A Hypothesis-Based Approach To Offline Handwriting Recognition

(Using Convolutional Neural Networks)

Written by
Tom G. Grigg

Supervised by
Dr. Mark J. Wheelhouse

What is *Offline* Handwriting Recognition?

Online

Feature Sequence Recogniser

hey

Offline

Image Recogniser

hey

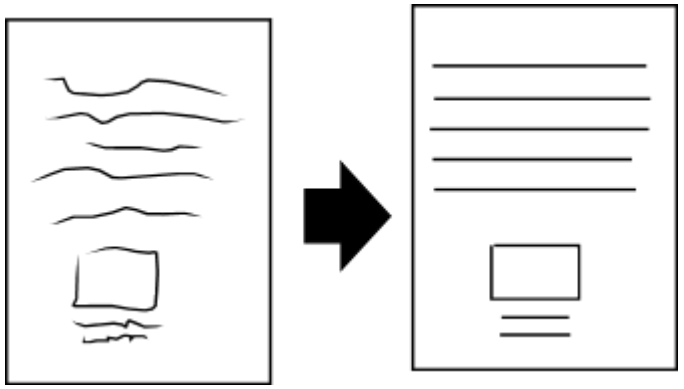
Online vs Offline

- Online recognition is easier (considered solved)
    - Given an explicit position-time sequence.

- Offline recognition much harder
    - Given only the image (no clear and natural ordering of information).

Document Recognition

- Offline Handwriting Recognition is a subfield of the field of Document Recognition



- Document Recognition – aims to transcribe entire handwritten documents – handwriting, mathematics, chemical/biological/engineering diagrams into digital format.

Project Focus: improving output accuracy of an underlying offline handwriting recogniser.

- Why? Existing classifiers seem to have unsatisfactory baseline accuracy.

- We are able to train systems to have a go at solving these problems but cannot rely on deep networks/self-learning algorithms to do all of the hard work – they are not ‘intelligent’ enough (yet!)

- So we install an in-built algorithm into the ‘mind’ of the recogniser to best handle uncertainty.

Proposition:

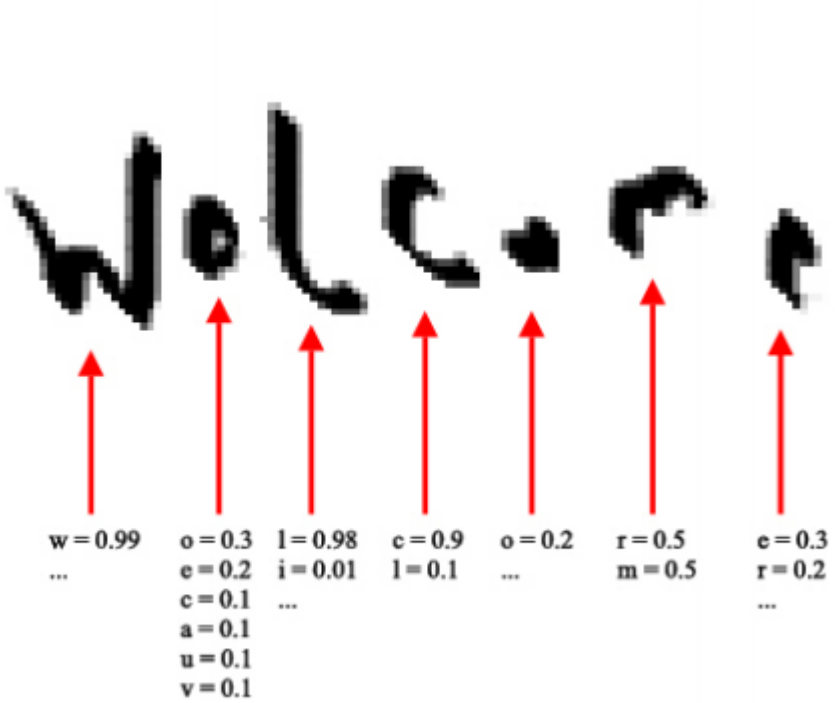
A Hypothesis-Based Approach

To decrypt a given string of handwritten characters:

1. Realise each character as its most likely value.
2. Is the resulting string (close to) a word? Yes -> 4 Otherwise…
3. Realise the next most likely string. -> 2.
4. Return this word.

Proposition:

A Hypothesis-Based Approach



"Wolcore" – 0.07

> Reject

"Wolcome" – 0.07

> Close to "welcome" (accept).

(In fact, next most likely word is "welcome" or "wolcomr" with 0.04)

Character-likelihood function

Essentially, we want a function $\mu : \text{Image} \rightarrow [0, 1]^{26}$

such that $\mu(i) = (\mu_a(i), \mu_b(i), \ldots, \mu_z(i))$

where $\mu_a(i)$ = likeness to the character ‘a’; 0 = certainly not ‘a’; 1 = certainly ‘a’.

(Fuzzy Classification)

Proposition:

A Hypothesis-Based Approach

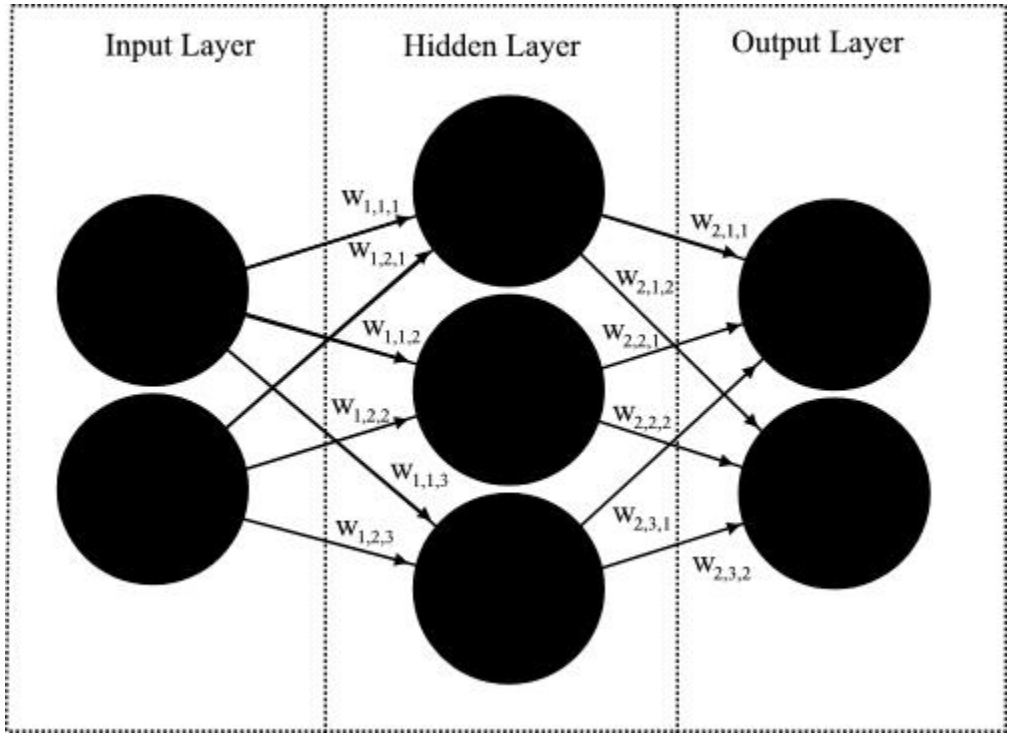
To decrypt a given string of handwritten characters:

