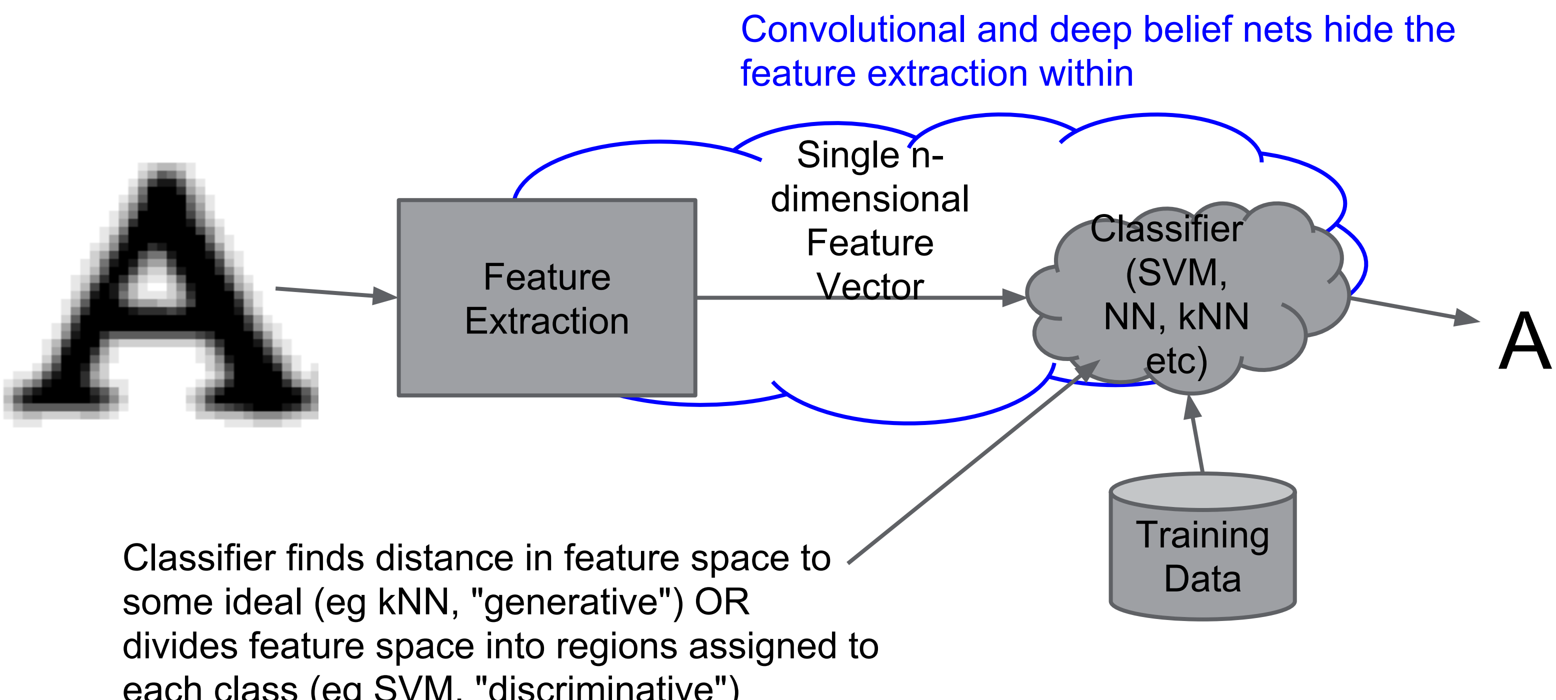
5. Features and Character Classifier

The real inside story

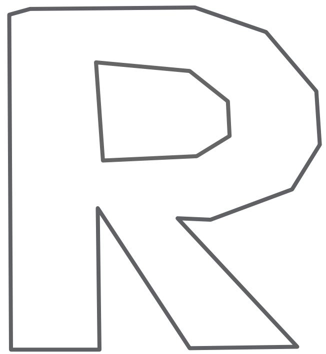
Ray Smith, Google Inc.

Background: Classical character classification

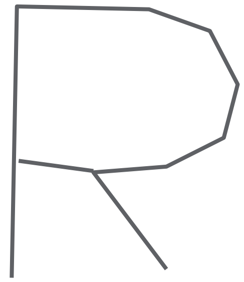


Motivation: How to extract features from Outlines?

