Query-Driven Global Graph Attention Model for Visual Parsing: Recognizing Handwritten and Typeset Math Formulas

by

Mahshad Mahdavi

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Imaging Science

Chester F. Carlson Center for Imaging Science
Rochester Institute of Technology

August 7, 2020

Signature of the Author _____

Certified by _____
PhD Program Director Date

Ph.D. IN IMAGING SCIENCE
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

Ph.D. DEGREE DISSERTATION

The Ph.D. degree dissertation of Mahshad Mahdavi
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Computing and Information Sciences

Dr. Richard Zanibbi, Dissertation Advisor

Dr. Daniel Phillips, External Chair

Dr. Carl Salvaggio

Dr. Harold Mouchère

Date

Contents

List of Figures

List of Tables

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Zanibbi for the continuous support of my PhD study and related research, and for his motivation, and immense knowledge. I would like to thank the rest of my thesis committee: Dr. Harold Mouchère, Dr. Carl Salvaggio, and Dr. Daniel Phillips for their insightful comments and questions to widen my research from various perspectives.

I would like to extend my gratitude to all the CROHME [68] organizers including Dr. Christian Viard-Gaudin and Dr. Utpal Garain for providing me an opportunity to lead CROHME2019 competition. I would like to thank Dr. Mouchère in particular for his guidance during the competition.

I thank my fellow classmates in CIS and labmates in DPRL for the stimulating discussions and for all the fun we have had in the last five years. I thank Michael Condon and Leilei Sun for their insight and contribution to this work.

I would like to thank my family and friends: my parents and my brother who set me off on the road to this PhD a long time ago, for supporting me throughout writing this thesis and getting my degree, Ali Rouzbeh for his encouragement, patience and motivation especially during this pandemic.

This material is based upon work supported by the National Science Foundation (USA) and the Alfred P. Sloan Foundation.

Statement of Originality

This thesis and the research to which it refers are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with standard referencing practices. I gratefully acknowledge the helpful guidance and support of my supervisor, Dr. Zanibbi. The main ideas and approach in this thesis were significantly refined through his comments. A portion of Chapter 4 was written in collaboration with my advisor Dr. Zanibbi, Michael Condon and Kenny Davila who contributed in LPGA_{RF} system which appeared before in [65]. A portion of Chapter 5 was written in collaboration with my adviser Dr. Zanibbi and Leilei Sun, who contributed in introducing binary masks in QD-GGA. This study also appeared before in [67]. Finally, a large portion of Chapter 3 appeared in [66,68] was written in collaboration with my advisor Dr. Zanibbi, Dr. Harold Mouchère, Dr. Christian Viard-Gaudin and Dr. Utpal Garain.

For my parents Maryam and Hamid.

Query-Driven Global Graph Attention Model for Visual Parsing: Recognizing Handwritten and Typeset Math Formulas

by

Mahshad Mahdavi

Submitted to the
Chester F. Carlson Center for Imaging Science Ph.D. Program in Imaging Science
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

We present a new visual parsing method based on standard Convolutional Neural Networks (CNNs) for handwritten and typeset mathematical formulas. The Query-Driven Global Graph Attention (QD-GGA) parser employs multi-task learning, using a single feature representation for locating, classifying, and relating symbols. QD-GGA parses formulas by first constructing a Line-Of-Sight (LOS) graph over the input primitives (e.g handwritten strokes or connected components in images). Second, class distributions for LOS nodes and edges are obtained using query-specific feature filters (i.e., attention) in a single feed-forward pass. This allows end-to-end structure learning using a joint loss over primitive node and edge class distributions. Finally, a Maximum Spanning Tree (MST) is extracted from the weighted graph using Edmonds' Arborescence Algorithm. The model may be run recurrently over the input graph, updating attention to focus on symbols detected in the previous iteration. QD-GGA does not require additional grammar rules and the language model is learned from the sets of symbols/relationships and the statistics over them in the training set.

We benchmark our system against both handwritten and typeset state-of-the-art math recognition systems. Our preliminary results show that this is a promising new approach for visual parsing of math formulas. Using recurrent

execution, symbol detection is near perfect for both handwritten and typeset formulas: we obtain a symbol f-measure of over 99.4% for both the CROHME (handwritten) and INFTYMCCDB-2 (typeset formula image) datasets. Our method is also much faster in both training and execution than state-of-the-art RNN-based formula parsers. The unlabeled structure detection of QD-GGA is competitive with encoder-decoder models, but QD-GGA symbol and relationship classification is weaker. We believe this may be addressed through increased use of spatial features and global context.

Chapter 1

Introduction

Mathematical notation is an essential source of information in many fields and the ability to recognize them is an important module in OCR (Optical Character Recognition). Math recognition is vital for several research-based and commercial-based applications, such as educational aids, navigational tools, digital assistance, and any visual-language task that needs machines to understand mathematical notations. The potential benefits of easier manipulation, modification, searching and accessing formulas have drawn different research groups' and companies' attention around the world to both the automatic recognition and retrieval of mathematical notation [95].

Parsing essentially means finding the underlying structure of the data we are given. In this work, we are interested in parsing math formulas from handwritten strokes or images. Visual parsing of mathematical expressions converts input images to a representation of formula structure, which in our case is a hierarchical arrangement of symbols on writing lines. Figure 1.1 shows an example of the inputs and outputs of a parsed formula. Graphs have been heavily used for representing spatial relationships between a given set of components. In math parsing, these components can be strokes in handwritten equations or connected components in images. Each stroke consists a sequential list of points representing sampled (x,y) coordinates as each stroke is written. Parsing a graph involves identifying a sub-graph with minimal cost or maximum probability. For math recognition, the final subgraph is usually a Symbol Layout Tree (SLT), see Figure 1.1.

In Figure 1.1, we introduce primitive-level graph representations used to

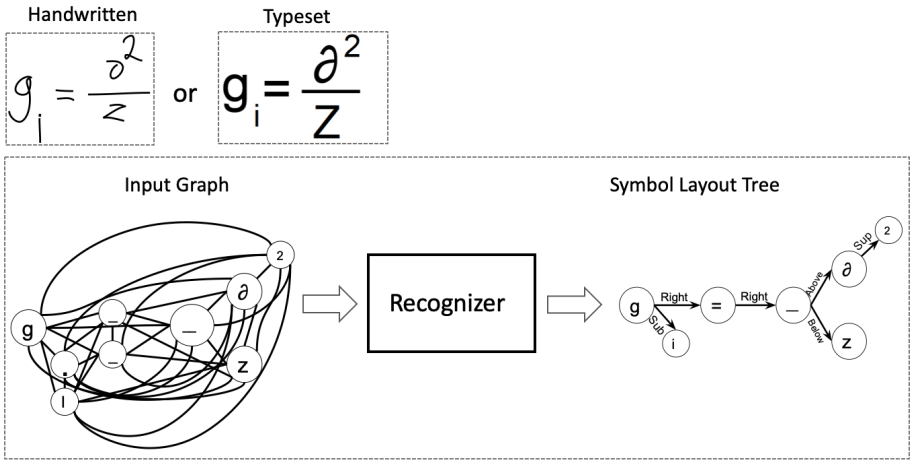


Figure 1.1: Visual parsing of mathematical expressions from handwritten input strokes or connected components in images. The input images are converted to a representation of formula structure.

parse formulae. Nodes represent individual strokes or connected components while edges represent possible relationships between primitives. These relations can identify strokes belonging to the same symbol, or spatial relationships between symbols such as Right (g and $=$) or Below ($fraction$ and z).

Ideally, we would like to reduce the size of the search space by allowing our input graph to have edges only between the primitives having a spatial relationship in the original setting (perfect precision) and avoid miscellaneous edges which add confusion to the problem. We use the Line-of-Sight (LOS) [19, 38] graphs to represent spatial relationships between the chosen set of visual primitives which can be handwritten strokes, symbols, lines, contours, or image segments. Figure 1.1 shows the edges exist in a LOS for $g_i = \frac{\delta^2}{z}$ marked with yellow.

We propose a CNN-based (Convolutional Neural Net) method to classify nodes and edges in the input LOS graph. After assigning the class probabilities to nodes (primitives) and edges (relations), the final tree is extracted using Edmond's algorithm which extracts the tree with the maximum score. We introduce a Query-Driven Global Graph Attention (QD-GGA) model for classifying nodes and edges efficiently and accurately.

1.1 Query-Driven Global Graph Attention (QD-GGA) parsing.

Our novel Query-Driven Global Graph Attention (QD-GGA) parser extends traditional CNN models designed for multi-dimensional data (e.g.,images) to graph-structured data. QD-GGA is comprised of the four modules: feature extraction, attention, classification, and Maximum Spanning Tree (MST) extraction.

