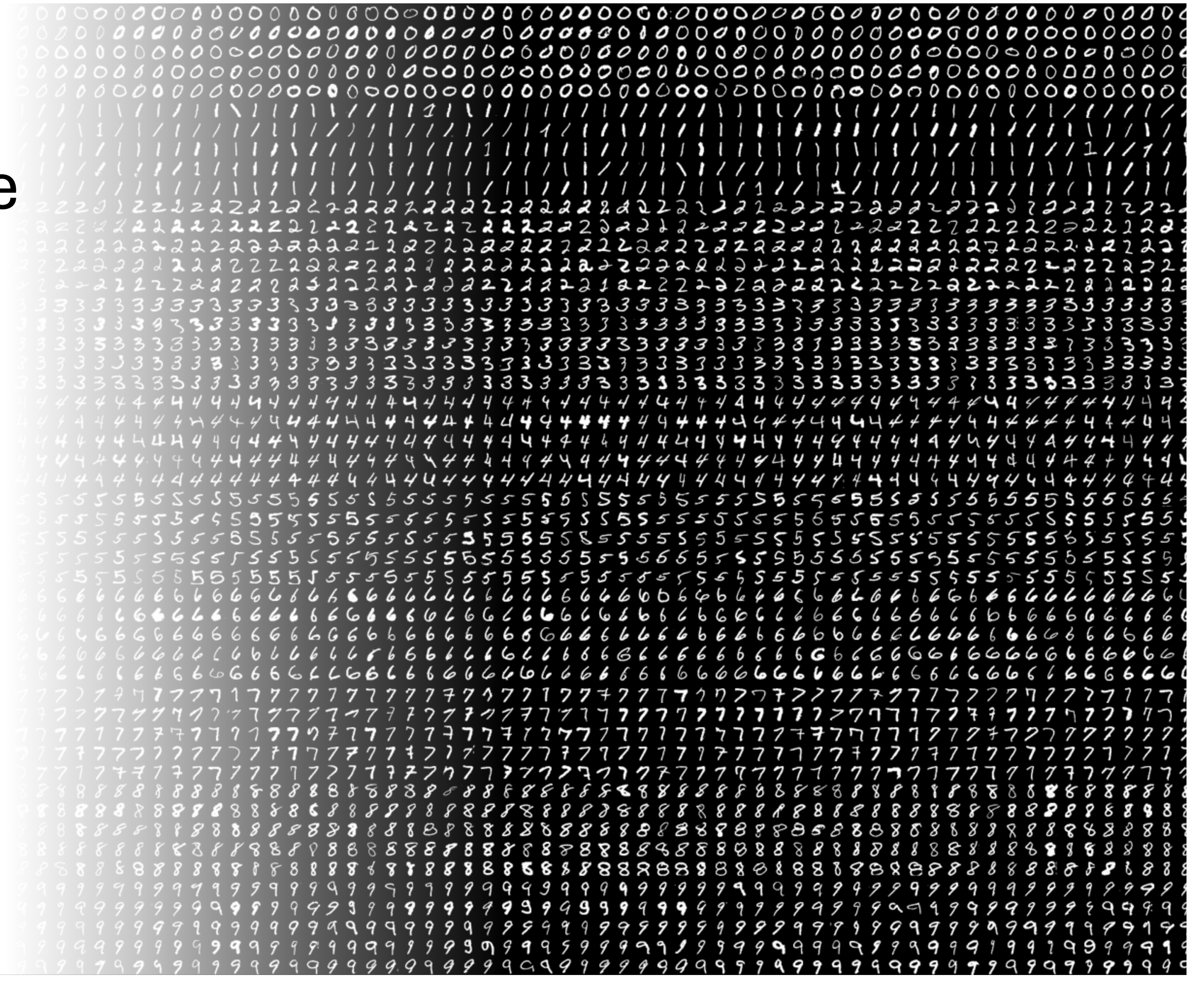
Trained U-net model to find center of character



# Limitations

- Works much better for real-world text
- Code is a lot slower than Tesseract

# Digits example



- From OpenCV Git samples
- Example of HOG classifier

```python
def preprocess_hog(digits):
        samples = []
        for img in digits:
                gx = cv.Sobel(img, cv.CV_32F, 1, 0)
                gy = cv.Sobel(img, cv.CV_32F, 0, 1)
                mag, ang = cv.cartToPolar(gx, gy)

                bin_n = 16
                bin = np.int32(bin_n*ang/(2*np.pi))
                bin_cells = bin[:10,:10], bin[10:,:10], bin[:10,10:], bin[10:,10:]
                mag_cells = mag[:10,:10], mag[10:,:10], mag[:10,10:], mag[10:,10:]
                hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in
                        zip(bin_cells, mag_cells)]
                hist = np.hstack(hists)

                eps = 1e-7
                hist /= hist.sum() + eps
                hist = np.sqrt(hist)
                hist /= norm(hist) + eps
                samples.append(hist)
        return np.float32(samples)
```

Compute the directions of edges in the image

Make histogram of the distribution of magnitudes found at each angle

Normalize histograms

```python
class SVM(object):
	def __init__(self, C = 1, gamma = 0.5):
		self.model = cv.ml.SVM_create()
		self.model.setGamma(gamma)
		self.model.setC(C)
		self.model.setKernel(cv.ml.SVM_RBF)
		self.model.setType(cv.ml.SVM_C_SVC)
	def train(self, samples, responses):
		self.model.train(samples, cv.ml.ROW_SAMPLE, responses)
	def predict(self, samples):
		return self.model.predict(samples)[1].ravel()
	def load(self, fn):
		self.model = cv.ml.SVM_load(fn)
	def save(self, fn):
		self.model.save(fn)
```

OpenCV is using a One-verses-All SVM model

```python
model = SVM(C=2.67, gamma=5.383)
model.train(samples_train, labels_train)
```

*From OpenCV’s Help description:*

Gamma:

**Role:** Defines the influence of a single training example. It controls the "width" of the RBF kernel (the most commonly used kernel with SVMs).

**Interpretation:**

- **Low gamma:** The influence of a single training example reaches far, leading to a smoother decision boundary and a more generalized model.
- **High gamma:** The influence of a single training example is localized, leading to a more complex decision boundary that closely fits the training data (potentially overfitting).

C:

**Role:** Controls the trade-off between maximizing the margin and minimizing the classification error.

**Interpretation:**

- **Low C:** A larger margin is encouraged, potentially leading to misclassifications on the training data but creating a simpler model that generalizes better.
- **High C:** The model tries to correctly classify all training examples, potentially leading to a more complex decision boundary and overfitting.