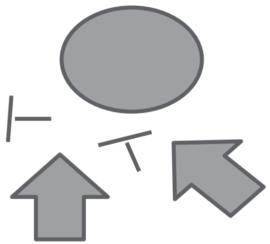
Outline



Skeleton



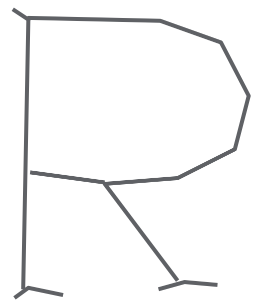
Topological Features



Skeletonization is Unreliable

Outline: Serifed

Skeleton: Decorated



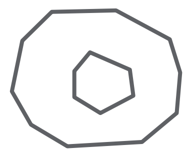
Arrrrh!

Lesson: If there are a lot of papers on a topic, there is most likely no good solution, at least not yet, so try to use something else.

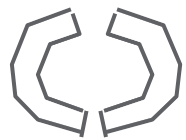
Topological features are Brittle

Damage to 'o' produces vastly different feature sets:

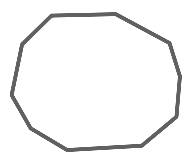
Standard 'o'



Broken 'o'



Filled 'o'

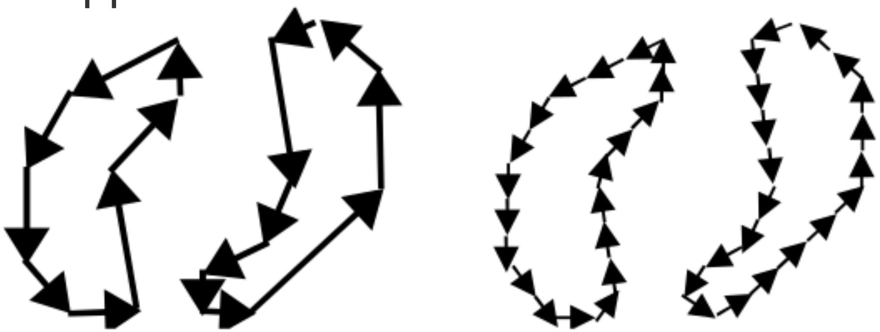


Lesson: Features must be as invariant as possible to as many as possible of the expected degradations.

Shrinking features and inappropriate statistics

Segments of the polygonal Approximation

Even smaller features



Statistical:

$$argmax(k)\prod_{l,i}\frac{1}{\sigma_{ijk}}\exp\left[-\frac{1}{2}\left(\frac{x_{il}-\mu_{ijk}}{\sigma_{ijk}}\right)^2\right]$$

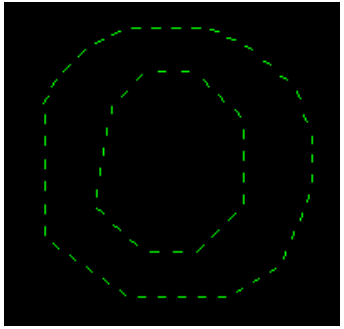
Geometric:

$$argmin(k)\frac{1}{M+J_k}\left(\sum_{l,i}(x_{il}-\mu_{ijk})^2+\sum_{j,i}(x_{il}-\mu_{ijk})^2\right)$$

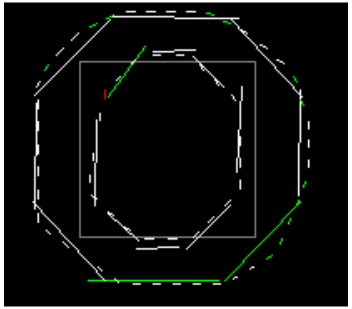
Lesson: Statistical Independence is difficult to dodge.

Inspiration: Even on a damaged character, most features still match if they are small!

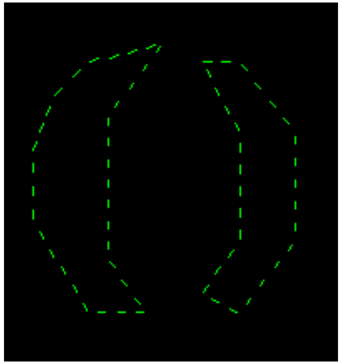
Features of clean ‘o’



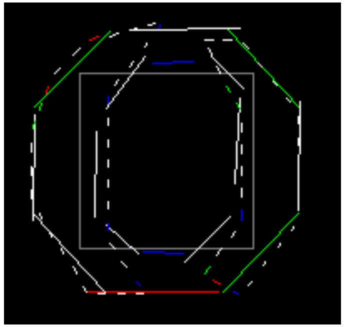
Matched with best template



Features of broken ‘o’