Our attention module uses the input graph to query (filter) CNN features from a single feature representation of the input image to efficiently obtain inputs for multiple classification, segmentation, and relationship decisions. Therefore, all class distributions for nodes and edges in a Line-of-Sight (LOS) graph are predicted at each iteration.

In the following, a brief introduction is given for each of the main modules in QD-GGA. A more detailed description of QD-GGA architecture and our general approach is provided in Chapter 5.

Feature Extraction. We use Convolutional Neural Networks (CNNs) equipped with query-driven trainable attention masks to detect and classify symbols, and parse formula structure. Our two sets of inputs: the equation image and the attention masks are fed into different branches. The main branch takes the input image and generates a feature map by passing it through multiple convolutions and poolings. We use SE-ResNext [37, 91] architecture for feature extraction in the main branch.

Attention. Our attention module uses the input LOS graph to generate spatial attention masks and guide the model by element-wise multiplication of each of these node/edge masks with the final feature map. So, instead of exhaustively searching pixels in the input image to find salient areas as used in common sequence-to-sequence models [20, 100], the proposed mechanism dynamically brings the part of the image important for decision making to the forefront using the knowledge given from the predefined map based on edges and nodes of the input graph. This permits to generate query features for all the targets in the equation (e.g., strokes and their relations in handwritten equation) and be able to classify all targets in a single batch and count all the errors in the equation and modified it globally similar to integrated approaches.

Learning Mechanism. Feature vectors representing nodes and edges of an expression go into the task-specific classifiers to get predicted. We have

three classifiers to (1) segments primitives into symbols, (2) classify primitives, and (3) find the relation between those primitives. Inspired by Multi-Task Learning (MTL) frameworks, these classifiers are trained simultaneously by defining a joint loss for the model.

1.2 Thesis Statement

Fast and accurate recognition of typeset and handwritten mathematical formulas can be obtained by an end-to-end CNN-based model equipped with query-driven global graph attention.

1.2.1 Contributions

- Visual Parsing:
  - **Query-Driven Attention Model for Visual Parsing.** A novel attention model that queries CNN features from a single image to efficiently obtain inputs for multiple classification, segmentation, and relationship decisions by defining a graph on the input image.
  - **Joint Loss over Adjacency Matrices for Visual Parsing.** End-to-end structure learning directly from a joint loss computed over adjacency matrices holding class distributions for primitives (e.g., strokes, connected components) and primitive-pair class labels.
  - **Recursive Training to Update Query Features.** Introducing the recursive training which allows primitive-level input masks to use the segmenter information and convert into symbol level. These updated masks query the features again. Joint loss is computed using this new features.
  - **LPGA.** A novel structure is proposed to add contextual visual information with a Convolutional Neural Network that has two branches (target/context) for symbols and relationships classification using LOS graph-based attention [65].
  - **QD-GGA.** A faster and more accurate generalization of LPGA: (a) The use of a global rather than local attention model.(b) Features

are trained for multiple tasks concurrently within an end-to-end architecture.

- **Evaluation Metric.** To evaluate and compare visual structure recognition using symbols and relationships directly, we introduce SymLG (Symbolic LG) representation [66]. SymLG defines node identifiers by the sequence of spatial relationships from the root symbol to each symbol in the tree without requiring segmentation information.
- **Support for Multiple Input Types.** Our approach can be applied directly to handwritten data from a tablet as well as raster images (e.g., PNG).

1.3 Overview Of This Document

The remainder of this document is organized as follows. In Chapter 2 we present related work that inspired our approach, mentioning similarities and dissimilarities with our proposed method and the intuition behind taking these approach for tackling math recognition problem. This includes the traditional techniques in online and offline math recognition, Encoder-decoders, Visual parsing using CNNs, Multi-Task Learning, Attentions in CNNs and more. We introduce the datasets, metrics and tools we used for evaluating our model performance in Chapter 3. LPGA methodology and experiments are discussed in Chapter 4. In Chapter 5, we present our QD-GGA approach including model architecture, attention module, loss definition, training process and implementation details. Finally, we review the design experiments in QD-GGA and discuss the important observations in each experiment shaping our research and benchmark our methods against state-of-the-art systems for both handwritten and typeset formulas in Chapter 6.

Chapter 2

Related Work

In the following we provide an overview of approaches proposed for structure parsing with a focus on methods pertinent to formula recognition. Structure parsing has slightly different meanings in different branches of natural language processing, computer vision, pattern recognition, and machine learning, but the term essentially means finding the underlying structure of the data we are given. For instance, understanding a text by detecting characters, words and their relations in a sentence for text to speech applications or understanding a scene by classifying the objects and finding their relations is a visual structure parsing task. In this work, we are interested in parsing math formulas for which symbols are classified and their relations are identified.

We review the common approaches in math expression recognition in Section 2.2. Syntactic Methods (Constituency Parsing), Minimum Spanning Tree extraction (Dependency Parsing), and Encoder-Decoder Models are the most common approaches. Table 2.1 presents a summary of these models.

Section 2.4 reviews some of the general visual parsing studies using Convolutional Neural Networks (CNNs) as we believe math parsing can also be done with a non-recurrent standard CNN. In these studies, structure learning is done with a CNN equipped with an attention module, or in a Multi Task Learning (MTL) framework in which the attention module provides task-specific features from a single global feature pool, whilst allowing for features to be shared across different tasks. Another common approach in visual structure learning is graph-based CNNs (GCNN) which have the most similarity to our proposed approach since convolutional features are utilized in these

system for graph parsing. Table 2.2 presents a summary of these works.

In the remainder of this chapter, we first introduce different representations that systems use to output recognized formula in Section 2.1. Next, we review traditional math recognition methods, the state-of-the-art systems based on encoder-decoder architectures and discuss their limitations in Section 2.2. We review general strategies in visual structure learning, mostly based on CNNs, that inspired our approach (Section 2.4). Finally, a summary of this chapter is provided in Section 2.5.

2.1 Math Encodings

Recognizing math produces a formula representation from the input (e.g., raster images, strokes, or PDFs) that identifies symbols and their relationships. Trees are one of the data structures that can be used for representing the formulas. In fact, MathML and LaTeX are actually trees with additional formatting commands. Therefore, an expression can be presented by trees either visually by a Symbol Layout Tree (SLT) giving symbols and their placement on writing lines, or semantically by Operator Tree (OPTs) describing mathematical content (i.e., quantities and operations [97]) as shown in Figure 2.1. One can switch from an SLT to OPT by using an expression grammar, but in our work, we focus our attention on extracting SLTs and we do not consider recognizing formula semantics.

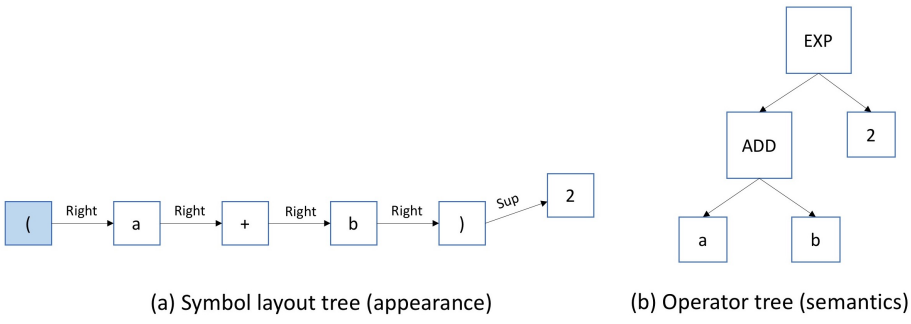


Figure 2.1: Math representations: symbol layout tree and operator tree for $(a+b)^2$ [95].

2.2 Math Recognition

Visual parsing of mathematical expressions converts input images to a representation of formula structure, which is a hierarchical arrangement of symbols on writing lines [95]. Techniques for parsing math can be classified by the criteria listed below.

Online vs Offline. Math expression recognition can be online or offline. For handwritten expressions (online), it is common to use strokes, sequences of points recorded in temporal order from pen down to pen up. For bitmap images (offline), the equivalent structure would be connected components or the sequence of pixels in sequential models.

Global vs Sequential. Three tasks need to be addressed when recognizing math equations: symbol detection (segmentation), symbol classification, and structure recognition (parsing). Some models approach these tasks sequentially and in isolation, such that no information is used from one task to another. However, global methods tackle this by jointly applying these three steps. In earlier attempts for solving math recognition globally, segmentation and recognition were addressed simultaneously. In these designs, the main criterion to group primitives is the probability of those primitives representing a given symbol [5, 30]. Techniques such as hidden Markov models (HMM) and Neural Nets (NN) can perform segmentation and recognition, simultaneously [1, 2, 5, 43, 49]. This is followed by an structural analysis to find the spatial relations between the symbols. In most cases to verify if the extracted equation is valid, syntactic analysis is applied as the last step in expression recognition.

