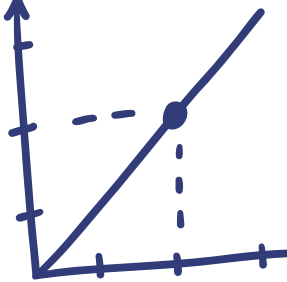
Rendering Handwritten Equations

Martijn de Vries

Capstone Project Presentation
June 12th, 2023



Problem Statement

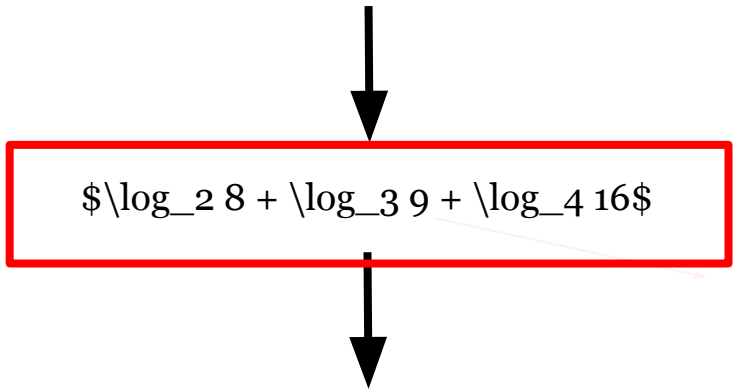
Develop a tool that can render handwritten equations in digital format.

Input: an image containing an equation

Output: the equation in digital format, using LaTeX math notation

For example..

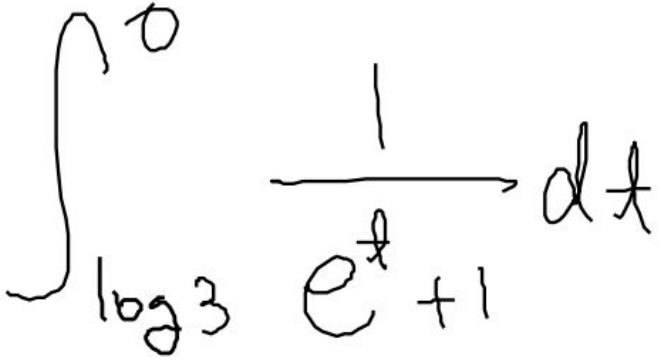
$\log_2 8 + \log_3 9 + \log_4 16$



$$\log_2 8 + \log_3 9 + \log_4 16$$

Why is this complicated?