1. Realise each character as its highest μ-value.
2. Is the resulting string a word? Yes -> 4 Otherwise…
3. Realise the string with next highest μ-value. -> 2.
4. Return this word.

Standard Approach to Offline HWR:

Convolutional NNs

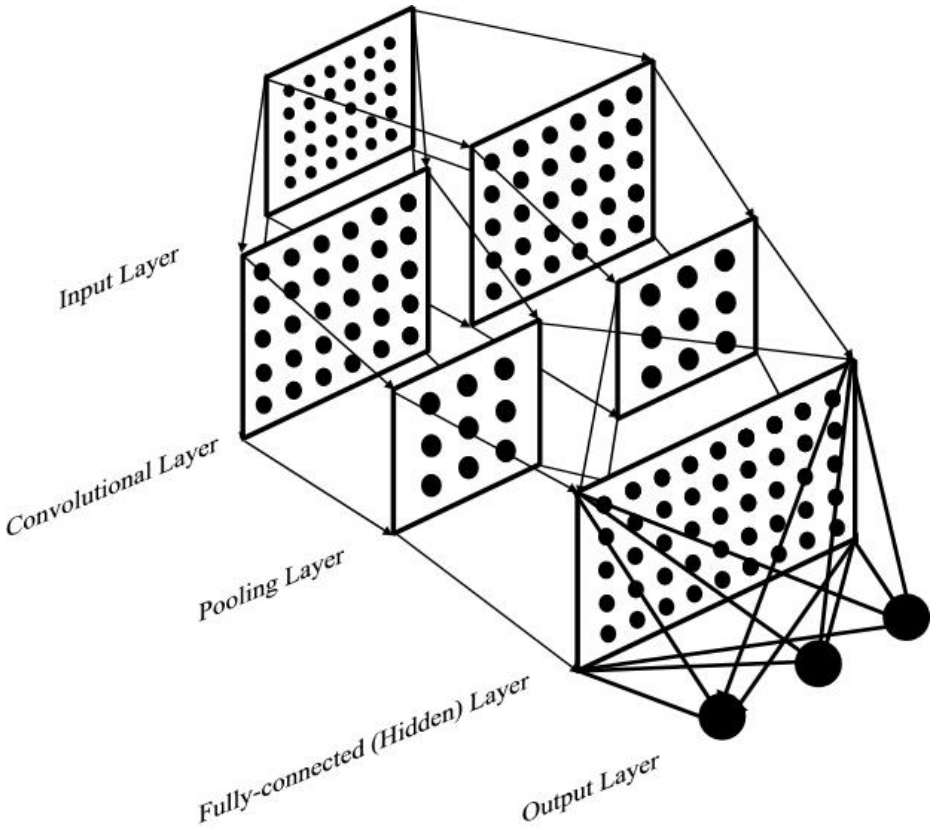
Neural network: reasoning machine implemented as a graph with weighted edges that can be updated via gradient descent to minimise an error function. A model of learning that does not require explicit feature definition from human trainers – no idea what the network learns.



Standard Approach to Offline HWR:

Convolutional NNs

- CNNs inspired by mammalian visual cortex. Three layer types: Convolutional, Pooling, Classification/Fully-connected.

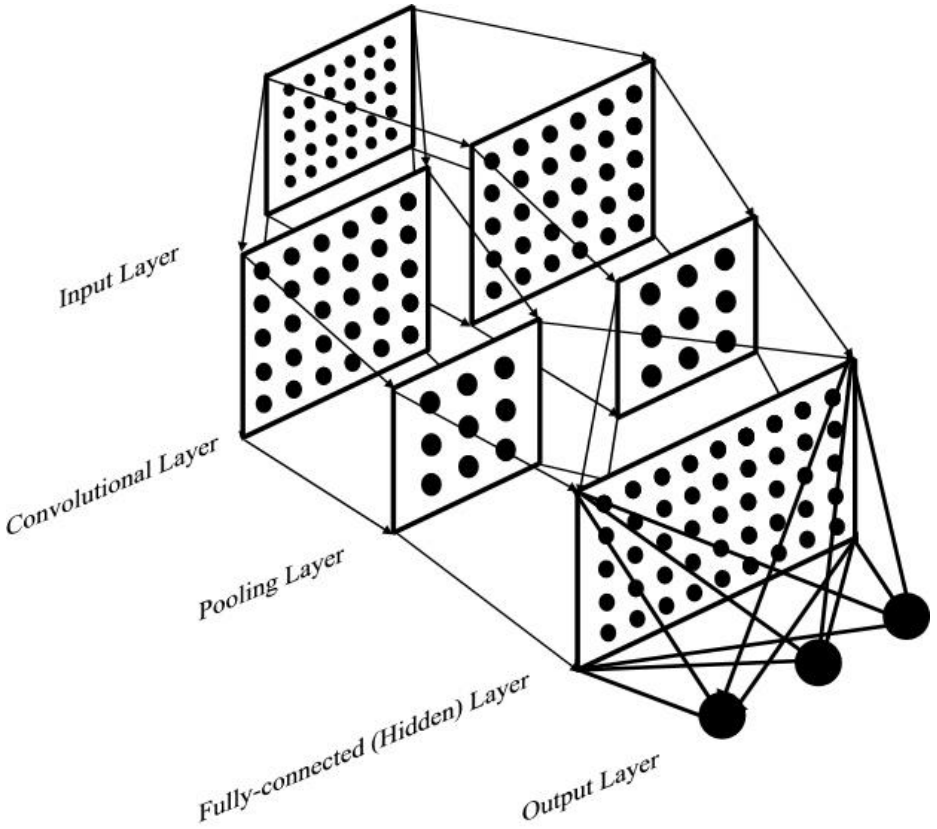


Standard Approach to Offline HWR:

Convolutional NNs

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j \in output} e^{x^j}}$$

- Use CNN with softmax activation function as a likelihood function approximator



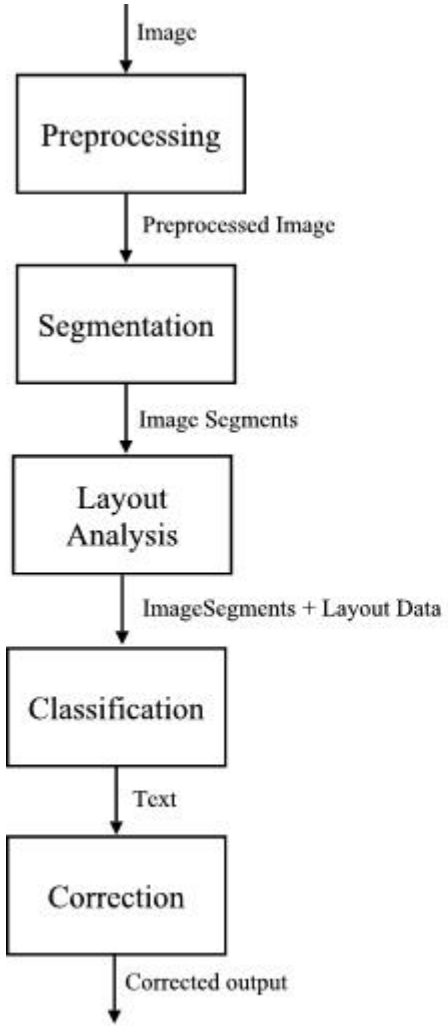
Building an offline handwriting recognition system

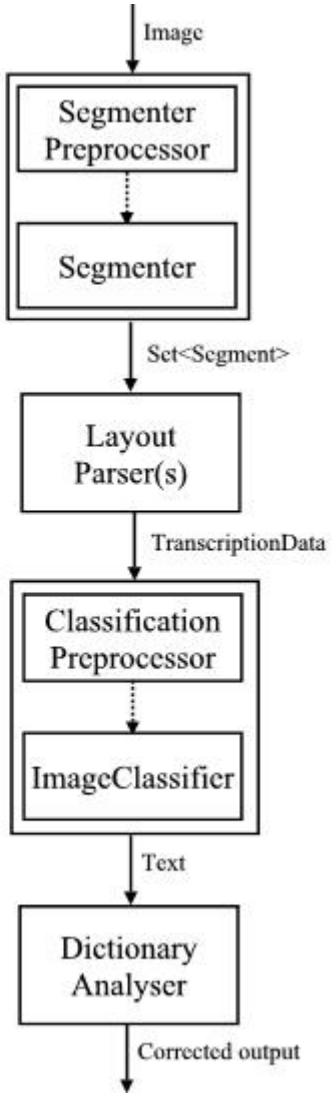
- Simplification: disconnected character only (suitable for the purposes of evaluating this method).

- Simplification: heuristic layout analysis, operate only on lower case English characters

- Purpose: evaluate this approach.

System Design

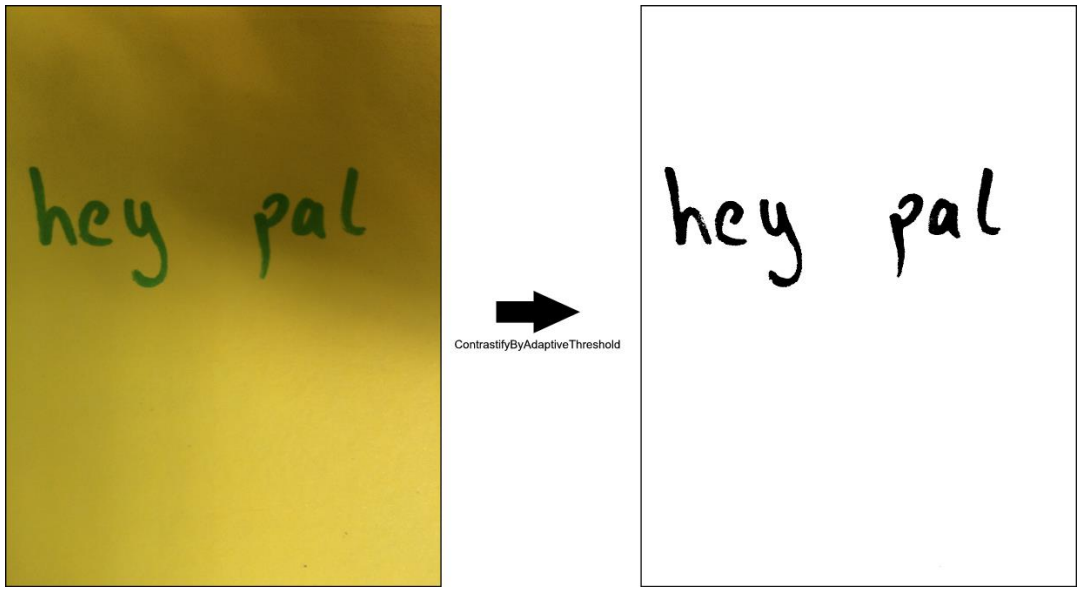




Problems:

Preprocessing

- Best method: Adaptive thresholding



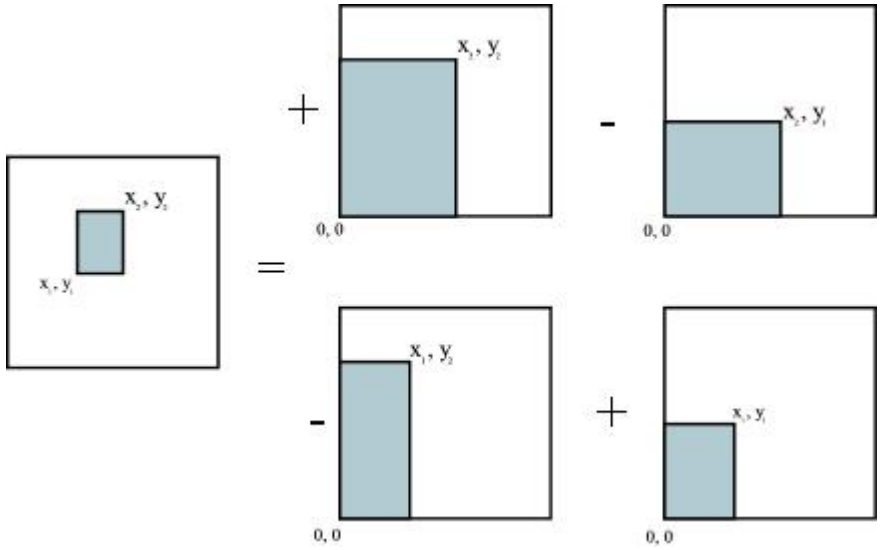
The variant we implemented uses the **integral image** for efficient calculation of local pixel-value averages.