Features. The main set of features for symbol classification are the visual features collected by CNNs in most cases. Whereas a common set of features used to represent the spatial relations between components (e.g., symbols) in segmentation and parsing are geometric features. Visual features have also been used for this task [95].

Symbol Classification. Different classifiers have been used for symbol classification. Among which K nearest neighbors [75, 80, 84], elastic matching [4, 78], support vector machines (SVM) [17, 46], rule-based classifiers [6, 28] require a prior segmentation, i.e., having the symbols extracted first. Therefore one can benefit from these classifiers when approaching math recognition sequentially. Yet, some techniques such as hidden Markov models (HMM) ,

Convolutional Neural Nets (CNNs), or Recurrent Neural Nets can perform segmentation and recognition simultniusely.

Relation Classification. In order to do layout analysis, the spatial relationship between the primitives, which can be symbols, strokes, connected components, or pixels, should be classified first. A number of methods have been proposed to recover symbol layouts such as graph parsing, Stochastic Context-Free Grammars (SCFG) [14], baseline extraction [5], encoder-decoders [20, 86] and adversarial learning [89, 102].

Structure Parsing. Syntactic analysis is usually the last step in expression recognition to confirm that the recognized equation is valid. Two principal approaches have been investigated in the literature: grammar or graph based analysis usually by applying grammatical rules or extracting a tree from the relation structures between primitives. So, the final output can be transferred into standard presentation format such as LaTeX or MathML. In some graph-based and grammar-based methods, language constraints are applied to reduce the computational time complexity. Graph rewriting can also be used for syntactic analysis. In this approach a sub-graph is replaced by a single node containing the syntax of the sub-expression [34, 42, 49].

In the following, we review studies focused on structure learning of math formulae that informed our research. Table 2.1 presents a summary of the studies we discuss in this section. In particular, sequential models have done a good job capturing the sequential nature of online handwritten equations. We review these techniques in 2.2.3.

2.2.1 Syntactic Methods (Constituency Parsing)

Syntactic Methods used for interpreting complex patterns in mathematical formulas as the notation has an obvious division into primitives, a recursive structure, and a well-defined syntax [5, 6, 9, 64]. Alvaro et al. [2, 3] presents an online handwritten math expression recognition system using Stochastic Context-Free Grammars (SCFG) defined at the symbol level. Segmentation is done by scoring symbol candidates using a symbol classifier and letting SCFG determines whether they should be merged or not based on the confidence scores generated by the SVM classifier. The system also searches over possible segmentations in CYK parsing algorithm. For parsing, first, lexical units are built from the set of symbol segmentation hypotheses. Second, a set of

Table 2.1: Summary of existing recognition systems for math expressions. Note that MTL stands for Multi Task Learning.

Paper	Attention	MTL	Input
Alvaro et al. [2,3]	-	-	online
HMM-based [49]	-	-	online
Rule-based [52,88]	-	-	online &offline
MEXREC [5]	-	-	online
Hu et al. [38,39]	-	-	offline
Zhang et al. [101]	tree-based BLSTM	-	online
LPGA [65]	graph parsing	parameterization(hard)	CNN
IM2TEX [20]	encoder-decoder	RNN-based	CNN
TAP [99]	encoder-decoder	GRU-based	CNN + online features
MAN [85]	encoder-decoder	multi-model GRU	offline & online features
Adversarial learning [89,102]	encoder-decoder	RNN-based	CNN + spatial

production rules in the grammar guide the stochastic parsing process in order to build a complete structure of the most probable expressions.

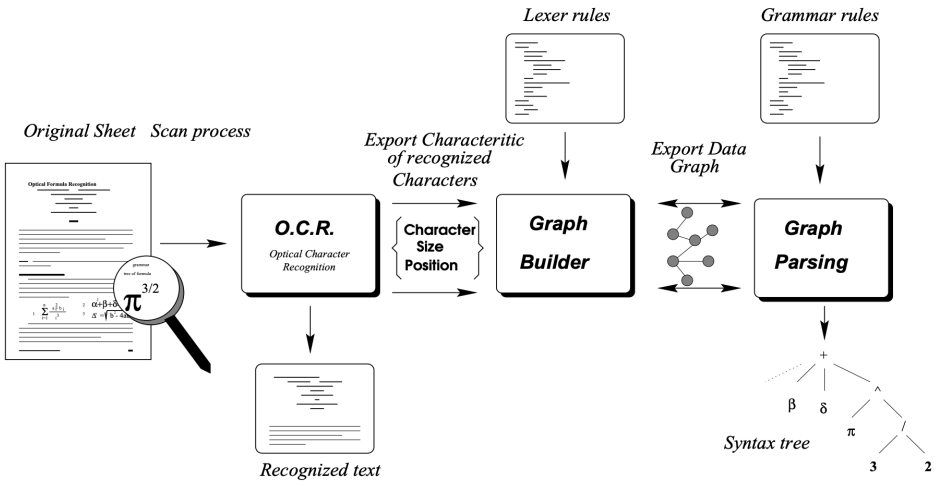


Figure 2.2: Lavirotte et al. proposed architecture for Optical Formula Recognition (OFR) [52].

Lavirotte et al. propose a system for extracting and recognizing mathe-

matical expressions in printed documents [52]. The syntax tree of a formula is built with graphical information which is recognized characters and their position. In this method character recognition, geometrical treatment, and grammatical treatment are studied separately in a design that each step provides the necessary information to the next process as shown in Figure 2.2. In the first step, OCR is applied to separate formulas from the rest of the document. This operation is not very challenging as the density of characters in formulas are not the same as text. This provides key information about symbols present on the sheet which will be used to construct a data graph in next step, such as recognized symbol, coordinates of the bounding box of each characters, reference point of the character (baseline). Second, from the result given by OCR step, a graph encoding relative positions of characters (geometrical treatment) is built. The graph builder constructs a graph with all recognized symbols in OCR. Each symbol is linked in all the 8 directions (left (l), right (r), top (t), bottom (b), top-left (tl), top-right (tr), bottom-left (bl), bottom-right (br)) with the closest symbols. A ninth type of connections also added: inside (i) e.g., in square roots. This provides a set of possible edges from which the ones who passed some test will remain as an edge in the graph. These criterion to prune edges are based on type of symbol and the manner to read or write mathematical expressions (left to right). Finally, context dependent graph grammars are applied to parse this graph. In order to remove ambiguities they add contexts to the rules. These ambiguities can appear when the global context is not considered in decision making resulting in more attention given to context.

Hidden Markov Model (HMM) based systems do not require prior segmentation and can handle symbol detection and classification simultaneously. In [49], first, HMM recognizer provides segmentation and recognition results for online data. Second, the spatial two-dimensional arrangement of the symbols are interpreted using a graph grammar approach for structure recognition. A graph grammar perform a rewriting system in a bottom-up manner, replace sub-graphs with a single node containing the syntax-tree of the recognized sub-expression. In this approach, the constraints concerning the writing from left to right and top to bottom can be relaxed.

Another work that avoids committing early to segmentation results by Winkler et al. studies the benefits of soft decisions [88]. In this method, first a directed graph is computed on the sorted input data from left to right in which

each node has an incoming edge from left and an outgoing edge to the symbol on the right. To label the unknown edges, first they define 5 relation type based on relative position of special symbols: fraction (numerator and denominator); summation, product and integration (upper and lower limits); root (power) and split the surrounding area of each special symbol into different regions. To label edges based on these spatial information, the system first identifies the special symbols e.g., fraction line. Thereby symbols, whose position is above or below the fraction symbol, are detected. To understand the relative positions of neighboring symbols to the target special symbol, distance is measured and the amount of shifting necessary for a non-ambiguous relation between the two symbols defines the relation type. If the relation is ambiguous, the graph is duplicated and different edges are used for describing the relation between these two symbols. After detecting symbol groups relative to this five types, the remaining undefined edges are either on the same line as the first symbol (lin) or it is part of the exponent (exp) or the index (ind) of the first symbol. In the last stage, a string containing the mathematical information is generated for each directed graph and verified syntactically. The remaining strings are sorted based on probabilities obtained during the symbol grouping. If more than one string passed the verification step, user must choose the correct one.