Mostly still matches



Interlude: Comparison with Recent Work

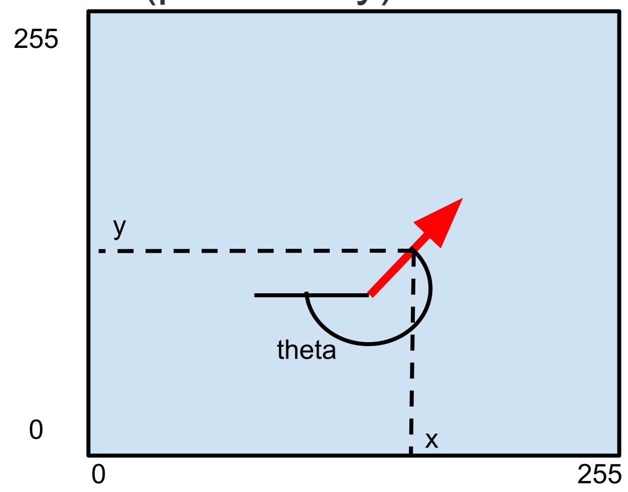
(Convolutional) Deep Belief nets:
1 pixel = 1 feature dimension
1 character (eg 32x32) = ~1K dimension feature vector

1.0
1.0
0.7
0.7
0.7
1.0
0.6
0.6
0.5
0.5
0.6
0.7
...

- Features are learned
- Usually edges
- Statistical dependence between pixels must also be learned: Purpose of network depth

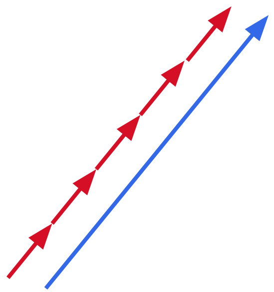
Features extracted from the unknown: 3D INT_FEATURE_STRUCT

- Multiple features extracted from a single unknown
- Each feature is a short, **fixed length**, directed, line segment, with (x,y) position and theta direction making a 3-D feature vector (x, y, theta) from an integer space [0, 255]
- Direction is measured (perversely) from the negative x-axis!



Features in the training data: 4D (hence Tesseract) INT_PROTO_STRUCT

- Elements of the polygonal approximation, clustered within a character/font combination.
- x,y position, direction, **and length** (as a multiple of feature length)



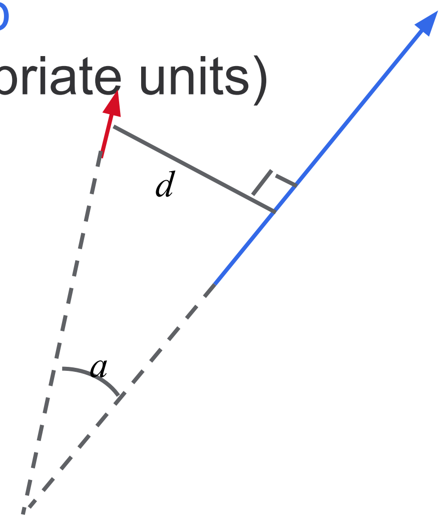
The Distance function: Single Feature to Single Proto

d = perpendicular distance of feature f from proto p

a = angle between feature f and proto p

Feature distance d_{fp} = $d^2 + a^2$ (in appropriate units)

Feature evidence e_{fp} = 1* / (1 + $kd_{fp}{}^2$)

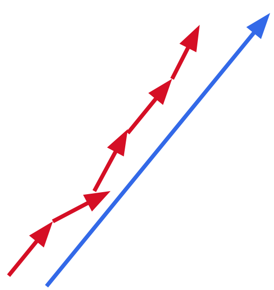


*In the actual implementation, everything is scaled up and run in integer arithmetic until the final result.

Feature Evidence and Proto Evidence
(For a single Font Config of a single Character Class)

Feature evidence $e_f = \max_{\text{p in config}} e_{fp}$

Proto evidence $e_p = \sum_{\text{top } l_p} e_{fp}$ (Proto p is of length l_p)



The CN (Character Normalization) Feature

Single 4-D feature for each unknown:

- Y-Position relative to baseline
- Outline Length (in normalized space)
- 2nd x-moment
- 2nd y-moment

The Distance Function: Unknown char to Prototype

$$d = 1 - \max \frac{\sum_f e_f + \sum_p e_p}{N_f + \sum_p l_p} \qquad d' = \frac{d l_o + kc}{l_o + k}$$

Feature-proto distance *CN correction*

l_o = Length of outline

c = Char position feature distance (CN feature)

k = classify_integer_matcher_multiplier (arbitrary constant = 10)

Rating = $d' l_o$

Certainty = $-20d'$

Rating *and* certainty? Why not just a “probability?”