Problems:

Preprocessing

- Best method: Adaptive thresholding

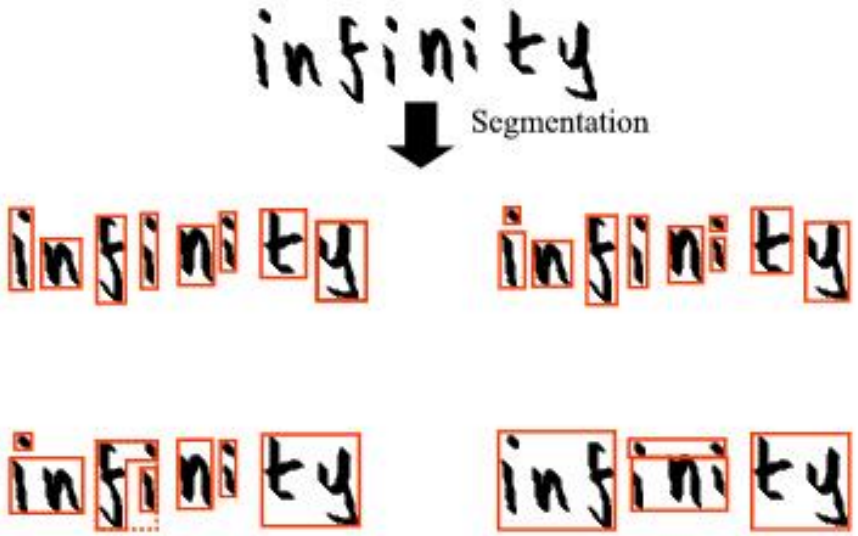
```
for i = 0..width do
    sum ← 0
    for j = 0..height do
        sum ← sum+inputImage[i][j]
        if i = 0 then
            integralImage[i][j] ← sum
        else
            integralImage[i][j] ← integralImage[i-1][j]+sum
        end if
    end for
end for
for i = 0..width do
    for j = 0..height do
        x1 ← i - s/2
        x2 ← i + s/2
        y1 ← j - s/2
        y2 ← j + s/2
        count ← (x2 - x1)*(y2 - y1)
        sum ← integralImage[x2][y2] - integralImage[x2][y1]
        - integralImage[x1][y2] + integralImage[x1][y1]
        if (inputImage[i][j]*count) ≤ (sum*(100 - t)/100) then
            outputImage[i][j] ← BLACK
        else
            outputImage[i][j] ← WHITE
        end if
    end for
end for
```



Problems:

Segmentation

- (Character) Segmentation refers to the problem of isolating characters for classification.

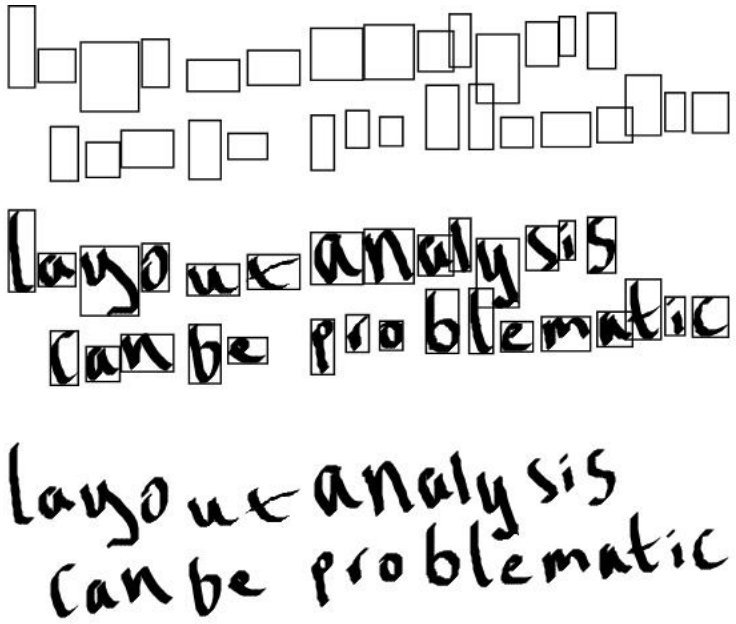


- Best method: Whitespace segmentation with distance heuristics to connect disconnected character segments ‘i’ and ‘j’ dots.

Problems:

Layout Analysis

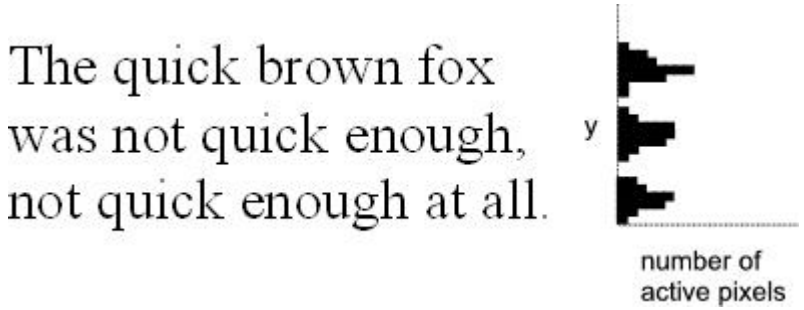
- Layout analysis refers to the deduction of what constitutes a paragraph, line, word within an image of handwritten text. We kept things simple and handled only line and word analysis.