Awal et al. introduces a global approach for handwritten math recognition [5] called MEXREC. MEXREC finds the best possible grouping of strokes (symbol detection), identifies the symbol corresponding to each group (symbol classification), and finally interprets the expression according to the language model (structure analysis). These steps participate in calculating the global cost function CE. 1) A symbol hypothesis generator produces all possible symbol candidates which will be passed to symbol classifier and structure analyzer. 2) Symbol classifier assigns a recognition score and a label to each symbol hypothesis. The top N candidates for each hypothesis is kept and the recognition score is converted to a cost number by computing the negative logarithm of the recognition score. This way it can easily contributes to the global cost CE. The recognition scores are assigned to hypothesis generated in the last step, so to get rid of wrong hypothesis, the classifier has a rejection class which is considered during the training phase. 3) Geometric approach used to evaluate the structural costs. Structural analyzer associates a structural cost with each node calculated according to the mean square error between the expected (ideal) positions and sizes for a given relation and the observed

ones. The observations are computed for each node according to its baseline position (y) and its x-height (h). 4) A relation tree is constructed by applying grammar rules on vertical and horizontal axis and then applying a CYK parser when reaching elementary symbols. Finally, a decision maker selects the set of hypotheses that minimize the CE (global cost).

2.2.2 Minimum Spanning Trees (Dependency Parsing)

Mathematical expression recognition can be posed as searching for a Maximum score/Minimum cost Spanning (MST) representing symbols and their associated spatial relationships in a graph of primitives.

An MST-Based math expression recognition system using virtual link networks proposed by Suzuki et. al [25]. Recognition is done by finding a spanning tree for the network with minimum weight. There is a local score and a global score to weight the edges. The local penalty is calculated based on the distribution of relative sizes and positions for each relation type in parent-child links and the global penalty is based on predefined rules designed to exploit context. The edges will be weight with the sum of these two scores.

Another MST-based math parsing method is presented by Hu et al. [39] using Edmond's algorithm [23] to extract a tree from a weighted Line-Of-Sight (LOS) graph [38]. They use an LOS graph for representing the layout of primitives and symbols in math expressions [15]. LOS graphs have a higher expressivity than many other graph representations (e.g., Delaunay, geometric MST, k-NN for $k \in \{1 \dots 6\}$), while also reducing the search space for parsing. They also modify the shape context feature [8] with Parzen window density estimation. These Parzen shape contexts are used for symbol segmentation, symbol classification and symbol layout analysis.

An generalization of the work done by Hu et al. [39] is the LPGA (Line-Of-Sight Parsing with Graph-based Attention) model [65]. In LPGA individual CNNs are trained for segmentation, classification, and parsing. A hierarchical approach is used to first segment nodes into symbols with a binary classifier, and then generate a second graph on symbols and train two separate models to learn symbol classes and spatial relationships. We describe this system in more detail in Chapters 4 and 5 .

Zhang et al. [101] introduces another graph parsing method for recognizing online handwritten math. In the first step, an intermediate graph is derived

from raw inputs where each node represents a stroke and edges are added according to several defined criteria such as visibility between the two strokes sharing an edge (similar to LOS definition), or defining five regions for each node to look for the potential Right, Inside, Above, Below, Supscript and Subscript relationships between strokes. Multiple trees are extracted from this graph and recognized using the tree-based BLSTM model in which points in the strokes are encoded directly. The tree-based BLSTM architecture allows for direct labeling of nodes (symbol) and edges (relationship) from a graph modeling the input strokes.

2.2.3 Encoder-Decoder Models

State-of-the-art systems for visual math parsing use Recurrent Neural Networks (RNNs). RNNs were designed to work with sequence prediction problems e.g., predicting the next word in a sentence as it can memorize what has been calculated so far and use that to make accurate predictions in the next sequence [31, 36].

IM2TEX [20], inspired by the sequence-to-sequence model designed for image caption generation by Xu et al. [92] directly feeds a typeset formula image generated using LaTeX into a Convolutional Neural Network to extract a feature grid, see Figure 2.3. For each row in the feature map, a Recurrent Neural Network (RNN) is used to encode spatial layout information. The encoded fine features are then passed to an attention-based RNN decoder that emits the final expression string.

Another encoder-decoder model by Zhang et al. [99] uses pen traces collected from handwritten strokes on a tablet for parsing. The model architecture is shown in Figure 2.3. In their model, the encoder is a stack of bidirectional GRUs (Gated Recurrent Units) [13] which is a gating mechanism in RNNs, while the parser (decoder) combines a GRU-based language model and a hybrid attention mechanism consists of a coverage-based spatial attention and a temporal attention. Unlike the attention module in IM2TEX model, which scans the entire input feature map at pixel level, the spatial attention in TAP learns an alignment between input strokes and outputs. The role of temporal attention in TAP model is to learn when to rely on the product of spatial attention and when to just rely on the language model as there is no spatial mapping for tokens representing spatial relationships e.g., superscript

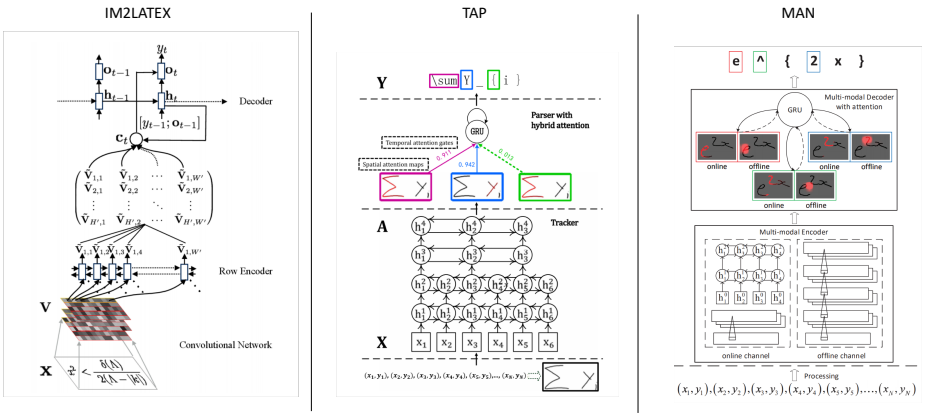


Figure 2.3: Encoder-decoder math recognition models. From left to right: IM2TEX [20], TAP [99], and MAN [85].

'∧' or subscript '_'.

A recent architecture in encoder-decoder networks used for math expression recognition has two input branches to encode both online and offline features [85]. This Multi-modal Attention Network (MAN), first take dynamic trajectories and static images for online and offline channels of the encoder respectively as shown in Figure 2.3. The output of the encoder is then transferred to the multi-modal decoder to generate a LaTeX sequence as the mathematical expression recognition result. This architecture is a hybrid design of both TAP and IM2TEX models explained earlier, except the fact that they use CNN layers in their online channel instead of using a stack of RNNs. Each decoder has their own attention modules. For the online branch, the attention module highlights strokes, whereas in the offline module it weights the pixels. Once the attention weights are calculated, the multi-modal context vector can be obtained by concatenating the single online and offline context vectors. A recent addition to sequential models [86, 89] exploits DenseNet [40] for encoding images. In this work an improved attention model with channel-wise attention is applied before spatial attention.

Domain-Invariant Features Learning. Generative Adversarial Networks (GANs) have been introduced for adversarial learning by Goodfellow et al. [33] originally for generative learning, but extensions of GANs have been used for invariant representations in different domains [10, 76]. Generative learning is an semi-supervised task in machine learning that involves discovering the patterns in input data in such a way that the model can be used to generate new examples that could have been drawn from the original dataset [76].

Zhang et al. [102] proposed an adversarial-learning-based method which learns from prior knowledge of printed templates and improved the traditional feature extractors to learn writer-independent features. Their system composed of three neural network components: (1) a feature extractor that generates features from handwritten and standard printed characters, (2) a classifier that takes the extracted features, and (3) a discriminator that guides feature extractor to learn prior knowledge. These are jointly optimized by the adversarial training algorithm with the goal of helping feature extractor focus only on writer-independent features. Liu et al. [61] proposes a similar idea for image feature learning for scene text recognition.

Another encoder-decoder architecture proposed by [89] highlights the importance of domain-invariant features learning in handwritten math recognition to improve the robustness of the recognizer with respect to writing styles. This model equipped with an attentional encoder-decoder recognizer R and a discriminator D. Similar to previous auto-encoders, the convolutional encoder (DenseNet [40]) encodes the input image and then the decoder parses these feature maps into LaTeX strings. The role of discriminator is guiding the recognizer to learn invariant features for making R more robust to writing-style variations. Handwritten equations paired with their printed templates are fed into the system as a pair. In this design, recognizer should correctly recognize both handwritten and printed inputs and try to extract indistinguishable features from the paired images and make it harder for discriminator to judge which features are from handwritten and which ones are from typeset equations.

