CS-519-M01 Applied Machine Learning
Project Final Report
Theoderic Platt & Long Tran

Report

Motivation

Translation of mathematical formulae from PDF can be challenging due to the variety of different formatting issues present. Some PDF's contain formulae as images while others use plain-text, which can be challenging to parse by hand.
The motivations of this project are to use machine learning models of characters, numbers, and operators to accurately identify the text present in an image taken of a pdf. The code focus of this project will be to be able to parse images of piecewise functions, and accurately format them as output. We will also have a focus on learning how data is generated, so no external datasets will be used.
This problem has already been solved by tools like MathPix (see References 1 below), but we still would like to attempt to solve the issue of parsing images with machine learning models to output the contents of a formula.

Problem Definition

This project focuses on reading in an image file containing mathematical formulation and outputting the formula present inside the image. The required segments of this are image processing, symbol identification using learned models, and processing of the output for correct formatting.
Because the focus of this project is on Machine Learning, we will be limiting our alphabet to the following scope:

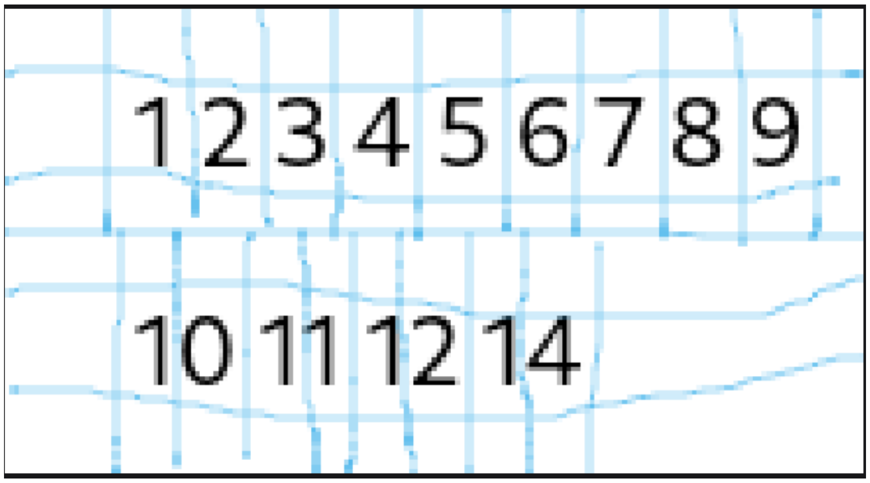
- Characters: a-z and A-Z.
- Numbers: 0-9
- Operators: (,) , + , - , = , "," , ÷ , × , {

Solution Description

For the problem of parsing an image containing a formula, several steps are required. First, the image needs to be taken in as input and the individual symbols found within this image need to be segmented into separate pieces. Second, these image segments containing individual symbols need to have a classifier run against them to determine if they are part of the Characters set, Numbers set, or Operators set. Once it has been determined which dataset the symbol belongs to, a corresponding model for this symbol needs to be used to predict the symbol. Finally, if the symbol is specifically the Operator '{', this tells us that we need to check if this symbol is a part of a piecewise function. In this event, we need to find what subsequent symbols are a part of this same function and what height layers they occupy. Finally the results of this code need to be outputted. The subsequent paragraphs will go into more detail on how each of these tasks are accomplished in this project.

For image segmentation, the general method used was through checking vertical and horizontal histograms:

1) We sum the grayscale image pixel values in each row and in each column. Then, for each row/column, we divide the sum by 255 (the value for white pixel), then we subtract this number from the number of pixels in that row/column. Thus, if the calculated value for a row/column is 0, then that row/column is an empty row/column; else, that row/column contains some non-white (non 255) pixels.
2) We first get a list of rows that are empty and segment the image by row. We segment the image by cropping out the sections that are in between the empty rows
3) Then, for each row sub-image, we segmentize by both row and column. An example of a segmentation can be seen below:



Picture 1: An image of segmentation

4) Then, one segmentation alone may not lead to a single symbol in the image. Thus, our segmentation will segment each sub-image recursively until the segmented image is the same as the original image.
5) Thus, the data structure of the return value is a tuple containing the original image, and a matrix of sub-images in which each cell/element in the matrix is a similar tuple for the sub-image ***(for the sub-image that have only white pixels, the value in the matrix of the original image is None and not a tuple)***. The positions of each element in the matrix corresponds to the position of the sub-image in the segmented image. For example, looking at Picture 1, the sub-image containing 8 will be in the first row image; that is, the image above the straightest horizontal blue line. In the matrix of that row image, the position of the sub-image will be (1, 8).

This approach often will segment each symbol correctly, but if 2 separated characters are connected to each other through gray pixels (neither 0 nor 255), then the 2 characters can never be segmented.
Another drawback of this method is that the spacing is not always correct, but we think it can be fixed quickly by the user and the speed benefits gained from the simplicity of the method can help decrease runtime.