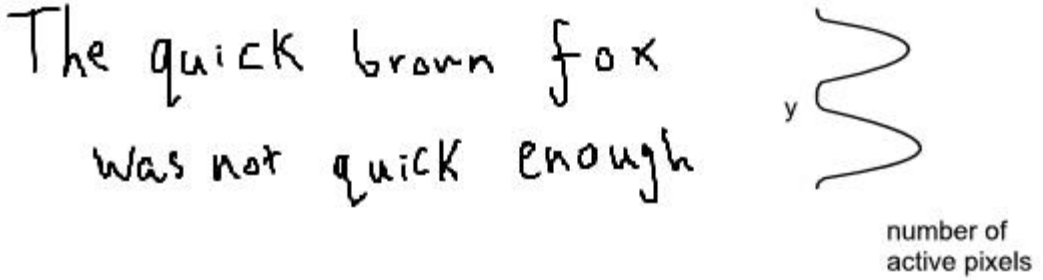


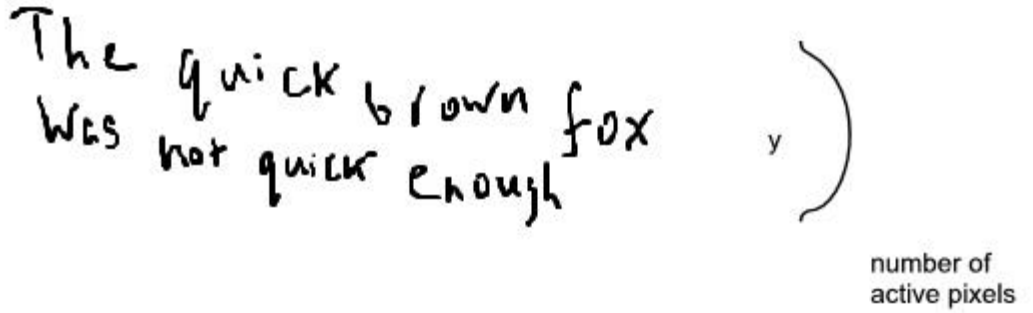
Problems:

Layout analysis: lines

- Standard method: horizontal histogram analysis.







Problems:

Layout analysis: lines

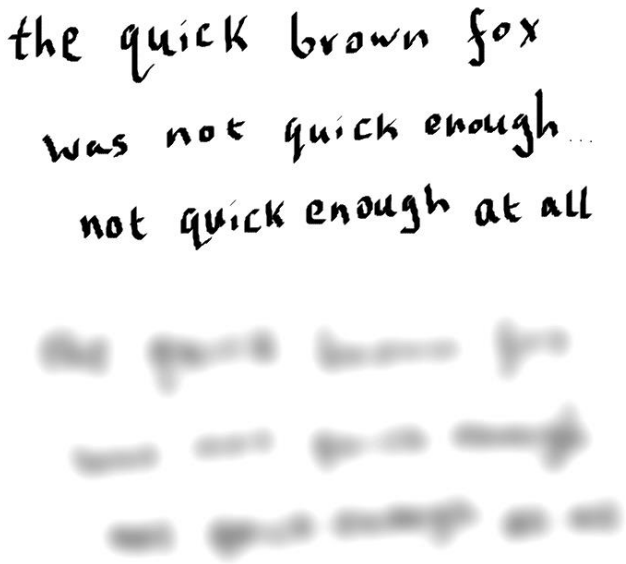
- Better: local-position scheme


```
averageSegmentHeight ← getAverageSegmentHeight(segments)
sort segments by their horizontal position, from left to right.
seg ← getTopLeftSeg(segments)
add seg to newSegments
remove seg from segments
topBound ← (getAverageSegmentHeight(segments)/2) - seg.y
bottomBound ← (getAverageSegmentHeight(segments)/2) - seg.y
while segments is non-empty do
   if there exists a segment cSeg s.t. seg.y - bottomBound ≤ cSeg.y ≤ seg.y + topBound, cSeg to
the right of seg then
      nextSeg ← leftmost such cSeg
      add nextSeg to newSegments
      remove nextSeg from segments
      if HEIGHT_FAC_HI*(topBound + bottomBound) ≥ nextSeg.height ≥ HEIGHT_FAC_LO*(topBound
+ bottomBound) then
         topBound ← nextSeg.top - nextSeg.y
         bottomBound ← nextSeg.y - nextSeg.bottom
      end if
      seg ← nextSeg
   else no such segment cSeg exists
      add newSegments.length - 1 to lineDividers
      seg ← getTopLeftSeg(segments)
      segments.remove(seg)
   end if
end while
segments ← newSegments
```

Problems:

Layout analysis: words

- Standard approach: smudging/blurring

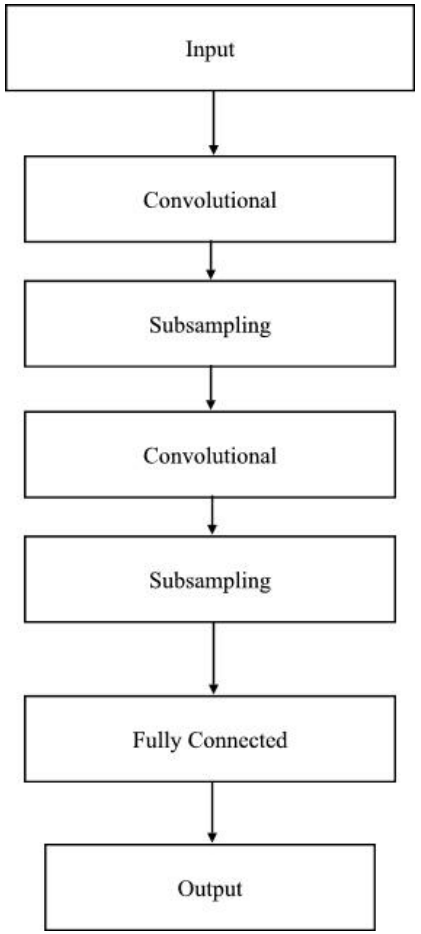


- Better: (for accuracy, but much slower) scored smudging/blurring (recognition and segmentation cannot be entirely decoupled – Yann LeCun).

Problems:

Classification

- LeNet-5 base Architecture

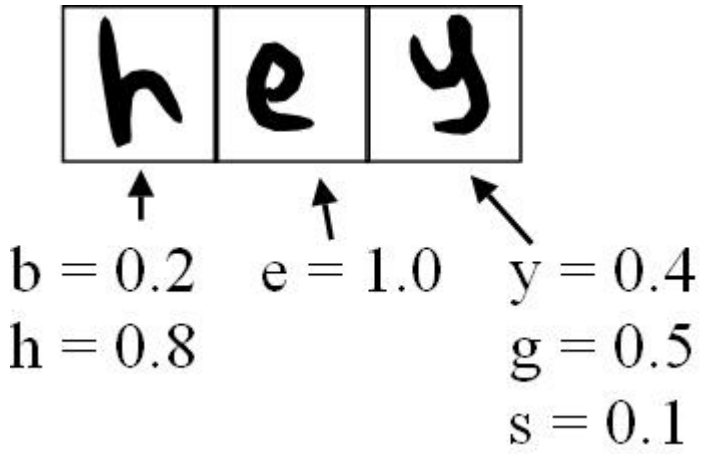


-DL4J (DeepLearning4Java) artificial neural network library for Java.

Algorithm:

Selecting Hypothesis words

- Given marginal distributions over the possible character classes in a fixed-length disconnected string of character images, how can we compute the *n best hypotheses* for the realisation of this word. (n most probable elements in the joint distribution).