2.3 Motivations for our Approach

Compared to sequential models, graphs are more natural and general for representing math equations as trees. In our model, primitives form the nodes of the input and output graph, while edges between primitives in the output represent symbol segmentation and spatial relationships between symbols (e.g., right, superscript). Our attention module uses this input graph to query (filter) CNN features from a single feature representation of the input image to efficiently obtain inputs for multiple classification, segmentation, and relationship decisions.

Our system does not need to learn an alignment between input strokes/connected components and outputs similar to encoder-decoder models, as we directly output a graph-based hierarchical representation of symbols on writing lines (as a *Symbol Layout Tree (SLT)*), and not a string. We also do not use expression grammars for language models, instead relying only upon the sets of symbol and relationship classes along with visual statistics captured by our CNN models. The language model in our system is data-driven and learned through the symbol/relationship labels, and the statistics over them in the training expressions.

2.4 Visual Parsing

Visual parsing is important for a variety of image understanding tasks, including real-world complex vision-language problems such as image caption generation [45, 92], visual relationship detection [16, 35, 73], scene graph generation [93], table detection, and form and table parsing [18, 96].

In the following, we first provide a brief description on Convolutional Neural Networks (CNNs). We present different approaches that CNNs can benefit from an attention module. Next, we explain how these CNNs equipped with attention make the multi task learning possible. This is particularly interesting to study as structure learning can be identified as a series of tasks to tackle, e.g., detecting objects in a scene, classifying those objects and finding the relation between them in a scene understanding problem or symbol detection (segmentation), classification of detected symbols, and detection/classification of relations between them in math structure parsing. Multi Task Learning (MTL) frameworks allow for solving these tasks simultaneously. Finally, we

review recent studies based on MTL approaches for multi-target (multi-query) tasks as there are usually multiple targets per each task in equations e.g., multiple symbols to classify.

Table 2.2 demonstrates a summary of visual parsing methods that will be discussed in this Section. We summarize whether these system are using attention modules, are they trained for multi-task learning, what is the feature extraction approach, and what is the approach constraints.

Table 2.2: Summary of visual parsing systems using CNNs. Note that MTL is short for Multi Task Learning.

Paper	Method	Attention	MTL	Constraints
Learn to pay att. [41]	ConvNets	soft	False	designed for one query per image
MGCAM [81]	ConvNets	hard and soft	False	designed for one query per image
Valve filter [24]	ConvNets	hard and soft	False	requires ROI map
MTAN [58]	ConvNets	soft	True	requires a parallel network per task
Cross-stich Net [69]	ConvNets	soft	True	requires an individual CNN for each task
Graph R-CNN [16], [94]	GCN	soft	True	requires another network to prune graph edges
Feature Fusion [55]	ConvNets	-	False	requires resizing features which reduces the speed

2.4.1 Convolutional Neural Nets (CNN)

Convolutional Neural Networks (ConvNets or CNNs) is a machine learning algorithm that have proven very effective in learning patterns in images, in particular for image recognition and classification [50,53,54]. The basic building blocks in CNNs are (1) Convolution layer (2) Non Linearity operation (3) Pooling or sub sampling (4) Fully connected layers (classifiers). The purpose of convolution in CNNs is to extract features from the input image. Unlike fully connected layers, that ignore the topology of the input, as each hidden unit is connected to all other units in the previous layer, convolution layers keep the spatial relationship between pixels by learning image features using small squares of input data [54]. This is achieved by forcing the receptive field of hidden units to be local.

2.4.2 Feature Fusion Methods

One common approach to detect multiple objects using a single deep neural network is extracting features in multiple scales [7,48,55,56,59,60,74]. In case

of having multiple targets/queries, usually not all objects are in the same size so when using CNNs it makes sense to extract features from multiple scales (intermediate layers) e.g., classifying a dot versus a long fraction line in an equation image.

In literature, there are two ways to merge multi-scale feature maps: element wise summation/product or concatenation. The latter is preferred as there is no need for feature maps to have the same number of channels. HyperNet [48], Parsenet [60] and ION [7] concatenate features from multiple layers before predicting the result. FPN [56], DSSD [29] and SharpeMask [74] employ a top-down structure to combine the different level features together to enhance the performance in a fast approach. These designs need multiple feature merging processes as features are fused from top to bottom layer by layer.

FSSD [55] is an extension of Single Shot Detection (SSD) [59] work which extracts features from intermediate layers and concatenates them resulting in improving performance significantly over SSD with just a little speed drop. In the feature fusion module, Conv 1×1 is applied to each of the feature maps extracted from different layers to reduce the feature dimensions. Then features either down-sampled with max-pooling or up-sampled with bilinear interpolation to the chosen size which is the size of conv4-3's feature map (38×38). Feature maps from different layers always have different ranges of values, so a Batch Normalization layer applied to features after concatenation and fed to multibox detectors to predict the final detection results.

2.4.3 Attention in CNNs

The broad term attention is used for a module in the network's architecture that is responsible for learning the association either between the input and output elements (General Attention) or within the input elements (Self-Attention) [72]. Including attention in CNNs can be a hard (deterministic) or a soft (stochastic) decision. A hard decision on highlighting the relevant parts of the input image often happens by simple parametrization such as masking out the salient areas with binary masks, whereas the soft-attention approach is probabilistic and can be trained by back propagation. Thus, it can be trained along with the main branch in the CNN and make the optimization easier in a global manner as all parameters are updating together according to the global

cost function.

Song et al. propose a model that learns features separately from the body and background regions for the task of Person Re-identification (ReID) [81]. First, a pair of inverse attention masks, which are essentially segmented binary masks of target and background, are generated for each input image (hard attention). The intuition behind that is to black out both the target and the background in different instances and generate a pair of inverse attention masks. Each RGB image and corresponding mask pairs are then fed into the second module, a multi-branch CNN model, to generate three feature streams: full-stream, target-stream and the background-stream (soft attention). Therefore, the proposed approach have three streams: the input image(f_{full}), the foreground image or positive mask (f_{att}^{+}), and the background image or negative mask (f_{att}^{-}), see Figure 2.4. The authors refer to this two-step system as Mask-Guided Contrastive Attention Model (MGCAM). Negative stream will add context/background information for each target. It should be mentioned that each stream generate a 128-dimensional feature vector at the end, but the one from general stream will be used for sample representation and the features from the other two streams are only used to guide the feature learning of the full stream. The input binary body masks are useful both for (1) reducing the background noise which makes the model be more robust to different background conditions and (2) presenting body shape information which is a very important feature for person re-identification.

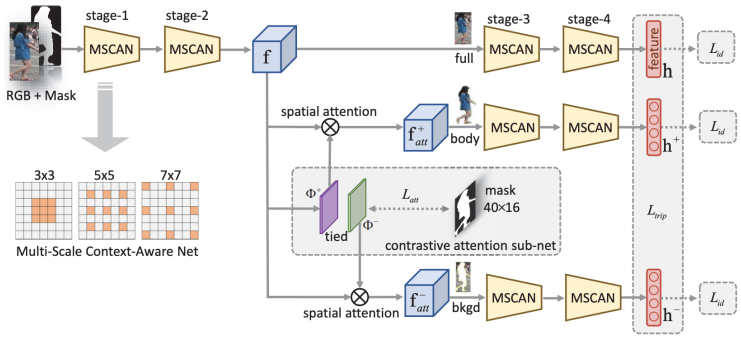


Figure 2.4: Framework of Mask-guided Contrastive Attention Model (MGCAM) for person ReID [81].

Another model using binary masks to guide the attention mechanism is [24]. In this work, a single filter is trained for each binary mask while training the main CNN model. The output feature maps from this trained filters and the input binary masks are called relevance maps. Outline of this approach is shown in Figure 2.5. The intuition behind introducing relevance maps for weighting the activation maps in the network is trying to avoid removing all the background information, which happens in the binary masks multiplication. Trainable attention masks learns which parts of the background are actually relevant to the classification of the target and worth keeping. The image and the binary masks are each passed through a separate convolution layer to generate a feature map and a relevance map respectively. The input to the next layer would be the result of element-wise multiplication of the feature map and the relevance map. The filter that is learned to generate the relevance map is called valve filter and its weights are learned by the net in the same way as image filters enabling the model to be trained end-to-end. It should be mentioned that in the current implementation of this model, the valve filter acts only on the first layer of the convolutional neural net, and the rest of the net remains unchanged.