Symbols are not just read left-to-right

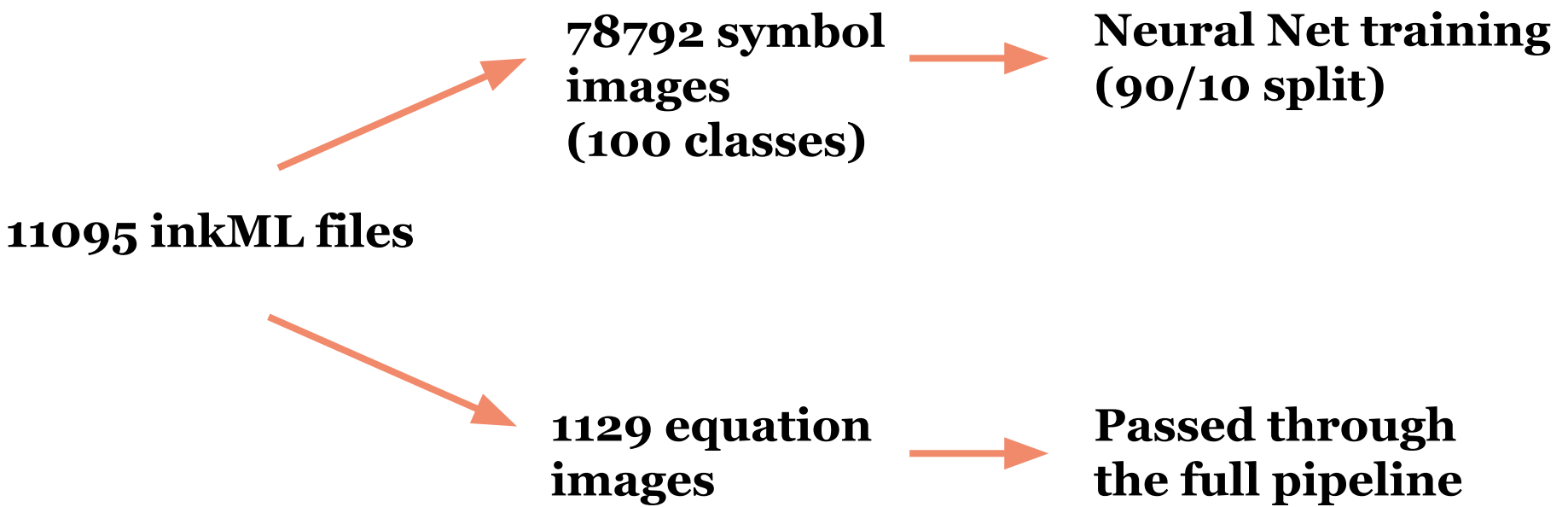


Data

- **CROHME**
- **‘inkML’ format**
- **Full equation label**
- **Labels for each symbol**
- **100 different symbols**

https://www.isical.ac.in/~crohme/CROHME_data.html

Data Processing



https://www.isical.ac.in/~crohme/CROHME_data.html

Pipeline Overview

1. Image tresholding

Change image into black/white values

2. Resolving Symbols

Figure out the symbol on the image and their order

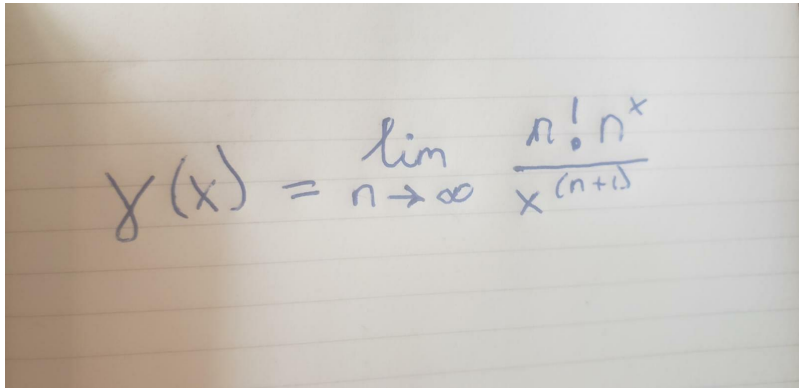
3. Model Prediction

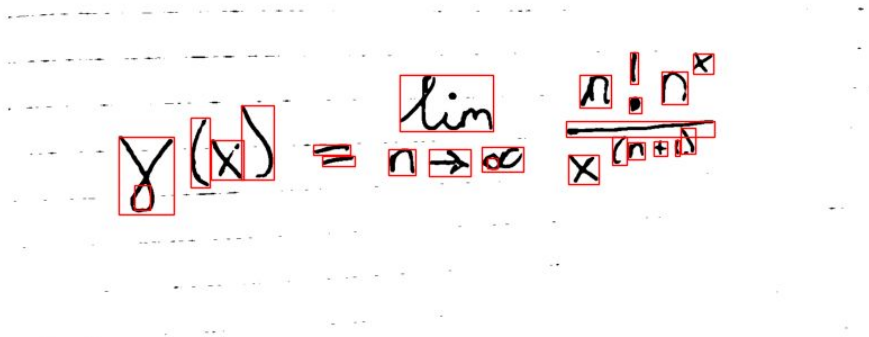
Use an EfficientNetB0 model to predict labels