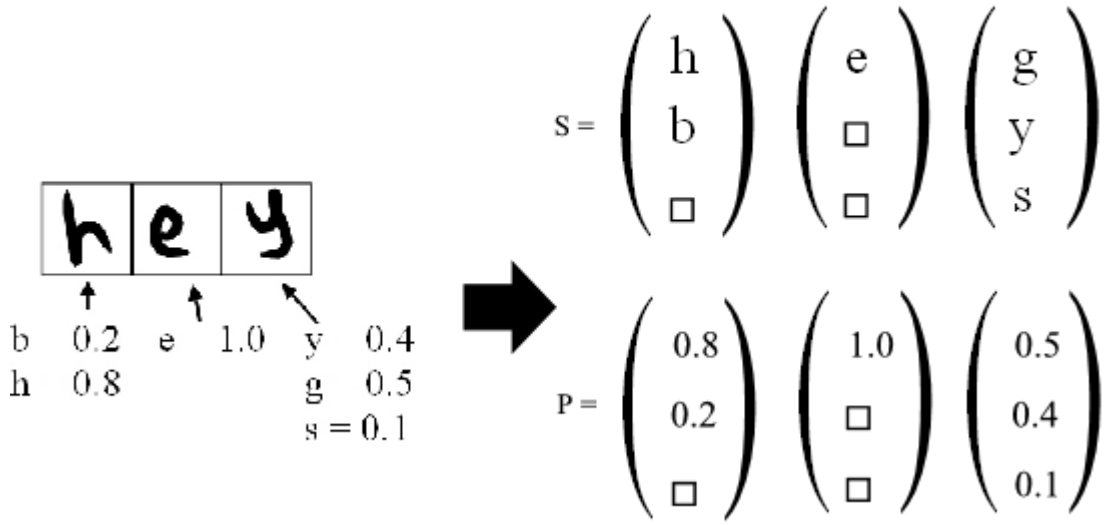
$P(\text{heg}) = 0.4$
$P(\text{hey}) = 0.32$
$P(\text{beg}) = 0.1$
$P(\text{bey}) = 0.08$
$P(\text{hes}) = 0.08$
$P(\text{bes}) = 0.02$

Easy to enumerate in small cases… but horrible if more character classes are involved/string is longer. 26^5 = 11.8 million

Algorithm:

Selecting Hypothesis words

Much better idea: order distributions and walk down the resulting matrix (P).



Algorithm:

Selecting Hypothesis words

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ p_{r1} & p_{r2} & \cdots & p_{rk} \end{bmatrix}$$

$$\text{where } p_{ij} \geq p_{xj} \text{ for } i \geq x$$

Algorithm:

Selecting Hypothesis words

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ p_{r1} & p_{r2} & \cdots & p_{rk} \end{bmatrix}$$

(1, 1, …, 1)

where $p_{ij} \geq p_{xj}$ for $i \geq x$

Algorithm:

Selecting Hypothesis words

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ p_{r1} & p_{r2} & \cdots & p_{rk} \end{bmatrix}$$

(r, r, ..., r)

where $p_{ij} \geq p_{xj}$ for $i \geq x$

Algorithm:

Selecting Hypothesis words

Example:

$$S = \begin{bmatrix} l & a & n & g \\ i & u & m & q \\ r & e & h & y \\ c & o & k & j \end{bmatrix}$$

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}$$

Guarantee: any path through the matrix is in order of decreasing probability.

Algorithm:

Selecting Hypothesis words

- Problem: multiple paths through matrix – no ‘path ordering’ (multiple choices for next string in path).

- Simple idea: keep track of best strings so far, and queue of best possible moves from each string – polling the queue gives us the next most likely string, and updating the queue is inexpensive. “This is effective as it allows us to store our answers and progress through the matrix as the same object in the same datastruture.”

- Need a matrix to tell us the change in probability from one string to the next string in a path. (I called this the **morph matrix** in development).

$$M = \begin{bmatrix} \frac{p_{21}}{p_{11}} & \frac{p_{22}}{p_{12}} & \cdots & \frac{p_{2k}}{p_{1k}} \\ \frac{p_{31}}{p_{21}} & \frac{p_{32}}{p_{22}} & \cdots & \frac{p_{3k}}{p_{2k}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{p_{r1}}{p_{(r-1)1}} & \frac{p_{r2}}{p_{(r-1)2}} & \cdots & \frac{p_{rk}}{p_{(r-1)k}} \end{bmatrix}$$

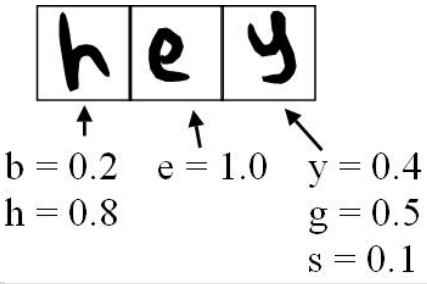
M_{ij} is the factor by which probability changes when we increment character j from position i to i + 1.

Algorithm:

Selecting Hypothesis words (MPS Algorithm)

1. Define a list `nMostLikely` where we will store (the column positions corresponding to) our strings, and a list `queue` where we will build a queue of best possible string modifications, ordered by the probabilities of the resulting string.
2. add the column positions corresponding to the most likely string to the list of `nMostLikely`. i.e. add $(1, 1, ..., 1)$ to `nMostLikely`. Calculate $p_0 = p_{11} \times p_{12} ... \times p_{1k}$ and add each of the tuples $(0, 1, m_{11} * p_0), (0, 2, m_{12} * p_0), ..., (0, k, m_{1k} * p_0)$ to `queue`.
3. Obtain and remove the first tuple in `queue`, `(stringPos, charPos, p)`.
4. Get the string in position `stringPos` of the list `nMostLikely`, i.e. let $(a_1, a_2, ..., a_k)$ = `nMostLikely[stringPos]`.
5. Increment the `charPos` character (indexed from 1) of this string, so that we have the string `new` $= (a_1, a_2, ..., a_{\texttt{charPos - 1}}, a_{\texttt{charPos}} + 1, a_{\texttt{charPos + 1}}, ..., a_k)$.

   This is the vector of column positions corresponding to the next most likely realisation of the string. (which has probability p).
6. If `new` is already in `nMostLikely` then go to 3. Otherwise add `new` to `nMostLikely`.
7. (For clarity) rename the column positions of `new` as $(u_1, u_2, ...u_k)$.
8. Add the following tuples to `queue`: $(\texttt{nMostLikely.size} - 1, 1, m_{u_1 1}) \times p), (\texttt{nMostLikely.size} - 1, 2, m_{u_2 2}) \times p), ..., (\texttt{nMostLikely.size} - 1, k, m_{u_k k}) \times p)$.
9. If `nMostLikely.size` is n then terminate, otherwise go to 3.