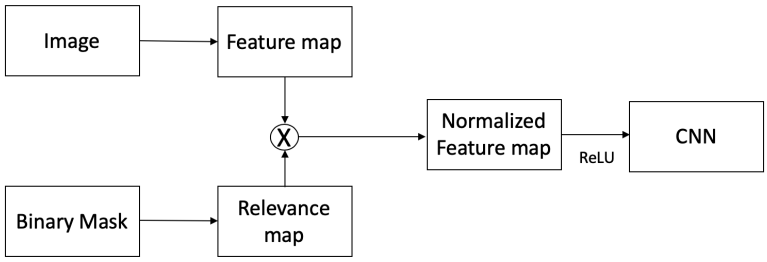


Figure 2.5: The relevance map approach for introducing binary masks as input to the CNN.

Computing relevance maps from binary masks is one way to keep the background information which is important for the target. Alternative approaches are (1) to black out both the target and the background in different instances and generate a pair of inverse attention mask (f^{+}_{att}, (f^{-}_{att})) to pass as separate inputs to the net as used in the MGCAM model [81] or (2) to use the binary

mask as another input for the net by concatenating it with inputs of specific layers of the net [7, 44].

Jetley et al. in [41] demonstrate an approach to soft trainable visual attention in a CNN model. This method encourages the earlier filters in the model to learn similar mapping with the ones that generate the global image descriptor g in the last layer by letting the intermediate feature vectors contribute directly to the final decision. Therefore, intermediate feature vectors are enforced to be more compatible with the final feature vector that goes into the linear classification layers at the end of the pipeline. The intuition behind this is providing the larger field of view of the last layer to the higher resolution earlier layers in the CNN. As a result, local descriptors in the intermediate layers can benefit from being more compatible with the global features as it provides the larger field of view of the last layer, while keeping the higher resolution, compared to the final feature map. Each of the local feature vectors (l_i) are weighted by a compatibility score which shows their similarity to g. The proposed model is used for multi-class classification. The authors demonstrate that soft trainable attention improves performance by 7% for CIFAR-100.

Relation to our work. In the studies we reviewed so far regarding attention in CNNs, there is always one task to do, one question to answer: e.g., What is this animal? who is this person? or is the vessel empty? [24, 41, 81]. This means the attention module has to highlight the relevant areas for one specific target in the input image. Our proposed model needs to label all nodes and edges of the input graph at each iteration simultaneously. To do that we have 3 classifiers, each answering one of the following questions:

- Segmenter: are these connected component belongs to one symbol?
- Classifier: what is the symbol class of this connected component?
- Parser: what is the relation of these two components?

Therefore, we need our attention module to understand what are the targets in the input equations and provide a feature vector for each target by filtering out different regions of the main feature map generated from the input image. Inspired by [24], we have a stream of binary masks going into our attention layer, which is made of a convolutional layer with 4 kernels per input binary mask, and the relevance maps coming out will be resized and multiplied with the final feature map from the main network to generate features

for multiple targets (nodes and edges). We pass our target masks (binary) to network similar to target stream in [81]. We have explored having a negative stream capturing the background for each taget. Results of this experiment is reported in Chapter 6.

Unlike [24,81] relevance maps in our soft attention module are not applied on the first convolutional layer. We apply them on the final activation map as we would like to keep the feature set shared and avoid dividing the main branch at the very beginning. This is helpful since we maintain most of the weights shared between the tasks. Training for multiple tasks simultaneously is called Multi Task Learning (MTL). Below, we review some of the major works in MTL which informed our approach.

2.4.4 Multi Task Learning (MTL)

MTL is a machine learning term which refers to solving multiple tasks at the same time often resulting in improving the generalization, learning efficiency and the performance of task-specific models [11]. When learning for each task given the shared representation, what is learned for one task can help other tasks and this type of information can be easily ignored by being focused on one task [11, 21, 26, 51]. This also introduces a generalization in feature learning which prevents models from memorizing features for a specific task (over-fitting). So, the network is encouraged to learn a general representation to avoid over-fitting, while providing the ability to learn features tuned for each task to avoid under-fitting. This is specially useful for those related task that doesn't come with a very large datasets [11].

MTL has been broadly used for transfer learning, [62, 69] or for learning related tasks jointly such as pose estimation and action recognition, or surface normals and edge labels in room layout [32, 87]. A multi-task learning is generally used with CNNs and the proposed architecture has a shared part and a task-specific part. Most of the studies in this area discuss one of the followings: (1) Joint loss definition. Loss function, which weights the contributions of each task, should enable learning of all tasks with equal importance, without allowing easier tasks to dominate. (2) The attention module, which is discussed in details in the previous section. (3) The trade-offs between different combinations of task-specific and shared representation in the architecture. The parameter sharing in MTL is either hard or soft [69]. In hard parameter shar-

ing, the network has some shared intermediate layers which then gets divided into task-specific output layers, similar to our proposed model architecture. In soft parameter sharing, there is a network trained for each task and there are units that connect these networks by defining how much sharing is needed between each block e.g., cross stitch units in [69].

In [69], Cross-stitch Networks for Multi-task Learning is proposed. There is a trained network for each task which are all linked to each other with Cross-stitch units, as shown in Figure 2.6. At each layer of network, cross-stitch units learns a linear combination of the activation maps which defines how much sharing is needed. This answers one of the most significant obstacles in designing a multi task learning architecture which is what should be the combination of shared and task-specific layers?

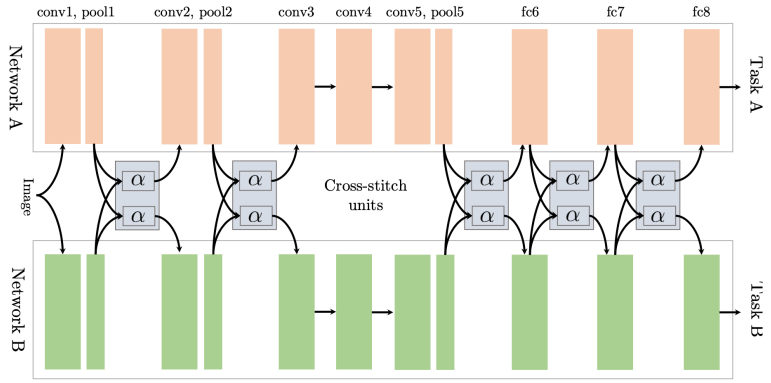


Figure 2.6: Using cross-stitch units to stitch two convolutional neural nets in [69].

Liu et al. [58] introduces Multi-Task Attention Network (MTAN) which is an example of a soft parameter sharing by having a single shared network and multiple task-specific attention networks with connections to the main network. The main branch is learning the global features from the input while each attention module is applying a soft attention to a particular layer of the global feature stream and passing only the features that are related to the question they are designed to answer. Each of these attention modules, except for the first one that only takes the shared features, takes the shared features

from the parallel block in the main branch and the task-specific features from the previous block. In this design, the question of how much of a network should be kept shared and how many layers should be trained for specific tasks is answered as these two parallel networks are connected with units defining how much sharing is needed between each block.

The model objective is defined as shown in equation 2.1 for K tasks, input X and task-specific labels $Y_i, i = 1, 2, ..., K$, as,

$$\mathrm{L}_{tot}(X, Y_{1:k}) = \sum_{i=1}^{k} \lambda_i L_i(X, Y_i) \tag{2.1}$$

This is the linear combination of task-specific losses L_i with task weightings λ_i. Training a network for multiple tasks is possible with a loss function that can gives a balanced attention to all tasks by let the individual tasks have equal importance in the final loss and avoid the dominance of the easier tasks. An adaptive weighting method named Dynamic Weight Average (DWA) which is inspired by GradNorm [12] is introduced in this work. DWA monitors the the rate of loss change for each individual task over time and then weight the loss using that information. Unlike GradNorm which requires the network gradient DWA can be computed only by having the task loss as shown below. They define the weighting λ_k for task k as:

$$\lambda_k(t) := \frac{K \exp\left(w_k(t-1)/T\right)}{\sum_i \exp\left(w_i(t-1)/T\right)}, w_k(t-1) = \frac{L_k(t-1)}{L_k(t-2)} \tag{2.2}$$

Here t is an iteration index, and T represents a temperature which controls the softness of task weighting, a very large T means tasks are weighted equally ($\lambda_i \approx 1$). Note that for $t = 1, 2$ weights are initialzed as 1 ($w_k(t) = 1$).

Relation to our work. In our work, sum of all the query errors contributing to the final tree (all nodes and tree edges) are used to compute the joint loss. In the proposed Tree Loss, (read more in Chapter 5) at each iteration, only the edges that remains in the output tree plus the ground truth edges contribute to the error to harvest hard negatives for training and avoid the possible imbalance in queries in each task due to fewer nodes compared to edges (edges are approximately 3.3 times more than nodes in Line of Sight graphs).