- Rating = Distance * Outline length
  - Total rating over a word (or line if you prefer) is normalized
  - Different length transcriptions are fairly comparable
- Certainty = -20 * Distance
  - Measures the absolute classification confidence
  - Surrogate for log probability and is used to decide what needs more work.
- Comparing products of probability or sums of log probs of different length requires a non-rigorous hack anyway.

Now it's Too Slow!

~2000 characters per page x
~100 character classes (English) x
32 fonts x
~20 prototype features x
~100 unknown features x
3 feature dimensions

= 38bn distance calculations per page...

Now it's Too Slow!

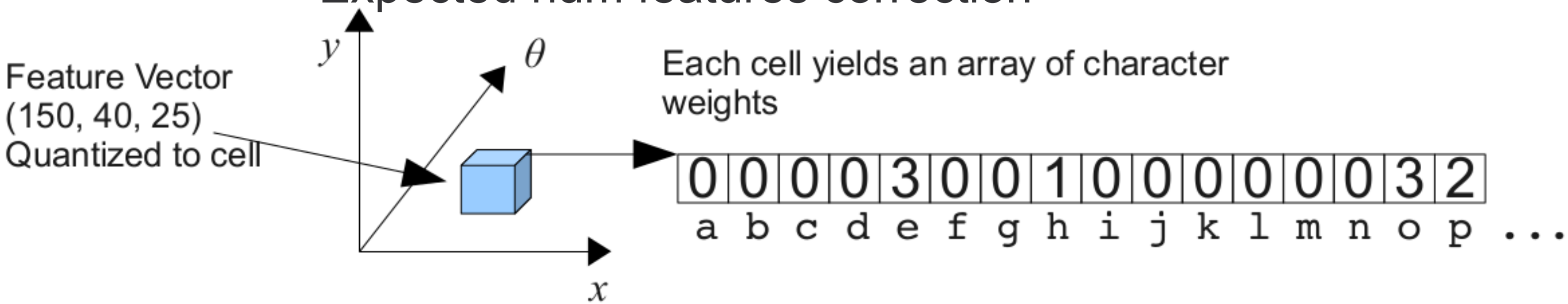
~2000 characters per page x
~100 character classes (English) x
32 fonts x
~20 prototype features x
~100 unknown features x
3 feature dimensions

= 38bn distance calculations per page...
... **on a 25MHz machine.**

Speeding up kNN: The Class Pruner

- Quantize feature space down from 256^3 to 24^3.
- Create inverted index: 3-D feature -> List of matching classes.
- Equivalent to a linear classifier with binary feature vector with 13824 dimensions.
- Fast, (~70 µs for Eng) but O(<num features> * <num classes>)
- Low top-n error rate (~0.01-0.5%), with low n (3-5)/110 even on unseen fonts, rising to 8% top-n on vastly different fonts.
- Top-1 error rate not so good at 8% typical.

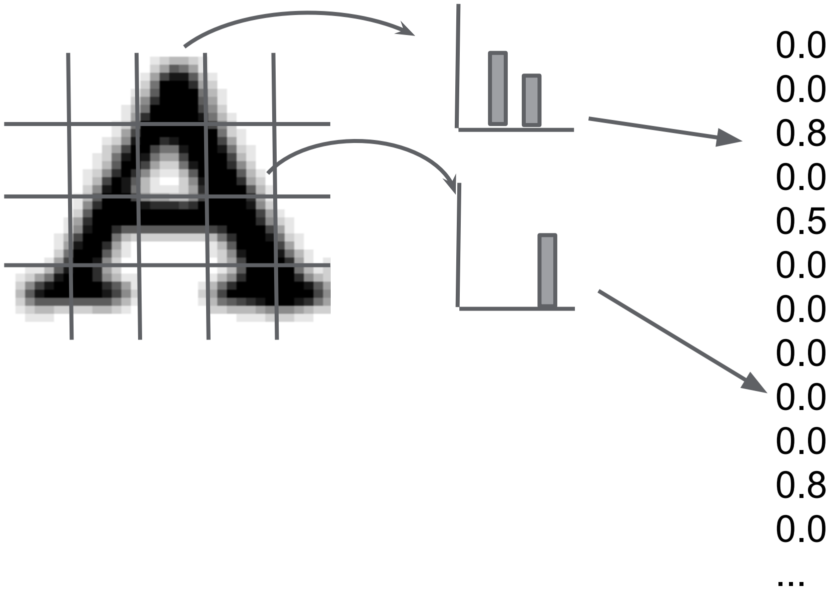
Secret sauces: 2-bit weights and spreading from the mean of the clusters.
Expected num features correction



Interlude: Comparison with Recent Work

Histogram of Gradients

- Quantize character area
- Compute gradients within
- Histograms of gradients map to fixed dimension feature vector
- Remarkably similar to class pruner



Interlude: Comparison with Recent Work: kNN

- Much has been published on speeding up kNN, eg randomized hashing, locality sensitive hashing etc.
- Much has also been published on indexing to speed-up recognition.