$P(\texttt{heg}) = 0.4$
$P(\texttt{hey}) = 0.32$
$P(\texttt{beg}) = 0.1$
$P(\texttt{bey}) = 0.08$
$P(\texttt{hes}) = 0.08$
$P(\texttt{bes}) = 0.02$

First, we compute P and M.

$$P = \begin{bmatrix} 0.8 & 1.0 & 0.5 \\ 0.2 & * & 0.4 \\ * & * & 0.1 \end{bmatrix}$$

$$M = \begin{bmatrix} 0.25 & * & 0.8 \\ * & * & 0.25 \\ * & * & * \end{bmatrix}$$

Now, applying the algorithm:

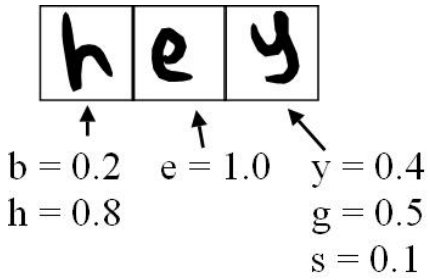
```
            nMostLikely = [(1, 1, 1)]
                     p0 = 0.4
                  queue = [(0, 3, 0.32), (0, 1, 0.1)]

(stringPos, charPos, p) = (0, 3, 0.32)
          (a1, a2, a3) = (1, 1, 1)
                   new = (1, 1, 2)
           nMostLikely = [(1, 1, 1), (1, 1, 2)]
                 queue = [(0, 1, 0.1), (1, 1, 0.08), (1, 3, 0.08)]

(stringPos, charPos, p) = (0, 1, 0.1)
          (a1, a2, a3) = (1, 1, 1)
                   new = (2, 1, 1)
           nMostLikely = [(1, 1, 1), (1, 1, 2), (2, 1 , 1)]
```

Algorithm:

Selecting Hypothesis words (MPS Algorithm)



First, we compute P and M.

$$P = \begin{bmatrix} 0.8 & 1.0 & 0.5 \\ 0.2 & * & 0.4 \\ * & * & 0.1 \end{bmatrix} \quad S = \begin{bmatrix} \texttt{h} & \texttt{e} & \texttt{g} \\ \texttt{b} & * & \texttt{y} \\ * & * & \texttt{s} \end{bmatrix}$$

$$M = \begin{bmatrix} 0.25 & * & 0.8 \\ * & * & 0.25 \\ * & * & * \end{bmatrix}$$

Now, applying the algorithm:

```
            nMostLikely = [(1, 1, 1)]
                     p0 = 0.4
                  queue = [(0, 3, 0.32), (0, 1, 0.1)]

(stringPos, charPos, p) = (0, 3, 0.32)
           (a1, a2, a3) = (1, 1, 1)
                    new = (1, 1, 2)
            nMostLikely = [(1, 1, 1), (1, 1, 2)]
                  queue = [(0, 1, 0.1), (1, 1, 0.08), (1, 3, 0.08)]

(stringPos, charPos, p) = (0, 1, 0.1)
           (a1, a2, a3) = (1, 1, 1)
                    new = (2, 1, 1)
            nMostLikely = [(1, 1, 1), (1, 1, 2), (2, 1 , 1)]
```

$P(\texttt{heg}) = 0.4$
$P(\texttt{hey}) = 0.32$
$P(\texttt{beg}) = 0.1$
$P(\texttt{bey}) = 0.08$
$P(\texttt{hes}) = 0.08$
$P(\texttt{bes}) = 0.02$

Algorithm:

Hypothesis scoring

- Often the case that the correct word does not lie within the first *n most likely* elements of the joint distribution

- Appropriate use of Levenshtein distance on subset of hypotheses to generate list of real words that are close – words are scored additively based on the underlying distribution

- Additive scoring prevents total mistakes from obliterating the recogniser's chances. i.e. An assignment of 0 by the approximated function where the target function would assign 1.

Algorithm

Results

- Best result: improved word accuracy of our 3 best otherwise naïve CNN classifiers **by an average of 35%!** (Accounting for all other variables – segmentation, layout analysis, etc.).

- It is possible that the method can *reduce* the accuracy of a classifier if the classifier was absolutely horrible at its job to begin with. Better classifier -> More effective use of this method.

Method displays high variance – either works really well or not very well at all. Again, likely due to underlying classifier being ‘better’ at one person’s handwriting than another’s.

Future Work

- Really what we want is a CNN-based recogniser upon which we can apply this method to the *unconstrained* offline handwriting recognition problem.

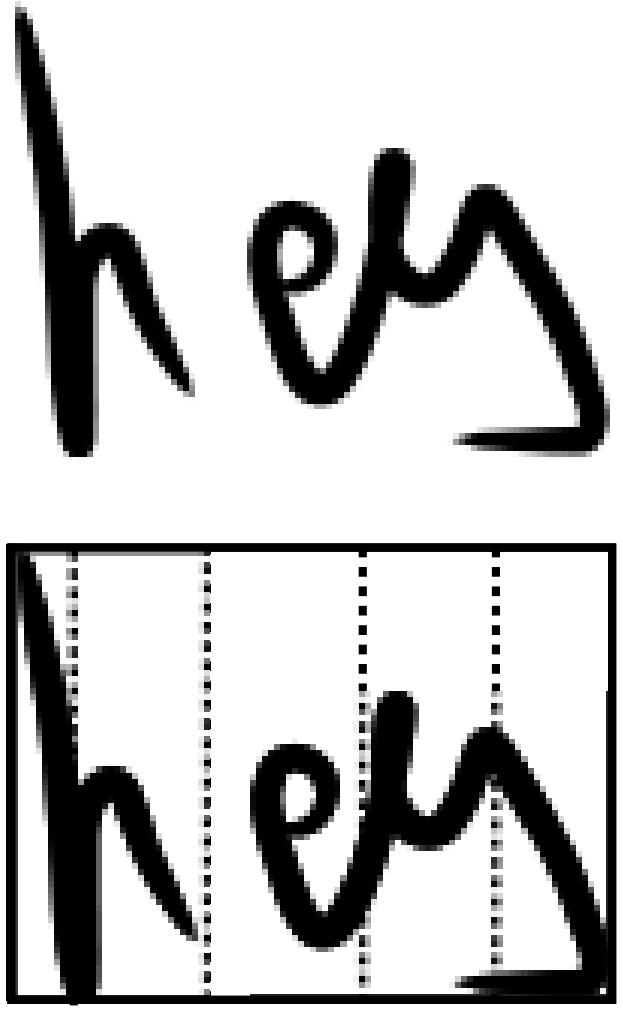
- This would involve generating and scoring hypotheses for segmentation and classification!