The MTL studies we reviewed above each has one object per task, whereas in our proposed system we have multiple queries per task. We have to define all relationships between components and label all those components to be

able to successfully recognize a math equation. Similar to our problem, to understand an image, first, the objects in the image should be recognized and second, all the relationship between objects should be defined. Graphs has been used for this purpose for a long time. The following section reviews the works that exploits graphs for understanding the spatial relations.

2.4.5 Graph Parsing

The most similar work to our known is from Dai et al. [16]. To define relationships between objects, they first do object detection in the input image. The next step is to produce a set of object pairs from the detected objects. With n detected objects, they form $n(n-1)$ pairs with some unlikely relations filtered out with a low-cost neural net. Finally, in [16] each retained pair of objects will be fed to the joint recognition module. Taking into account multiple factors and their relations, this module will produce a triplet in the form of (subject, predicate, object), as the output.

Inspired by this work, [94] use Graph Convolutional Networks encoder plus Long Short-Term Memory decoder (dubbed as GCN-LSTM) architecture to encode both semantic and spatial object relationships. The goal of this study is exploring visual relationship for image captioning. In this work, salient image regions (nodes) are computed using off-the-shelf object detection algorithms e.g., Faster R-CNN [77]. Second, two directed graphs are generated on the detected regions, one for spatial relations (e.g., inside, overlap, over, etc.) and another one for semantic relations (e.g., riding, eating, biking, etc.). Graph Convolutional Networks (GCN) are then exploited to encode region representations and visual relationship in both graphs. Next, the learnt relation-aware region representations are feed into individual attention LSTM decoders to generate the sentence. To integrate outputs of two decoders, the predicted score distributions on words from two decoders is averaged at each time step. The results from the two decoder are fused in an inference stage adopting a late fusion scheme to linearly fuse the results from two decoders. The semantic graph decides which relations should be established between objects, leaving the spatial relations between image regions unused. A second graph defining spatial regions is then generated over the detected regions. When doing classification in the semantic graph, similar to our work, a 'NoRelation' class is added to the set of class labels N_{sem}. They compute the probability

distribution on all the $(N_{sem} + 1)$ relation classes for each object pair. If the probability of NoRelation is less than 0.5, a directed edge connects the region vertex of parent (subject noun) to the region vertex of child (object noun). The relation class with maximum probability is regarded as the label of this edge.

Yang et al. [93] also use a graph parsing approach for scene understanding. Their proposed model called Graph R-CNN has three modules: (1) object node detection (2) relationship edge pruning with a Relation Proposal Network (RePN), and (3) graph context integration. RePN learns to compute 'relatedness' scores between object pairs which are used to prune unlikely scene graph connections. Next, the proposed attentional graph convolution network (aGCN) [47] is applied on pruned graph to encode higher-order context throughout the graph and provide information on each object and relationship representation based on its neighbors.

Relation to our work. Our method differs in three aspects: (1) We use connected components from rendered handwritten strokes or images as input generating a Line-of-Sight graph. Whereas in these studies nodes are usually a set of detected objects and the edges defining the relation between them are computed with an additional low-cost network. (2) We train our model to do object segmentation, classification and relation prediction jointly using a multi-task classification framework. (3) Unlike [93], our model handles multiple queries (i.e., classification problems) using a soft attention module, so that attention masks generated for each node and node pair are refined through back propagation when training the model end-to-end.

2.5 Summary

In this chapter, we have covered related works from structure parsing in different areas and systems designed specifically for math parsing. We discussed math representations used in different systems (OPT vs SLT) for presenting the recognized equation and clarified that our system outputs Symbol Layout Trees (SLTs).

In Section 2.2, we listed different aspects of a math recognition problem that can be handled with different techniques such as online vs offline recognition. We then discussed syntactic methods, MST extraction, and finally encoder-decoder approaches. We mentioned the limitation of autoencoders

e.g., generating string outputs, searching over all pixels in the input image, forcing data into 2D and also explained why we think math recognition can be solved by adapting a simple convolutional neural networks (CNN).

In Section 2.4, we have got into Convolutional Neural Nets and discussed some of their attributes that make math parsing with CNNs possible such as: Attention Module to generate query features and classifying all symbols and relations in an equation in one iteration. Another CNN-related method that we have covered is Multi Task learning (MTL) that permits training for segmentation, classification and parsing simultaneously. We reviewed some of the Graph Parsing studies using CNNs and provide detailed explanation on how these systems inspired QD-GGA and how our modules differ from what they are presenting.

Chapter 3

Evaluation Methods and Datasets

In this chapter, we first introduce the CROHME and INFTY datasets containing handwritten formulas and scanned images of typeset formulas in Section 3.1. These two datasets are used for evaluating QD-GGA and LPGA performance. Section 3.2 presents the tools and metrics used to evaluate our techniques. We introduce SymLG that makes the comparison between systems that generate LaTeX strings and systems that output trees possible. Finally, we talk about how the new metrics (SymLG) were used in CROHME 2019 competition [68].

3.1 Datasets

Our model can take both online (handwritten strokes) and offline (images) datasets as inputs. We use the CROHME dataset [70] to benchmark our model against handwritten recognition systems and InftyMCCDB-2 [15,82] to evaluate our recognition system for typeset math equations.

CROHME. The stroke data provided in CROHME dataset is a sequential list of points representing sampled (x,y) coordinates as each stroke is written. The ground truth files provide segmentation information (stroke grouped into symbols) along with the symbol classes and the symbol structure. The ground truth files are provided in InkML format (an XML tag-based representation)

and Label Graph files (LG files), which have a simpler CSV-based representation.

For our experiments on handwritten equations, we use CROHME 2019 [68] train, test, and validation sets. The dataset is collected and labelled by Mouchère et al. [70]. Training set for CROHME 2019 is expanded by adding previous test sets (2013, 2012) to it. These three are going to be used as a train set which contains 9993 expressions. The validation set for CROHME2019 is the previous test set in 2014 (986 expressions). Finally, a new test set for the main task is provided containing 1199 expressions. This dataset includes 101 symbol classes.

InftyMCCDB-2. For our experiments on typeset equations, we use InftyMCCDB-2, a modified version of InftyCDB-2 [82] which contains mathematical expressions from scanned article pages. The dataset includes 213 symbol classes and the expressions range in size from a single symbol to more than 75 symbols, with an average of 7.33 symbols per expression. In the original set there are 21,056 mathematical expressions, but the formulas with matrices and grids are removed in the modified version, leaving 19,381 formulas left in InftyMCCDB-2 which is divided to a training (12551 images) and testing (6830 images) subset with approximately the same distribution of symbol classes and relation classes.

The ground truth files provided for InftyMCCDB-2 are at connected component level [15]. For each connected component, the bounding box and the symbol it belongs to is provided. This information is presented in Label Graph (LG) files [70]. A label graph file stores connected components with individual identifiers, groupings of components into symbols with their labels, and finally directed Symbol Layout Tree edges. There are seven spatial relationships in InftyCDB-2: horizontal (HORIZONTAL), right/left superscript (RSUP and LSUP), right/left subscript (RSUB and LSUB), above (UPPER) and below (UNDER). Another modification in InftyMCCDB-2 is adding a new spatial relationship called "PUNC" for separating punctuation relation from the baseline structure [15]. As seen in Figure 3.1, punctuation symbols are spatially more similar to subscripted symbols than horizontally adjacent symbols. Having punctuation symbols on the baseline separated from symbols on the main baseline, and placed in their own nested region relative to their parent symbol allows horizontal relationships to be represented in a more consistent manner, and punctuation to be associated with sub-expressions more accurately.

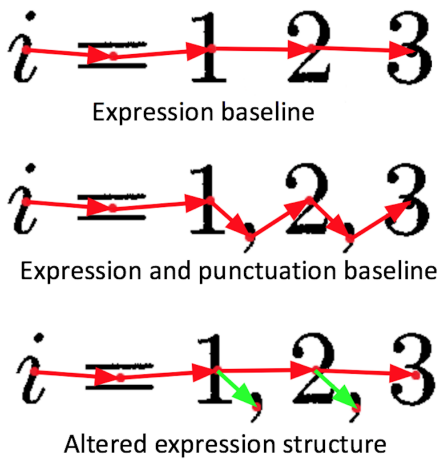


Figure 3.1: Modifying SLTs for punctuation in InftyMCCDB-2. In the middle image, punctuation is represented using horizontal relationships (red edges) despite the marked shift in vertical positions. To avoid inconsistent 'horizontal' relationships, a punctuation relation (PUNC, shown in green) is defined.