Live Classifier Demo

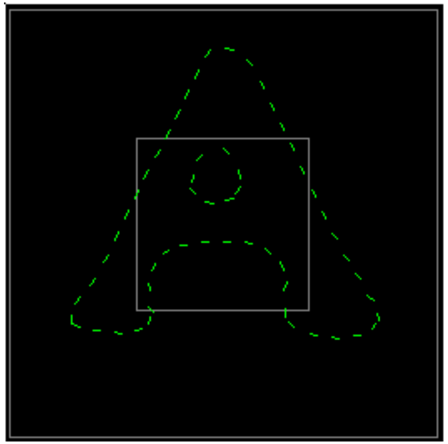
```
api/tesseract unlv/mag.3B/0/8050_078.3B.tif test1 inter matdemo
```

Easy ‘e’: Col 1, 1st line of non-italic, body text, word 2: ‘evaluate’
Italic ‘e’: Col 1, last line of italic question, word 1: *‘cycled’*
Hard ‘e’: Col 1, Question line 2, word 1: ‘the’
Joined pair: Col 1, line 5 of body text, word 2: *‘recycled’*

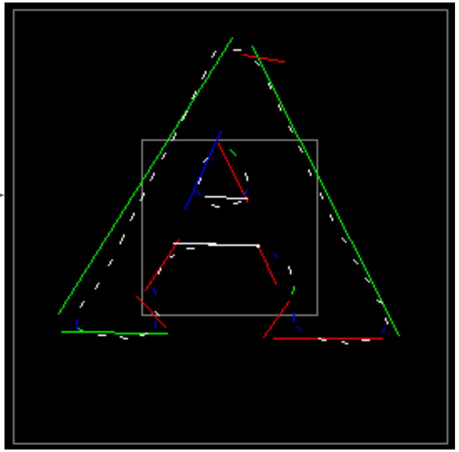
Real Classification Example

Insert live demo command line

Multiple (varied) features, each of 3 dimensions (x, y, direction), of unit length.



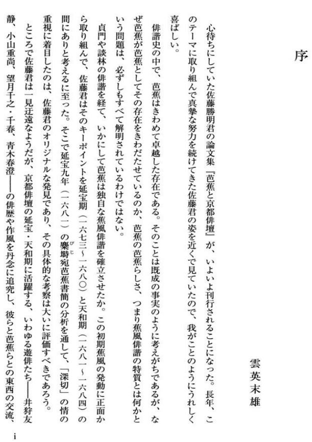
Classifier measures overall geometric similarity to closest character/font combination (kNN, generative).



Normalization

There are many stages of Normalization in Tesseract:

Image



序

雲英末雄

心待ちにしていた佐藤勝明君の論文集『芭蕉と京都俳壇』が、いよいよ刊行されることになった。長年、このテーマに取り組んで真摯な努力を続けてきた佐藤君の姿を近くで見ていたので、我がことのようにうれしく喜ばしい。

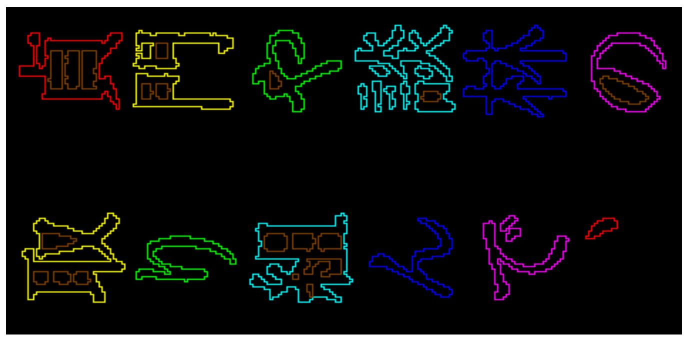
俳諧史の中で、芭蕉はきわめて卓越した存在である。そのことは既成の事実のように考えがちであるが、なぜ芭蕉が芭蕉としてその存在をきわだたせているのか、芭蕉の芭蕉らしさ、つまり蕉風俳諧の特質とは何かという問題は、必ずしもすべて解明されているわけではない。