- Luckily, the algorithm we have presented is easily extendible to incorporate this once we have generated a set of hypothesis segmentations.

Algorithm

Heuristic Over-Segmentation

- Aim to definitely capture the true segmentation by oversegmenting.

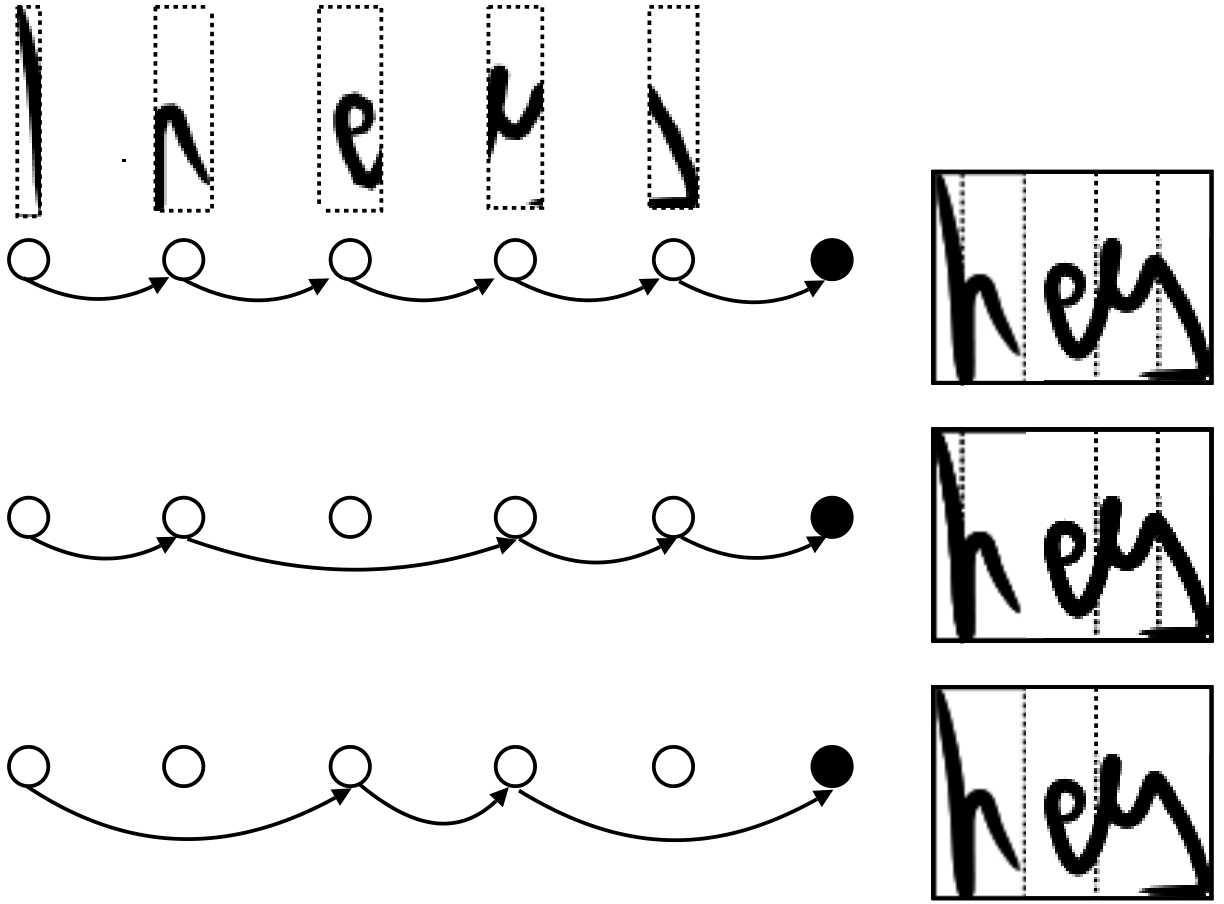


Future Work

Algorithm

Heuristic Over-Segmentation

- Build Directed Acyclic Graphs to place constraints on allowed segmentations.



Future Work

Algorithm

Heuristic Over-Segmentation

- For a given segmentation, we can get the fuzzy classification (character distribution) of each segment image. Then, we are in exactly the same situation as when we applied our algorithm before.

- So our hypotheses now need to take the form (segmentation, word) rather than simply considering a hypothesised word.

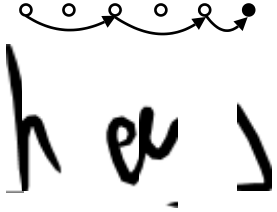
- Modifying the MPS algorithm is very easy – as well as keeping track of the column positions of each next most likely answer, also record the corresponding segmentation for this answer. (Also, for simplicity, pre-multiply the P matrix by the probability score of its corresponding segmentation).

Future Work

Algorithm

Heuristic Over-Segmentation

Future Work



$$S = \begin{bmatrix} h & e & g \\ b & \times & y \\ \times & \times & \times \end{bmatrix} \qquad S = \begin{bmatrix} h & a & j \\ b & \times & l \\ \times & \times & r \end{bmatrix}$$

$$P = \begin{bmatrix} 0.9 & 1.0 & 0.6 \\ 0.1 & \times & 0.4 \\ \times & \times & \times \end{bmatrix} \qquad P = \begin{bmatrix} 0.9 & 1.0 & 0.5 \\ 0.1 & \times & 0.3 \\ \times & \times & 0.2 \end{bmatrix}$$

- The MPS algorithm simply needs to compute the M P and S matrices for the best candidate segmentations, and then apply itself as normal, recording the segmentation to which a string belongs when building its queue of next best moves.

- In fact, any dependent random variable can be handled like this, by adding another branch to the paths… could find use to have higher order hypotheses.

References

Heuristic Over-Segmentation and **Convolutional Neural Networks**:

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner. "GradientBased Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no.11, November, 1998.

Hidden Markov Models:

Thomas Plötz, Gernot A. Fink. "Markov models for offline handwriting recognition: a survey," International Journal on Document Analysis and Recognition, vol. 12, no. 4, December 2009.

Adaptive Thresholding using the Integral Image:

Derek Bradley, Gerhard Roth. "Adaptive Thresholding Using the Integral Image," Journal of Graphics Tools, vol. 12, no. 2, pp. 13-21, 2007.