4. Rendering the equation

Stitch the predictions together into an equation

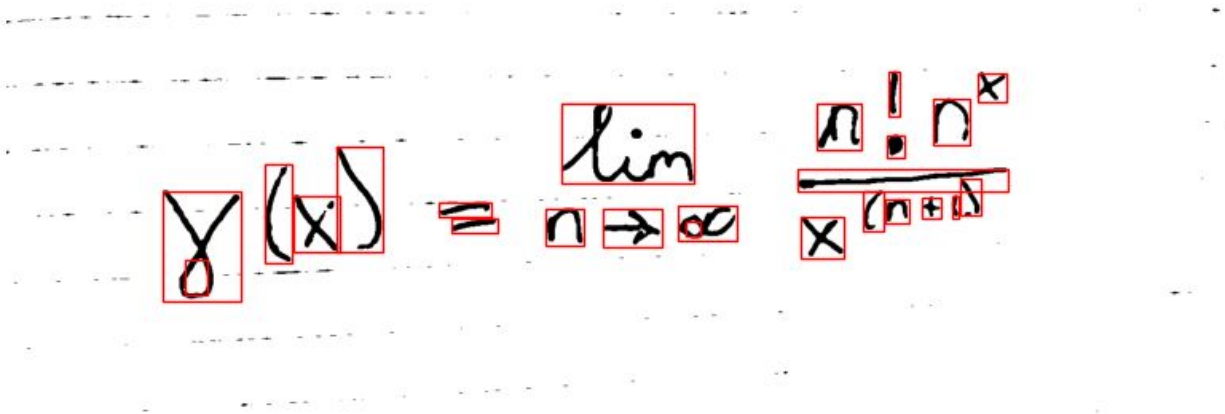
1) Image Tresholding





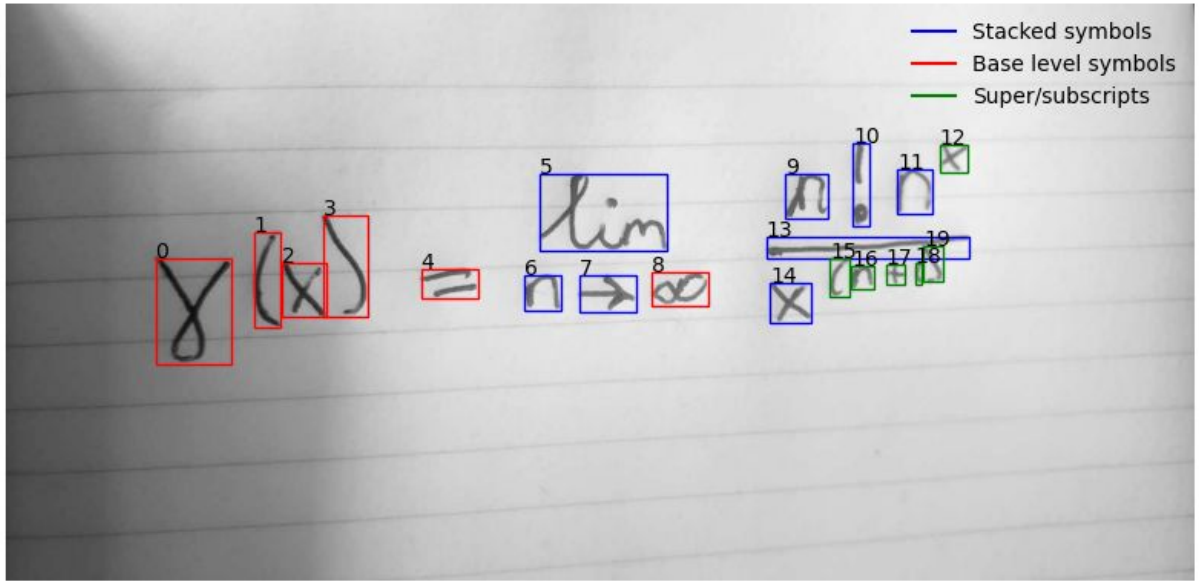
Adaptive Gaussian Tresholding

2) Resolving Symbols



Find Contours and Bounding Boxes

2) Resolving Symbols



i) Remove Inner Contours
ii) Merge Boxes

iii) Determine Symbol order
iv) Determine Script level

3) Model Predictions

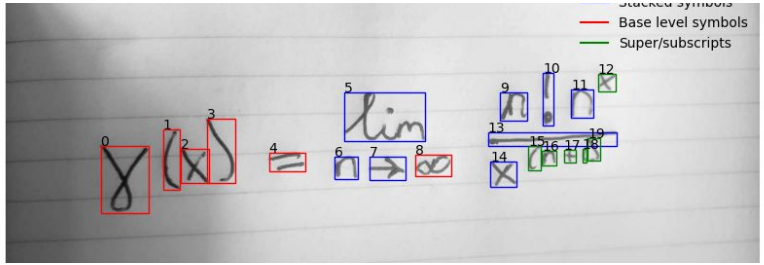
Model: EfficientNetB0 CNN

95% accuracy on validation data
(*individual symbol images)*

Prediction Step:

Symbol Images ->
List of class labels

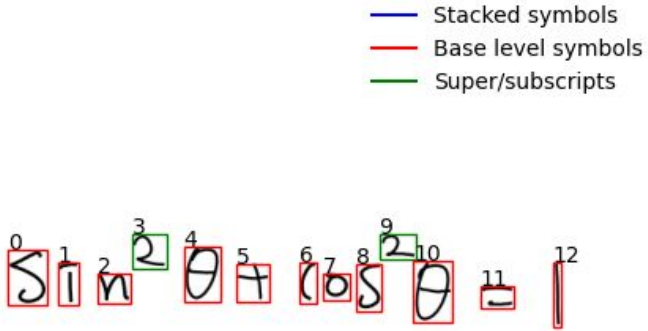
4) Render the Equation



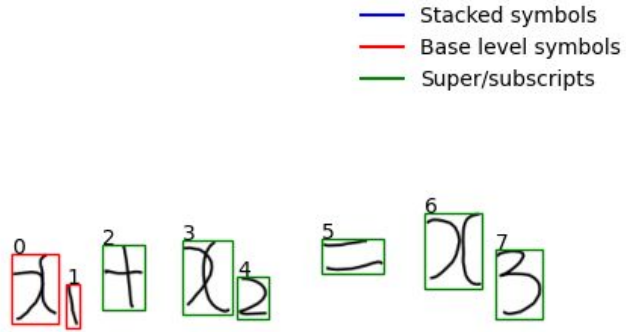
$\\gamma(X)=\\lim_{n-}\\infty\\frac{n!n^{k}}{\\times^{(n+1\\prime}}$

$$\gamma(X) = \lim_{n-} \infty \frac{n!n^k}{\times^{(n+1\prime}}$$

Two more examples



Prediction: $\sin^2\theta + \cos^2\theta = 1$



Prediction: $x1^{+x2=x3}$

Overall Pipeline Performance

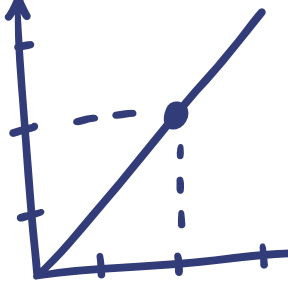
Damerau-Levenshtein distance

Eg:
‘fish’<-> ‘ifsh’

Edit distance = 1
normalized distance = ¼ = 0.25

DL distance	% of equations
0	**15.7%**
< 0.10	**20.9%**
< 0.50	**65.9%**

Final thoughts..



This is a difficult problem to solve!

Errors compound through the 4 pipeline steps

Different linewidths seem to make the predictions less accurate than the 95% in NN

Multiple Object Detection algorithms