貞門や談林の俳諧を経て、いかにして芭蕉は独自な蕉風俳諧を確立させたか。この初期蕉風の発動に正面から取り組んで、佐藤君はそのキーポイントを延宝期（一六七三～一六八〇）と天和期（一六八一～一六八四）の間にありと考えるに至った。そこで延宝九年（一六八一）の麋塒宛芭蕉書簡の分析を通して、「深切」の情の重視に着目したのは、佐藤君のオリジナルな発見であり、その具体的な考察は大いに評価すべきであろう。

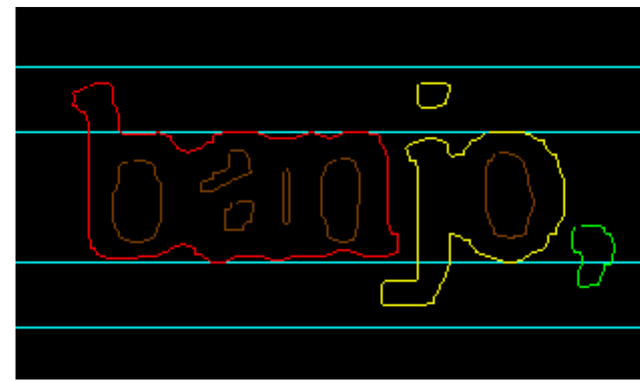
ところで佐藤君は一見迂遠なようだが、京都俳壇の延宝・天和期に活躍する、いわゆる遊俳たち――井狩友静、小山重尚、望月千之・千春、青木春澄――の俳歴や作風を丹念に追究し、彼らと芭蕉らとの東西の交流、

i

Rotated



Word BLN



Google

Normalization Rules

Image:

- Only the API knows of possible scaling and cropping of the source image. Inside Tesseract, the image is not touched.

C_BLOBs:

- Constrained to be pixel-oriented, they cannot be scaled or rotated other than by multiples of 90 degrees.
- The only difference between C_BLOBs and the image is a possible block rotation to keep textlines horizontal.

TBLOBs:

- Begin life as Word-Baseline-Normalized, these are the input to the chopper and classifier.
- There may be an additional rotation for classification. (CJK)

Classifier Normalizations

Beginning with a word-baseline-normalized TBLOB (possibly rotated again to be the right way up) the classifier further normalizes for feature extraction:

- Baseline Norm: No further scaling, but x-center the character in the classifier feature space.
- Character Norm: Center the character in the feature space by centroid, and scale by 2nd moments anisotropically.
- Non-linear Norm: Scale to preserve edge density in a non-linear way to fill the feature space in some sense. (Not used, but maybe in the future.)

Applications of different Normalizations

Baseline/x-height

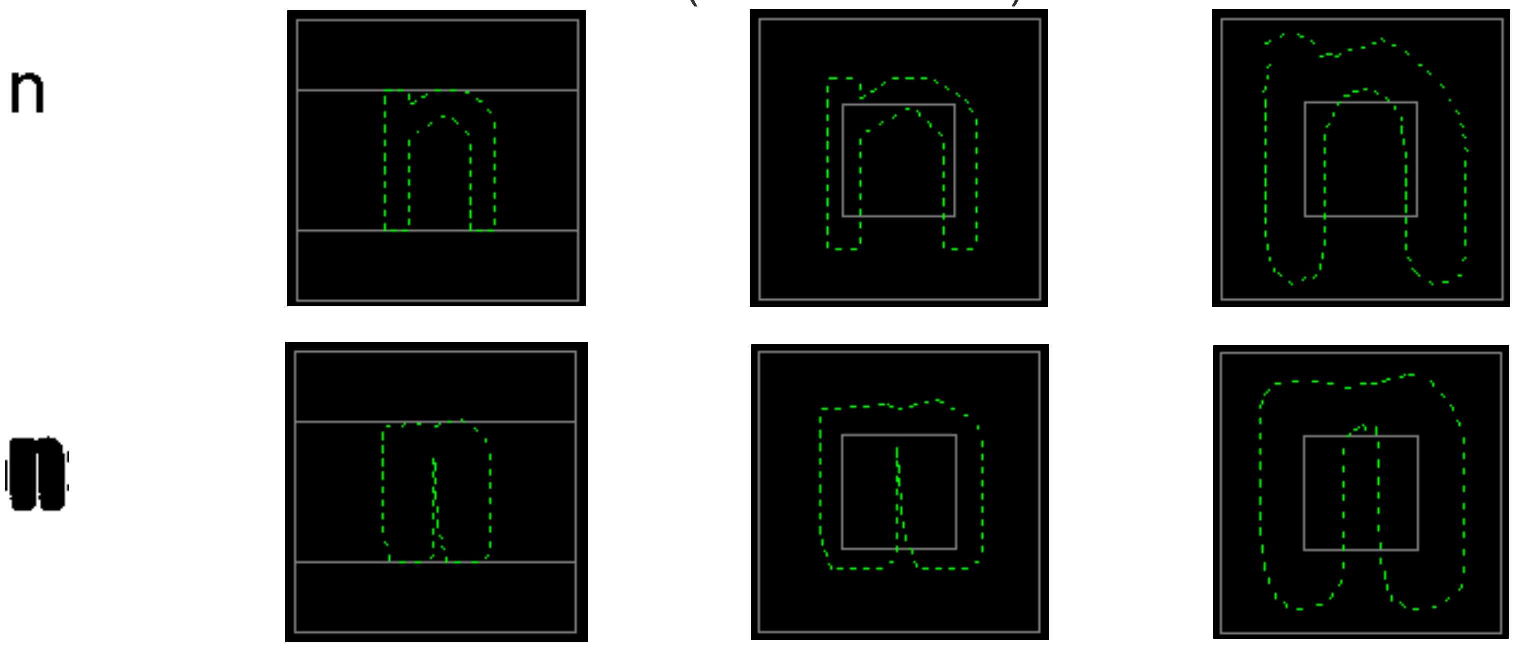
- Used by Adaptive classifier
- Align on x centroid, y on baseline
- Scale uniformly by xheight.
- Sees sup/super as different classes.
- Ignores speckle noise well.