The conversion process is also done recursively on the returned recursive data of each row image:

1) In the tuple of each row image, we recursively go down to the leaf, predict the leaf, and return the predicted value upward.
2) Then at the root recursion level of each tuple, we will add a space when we see None and add a newline when we go to a new sub-row.
3) At the end of each row image, we will also add a newline.
4) There are special cases in our conversion due to segmentation in which we have to check for the size of the matrix and the predicted values of the sub-images to determine if a parent image is the following symbol:
    a) Division.
    b) Letter i.
    c) Equality.

Once image segmentation is complete, the learned models need to be used. These models are all trained on the datasets generated as part of this project. These datasets were all generated by selecting fonts and symbols, and using python sympy's preview function to save images of LaTeX. Further below in this paper will be a section dedicated to describing these datasets in more detail.
There are four models used for this program: Characters, Numbers, Operators, and Inter-Class.

- The Characters model is trained on the Characters dataset, containing classifiers for all characters a-z and upper-case characters A-Z. This model uses Logistic Regression with the solver set at ‘lbfgs’, the max iterations set at 10000, and the C set at 10. The multi-classification method used is One vs Rest.
- The Numbers model is trained on the Numbers dataset, containing classifiers for the digits 0-9. This model uses Logistic Regression with the solver set at ‘lbfgs’, the max iterations set at 10000, and the C set at 10. The multi-classification method used is One vs Rest.
- The Operators model is trained on the Operators dataset, containing classifiers for the following set of symbols: [*(,) , + , - , = , “,” , ÷ , × , {*] . This model uses Logistic Regression with the solver set at ‘lbfgs’, the max iterations set at 10000, and the C set at 10. The multi-classification method used is One vs Rest.
- The Inter-Class model is trained on all datasets, and serves the goal of identifying which dataset a given symbol belongs to so that its specific model can then be applied to it. The three classes outputted by the Inter-Class model are [1,2,3]. The class 1 represents that the instance is part of the Numbers dataset. The class 2 represents that the instance is part of the Operators dataset. The class 3 represents that the instance is part of the Characters dataset. This model was trained on a Convolutional Neural Network over 50 epochs, using a batch size of 64. Figure 1 below shows the architecture designed for this CNN model.

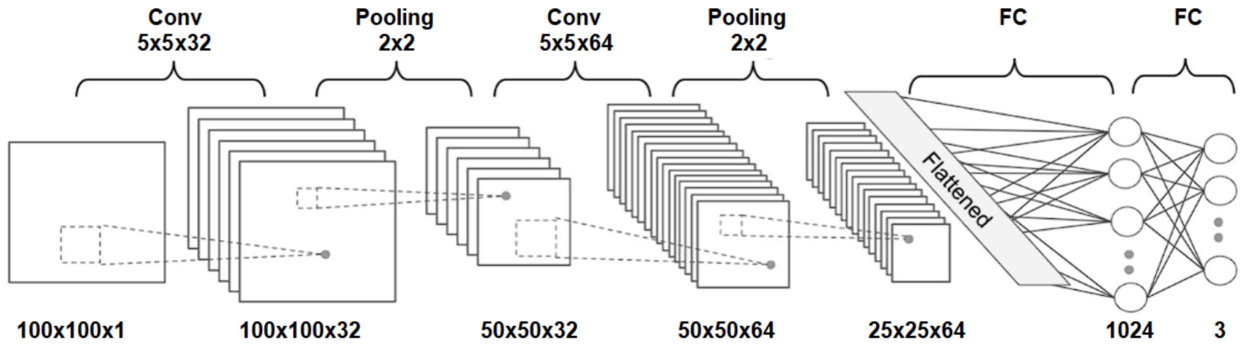


Figure 1: Inter-Class CNN Architecture Diagram.

For the Characters, Numbers, and Operators models, the classifier model is the last step in the pipeline:

1) HogTransformer: Histogram of Oriented Gradients. The general idea is that we group 8 by 8 pixels into a calculated magnitude. (See Reference 3 for more details)
2) StandardScaler().
3) PCA(n_components=0.9).
4) Our classification model of choice.

For cases where a piecewise function is detected, the { symbol will be segmented first. When we detect a single { in the sub-image matrix, we would get the sub-image to the right of { out of the parent image, and convert that using the steps discussed above. Thus, we will get the lines of the piecewise functions converted properly.

Datasets

Three datasets were used in this project, these are the Characters Dataset, the Numbers Dataset, and the Operators Dataset. All three datasets were generated locally by taking a set of the elements we wanted, and parsing them through a LaTeX library which allowed us to reformat the symbols into different fonts. The datasets are stored as a .csv file containing two columns. The first column for each instance contains a relative path from the .csv to a .png image file containing an image of the symbol. The second column for each instance is the classifier, containing the symbol contained within the corresponding image.

Characters Dataset
The characters dataset consists of 100 instances per class. The classes contained in this dataset are: [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]. The images are each a 100x100 pixel .png.

The image will be converted into grayscale during training and predicting. The features consist of the grayscale 100x100 pixels.

Numbers Dataset
The numbers dataset consists of 100 instances per class. The classes contained within this dataset are: [0,1,2,3,4,5,6,7,8,9]. The images are each a 100x100 pixel .png.