Character Moments

- Used by static classifier.
- Align on Centroid
- Scale by 2nd moments independently in x and y
- Eliminates a lot of font variation.
- Makes '-' '.' '_' 'l' ambiguous.
- Makes sub/superscript appear same as normal
- Fooled by speckle noise

Character-level Normalization

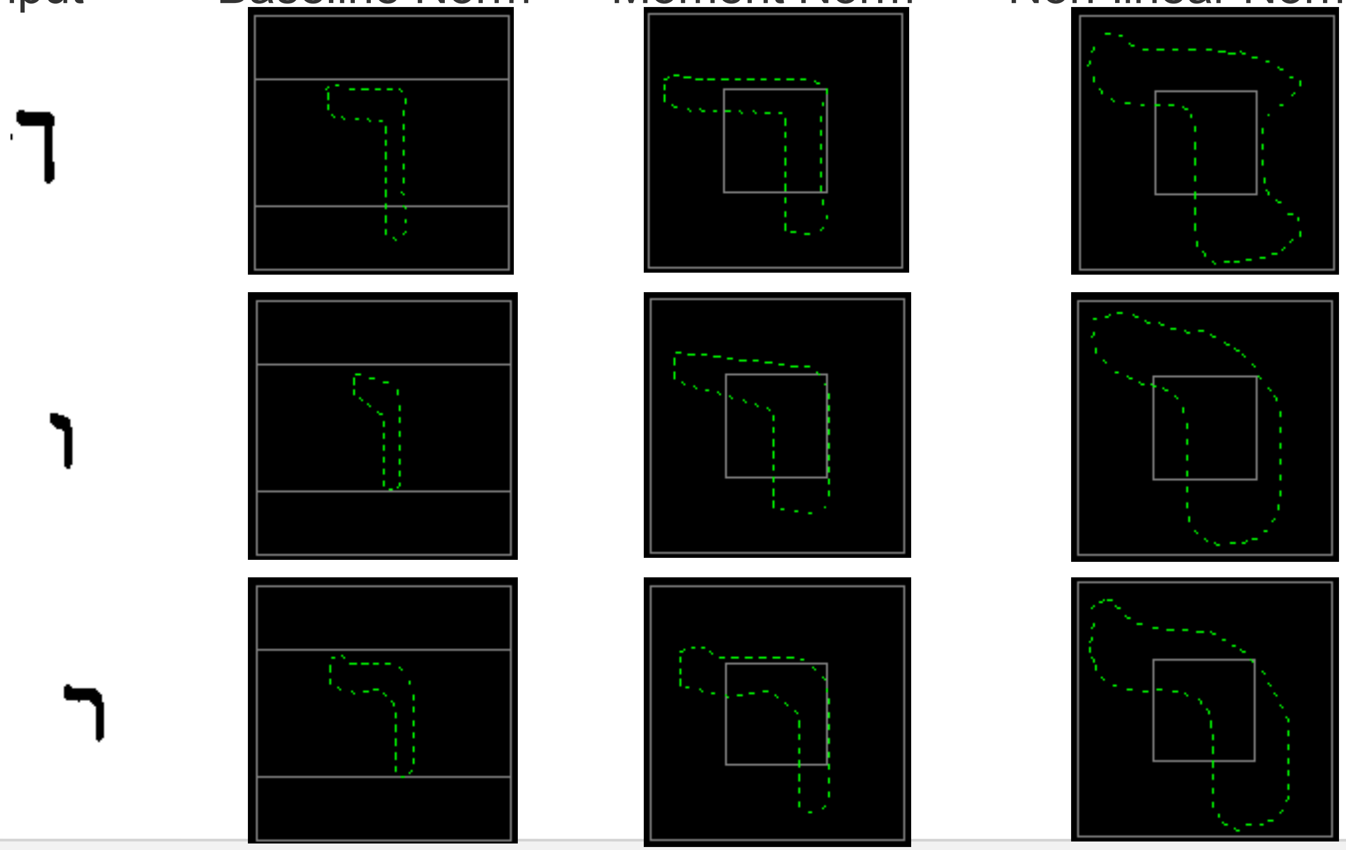
Input	Baseline-norm	Moment-norm (aka char-norm)	Non-linear norm



Google

Character-level Normalization (Hebrew example)

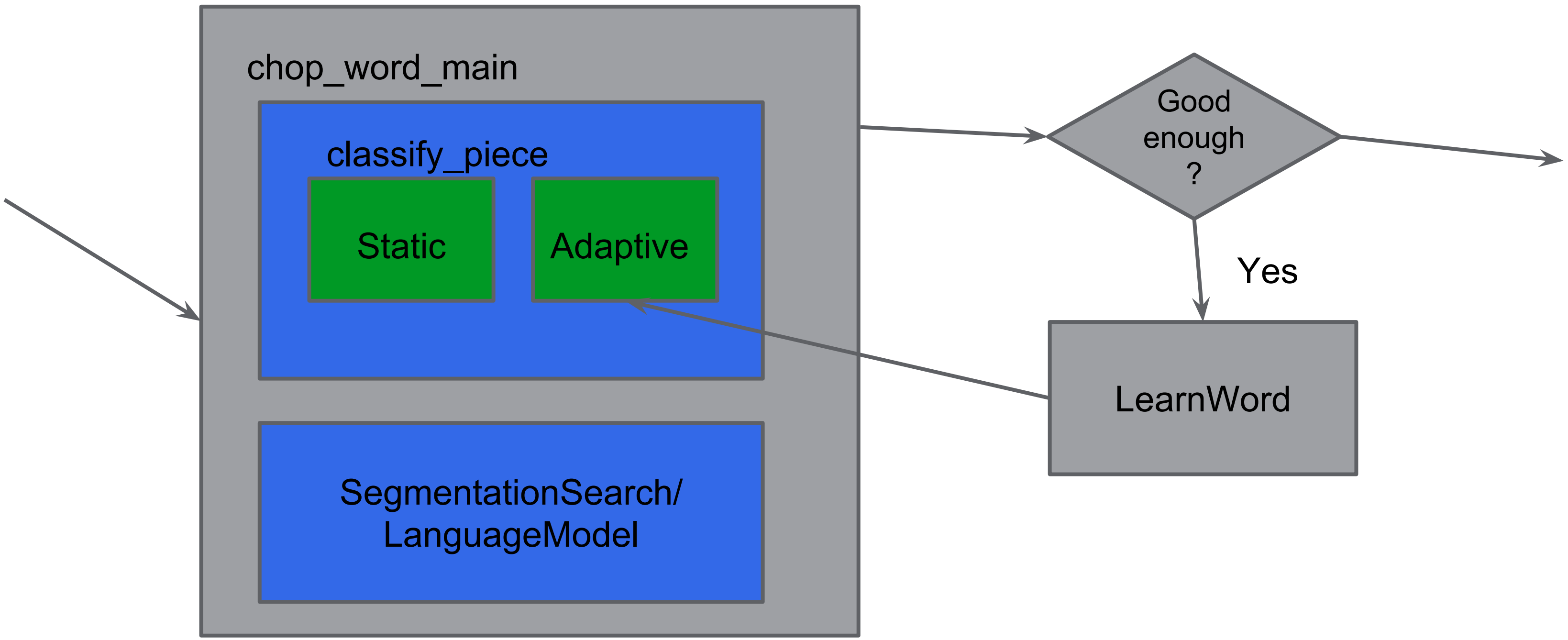
Input | Baseline Norm | Moment Norm | Non-linear Norm



Google

Adaptive Classifier: On-the-fly Adaption

classify_word_pass1



Thanks for Listening!

Questions?