The image will be converted into grayscale during training and predicting. The features consist of the grayscale 100x100 pixels.

Operators Dataset
The operators dataset consists of 100 instances per class. The classes contained within this dataset are [(,),+,-,=,‘,’,divide,times,curly_bracket]. Due to some dependencies, we were unable to directly store the symbols for division, multiplication, and curly brackets as their classifiers and needed to substitute these symbols with their names. The divide classifier corresponds to the unicode symbol ‘÷’, commonly accepted as the division symbol. The times classifier corresponds to the unicode symbol ‘×’, commonly accepted as the multiplication symbol. The classifier curly_bracket corresponds to the symbol ‘{’. The images for this dataset are all 100x100 pixel .pngs.

The image will be converted into grayscale during training and predicting. The features consist of the grayscale 100x100 pixels.

Analysis

All testing and analysis were run remotely on the linux machine Kahn located in the Science Hall. Performance analysis pertaining to training times are dependent on this machine. The machine’s specifications can be found below after this Analysis section.
When training the models there was an 80:20 train:test split applied. Different parameters and classifiers were experimented upon including Perceptron, Decision Trees, Logistic Regressions, and CNNs. The metrics shown in this analysis section cover the classifiers that were selected and detailed in the *Solution Definition* section of this paper. The accuracy for training and testing are measured, as well is the time taken in seconds to train each model. Table 1 below shows these metrics.

	Training Accuracy	**Testing Accuracy**	**Training Time (seconds)**
Characters Model	100.0%	99.5%	1.1865
Numbers Model	99.783%	97.994%	9.6201
Operators Model	100.0%	99.444%	1.0680
Inter-Class Model	99.436%	96.795%	280.6467

Table 1: Model Training Analysis Metrics

Based on the small disparity between training and testing accuracies, we have determined that the models are likely not overfitting. Because the models show high accuracy, we have determined that the models are not underfitting.

Finally, the training times are expected as the Characters and Operators models have significantly fewer classes to train on than the Numbers model. As is outlined above, the Inter-Class model uses a CNN while the others do not, which explains the high training time for this model.

To analyze the models and datasets we have trained, several test cases were designed. Each of these test cases consists of an image containing a series of characters, numbers, operators, or a combination of the three sets. The evaluation metric used is accuracy, which is measured in three places:

1) (Metric 1) Accuracy is taken to determine how well the Inter-Class model can determine what class a symbol belongs to.
2) (Metric 2) Accuracy of the Characters, Numbers, and Operators models is taken together, only on the set of test instances which were correctly classified by the Inter-Class model. This excludes misclassified instances by the Inter-Class model, and provides a more meaningful metric for analysis of the three single classifiers.
3) (Metric 3) The overall accuracy is taken including both steps one and two. This metric demonstrates how well the entire pipeline of model predictions is able to work to accurately produce results.

In addition to accuracy metrics, the running time in milliseconds to predict the models is measured. This time represents the total time to run all models and predict the test case, running times will not be measured for intermediary steps.

	Accuracy (metric 1)	Accuracy (metric 2)	Accuracy (metric 3)	Running Time (Seconds)
Test 1	75.0%	66.666%	50.0%	0.5131
Test 2	82.352%	100.0%	82.352%	0.5703
Test 3	76.923%	100.0%	76.923%	0.5652
Test 4	90.0%	100.0%	90.0%	0.6117

Table 2: Analysis Metrics of Test Cases

Based on the results of Table 2, it can be seen that the accuracy of the Characters, Numbers, and Operators classes are on average very high. Test 1 shows the lowest accuracy by metric 2, which is likely due to a known issue we encountered during this project, the equals symbol is commonly misclassified as two minus symbols (thus, it is a special case for conversion as mentioned above in Solution Explanation).
Test 1 contains the most equals symbols out of the test cases used. The accuracy by metric 1 demonstrates that the Inter-Class model has an adequate accuracy, ranging from 75-90%. By metric 3 it can be seen that the Intra-Class classifier in most cases holds back the accuracy of the overall prediction. This metric is decreased greatly whenever either metric 1 or metric 2 are low, as can be observed in Test 1's Accuracy metric 3 score.
Evaluation of all test cases show a high running time, as all test cases completed in under a minute and the variation between the smallest test case, Test 1, and the largest test case, Test 4, is not high.

Test Machine

The test machine specifications:

+ CPU: Intel(R) Core(™) i9-9900K CPU @ 3.60GHz
+ RAM: 64 GiB.
+ OS: OpenSUSE
+ GPU: NVIDIA Quadro P2000
+ Python Version: 3.10.10

References & Codebase

1. MathPix. Retrieved from https://mathpix.com/
2. Reddy S. (2019). *Segmentation in OCR !!*. Retrieved from https://towardsdatascience.com/segmentation-in-ocr-10de176cf373
3. Steegstra P. (2018). Retrieved from https://kapernikov.com/tutorial-image-classification-with-scikit-learn/
4. **GitHub Project:** https://github.com/Theo-Platt/CS519_Group_Project