3.2 Evaluation Methods

In this section, we introduce tools and metrics we use to evaluate our systems. Evaluation of structural recognition systems for math expressions is often complex because of the interaction between input primitives, detected symbols, and their spatial relationships. For example, when a relationship between one correctly classified symbol and one incorrectly classified symbol is detected correctly, can we find a tool that allows for counting the partially correct structure (relation) and reporting the incorrect part (symbol class) from the above example?

The evaluation tool introduced in CROHME competitions [70] called the LgEval library allows partially correct recognition results to be located and measured precisely. Formulae are represented by labeled adjacency graphs over strokes (LG). This representation allows errors to be unambiguously identified by lgEval tool. However, this requires primitive-level (stroke-level) information and encoder-decoder systems generating string outputs cannot provide primitive-level information (segmentation). To be able to use lgEval tool with

these types of techniques we introduced SymLG format.

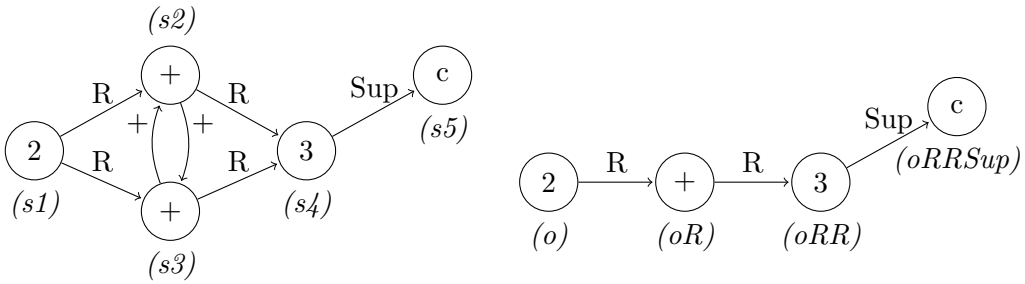
Primitive-level error analysis. LgEval metrics include formula, symbol recognition rates, along with recall and precision metrics for detection (segmentation) and detection + classification for both symbols and relationships [70]. In primitive-based output representations, such as the stroke label graph (LG) in Figure 3.2(a) [98], all primitives in a symbol share the same spatial relationship with primitives in the related symbol (e.g., for 'a' and '+'), and symbol segmentation is given using bidirectional edges labeled with their associated symbol's class (e.g., for '+'). Label graphs (LGs) permit detailed error analyses. We can determine precisely which primitives were grouped correctly or incorrectly for the target symbol and evaluate their relations even if they are grouped incorrectly. This allows us to make new analyses using small subgraphs in ground truth and then count the number of times different errors are made for each. For instance, if we choose two nodes for the small subgraph, we can count and visualize incorrect label graphs representing any combination of classification, segmentation and relationship mislabelings for primitives in a pair of symbols (subgraph is made of two nodes and one edge). See Figure 6.6 in Results Section for error analysis with LgEval tool choosing one node and two nodes in the subgraphs.

Symbol-level error analysis. For recognition systems that produce LaTeX strings, we have presented a technique that allows string and tree-based formula structure representations to be meaningfully compared at symbol-level [66]. In this method, LaTeX strings are first converted to MathML using `pandoc`.[1]. This transformation preserves symbols and spatial relationships while removing formatting directives (e.g., `\quad`, fonts). Once we have a MathML representation for a formula, we generate symbol identifiers using the spatial relationship sequence (edge labels) from the root symbol, e.g., "RRSUP" when the symbol has two "Right" and one "SUP" relation from the root node (see Figure 3.2(b)). Identifiers allow us to address symbols on writing lines from different structure representations. This produces a symbolic representation for recognition outputs. This new symbolic representation is called: The **symLG** (symbolic LG) representation.

Figure 3.2 shows two graph representations for the same expression '$2+3^c$.' In the Stroke Label Graph (LG), there are 5 nodes (one per stroke), and edges represent segmentation and spatial relationships between pairs of strokes (in-

[1] https://pandoc.org

cluding 'no relationship'). For the Stroke Label Graph, node identifiers are for individual strokes. In the Symbol Label Graph (symLG) on the right, there are 4 nodes (one per symbol) and edges represent relationships between symbols (no segmentation information is provided). For symLG, node identifiers are constructed from the sequence of relation labels on the path from root to the symbol. For example, 'c' in Figure 3.2 has the identifier 'oRRSup' (origin/root, Right, Right, Superscript).



(a) Stroke Label Graph (LG) **(b) Symbol Label Graph (symLG)**

Figure 3.2: Different graph representations for formula '$2 + 3^c$' written using 5 strokes. Node identifiers are shown in brackets.

The symLG representation allows us to identify errors as symbol classification errors, relationship classification errors, or structure errors. Note that this change in representation has several impacts on the results; in particular, segmentation information cannot be calculated - it is possible for a formula to be recognized with the correct Symbol Layout Tree, but without correctly segmenting symbols. This makes the expression rate less strict than at the stroke-level. Also, because labels on relationship paths identify symbols, when symbols do not appear at expected locations in the output they are treated as missing ('ABSENT' in LgEval), which leads to an underestimate of symbol recall. Figure 3.3 shows the results of using LgEval tools for error analysis on SymLG outputs generated for CROHME 2019 with QD-GGA best configuration. Since symbols are identified with path from root node, most frequent errors are related to absent nodes.

Figure 3.3: Error analysis on SymLG outputs for QD-GGA (best config) on CROHME2019 dataset choosing one node (left) and two nodes (right) for subgraphs. Since symbols are identified with path, most frequent errors are related to absent nodes.

3.3 SymLG Applications

We use symLGs to evaluate the IM2TEX system by Deng et al. [20]. Previously, the system was evaluated by rendering each output LaTeX string, and then calculating the exact pixel matching with the corresponding ground-truth formula image. Our symLG-based metrics were also used for the recent ICDAR 2019 CROHME + TFD competition [68], as they are simple to understand and allow systems that generate primitive-level and symbol-level results to be compared directly.

IM2TEX re-evaluation. Most of the encoder-decoder systems have been evaluated using string-based metrics such as exact matching, string edit distances, n-gram-based metrics such as BLEU, or by computing distances between images produced after rendering TeX formulas. Deng et al. [20] use the BLEU metric based on agreement between n-grams in the LaTeX character strings. Neither the image matching metric nor BLEU number provides information on failure cases or error analysis.

As shown in Table 3.1, our symLG metrics provide measures for **Symbols Det.**: correct symbol detection (i.e., symbols exist at expected spatial

Table 3.1: symLG im2latex-100k results (9,378 test formulas). Shown are correct symbol/relationship locations (Detection), symbol/relationship classes (Det.+Class), formula SLT structure ignoring symbol labels, and valid structure and symbol labels (Str.+Class).

	Symbols		**Relationships**		**Formulas**	
	Det.	Det.+Class	Det.	Det.+Class	Str.	Str.+Class
IM2TEX	95.70	93.48	95.50	95.50	86.79	**83.15**

locations), **Symbols Det.+ Class**: correct symbol locations and labels, **Relationships Det.**: correct relation detection, **Relationships Det.+Class**: correct relation classification (labels), **Formulas Str.**: correct structure of the formula, and **Formulas Str.+Class**: structure and symbol classification accuracy at the expression level. Note that because spatial relationships determine symbol locations, a correctly *detected* relationship is also correctly classified.

Im2latex-100k data set used for evaluation of im2tex system provides 103,556 different LaTeX math equations along with rendered pictures. Formulas are extracted by parsing LaTeX sources of papers from the arXiv. Source files are obtained from tasks I and II of the 2003 KDD cup (Gehrke, Ginsparg, and Kleinberg 2003) containing over 60,000 papers. The extracted equation are rendered in a vanilla LaTeX environment and the rendered PDF files are converted to PNG format. The final dataset contains 103,556 images of resolution 1654×2339 with black equations against transparent background.

We were able to convert 9,378 of the 10,355 test formulas (90.6%) from LaTeX to MathML using `pandoc`. Many failed conversions are caused by invalid syntax (e.g., missing brackets). For the 9,378 formulas that were converted successfully to MathML, we are now able to report that the percentage of correct formulas with both correct symbols and structure is 83.15%, that 93.48% of symbols are in the proper location with their correct class, and that 95.50% of spatial relationships are correct. The metrics previously reported by the IM2TEX authors include BLEU (tok) at 58.41, BLEU (norm) at 87.73, exact image-based pixel matching of 77.46, and image-based pixel matching with a whitespace tolerance (-ws) of 79.88 [20].

Moreover, using symLGs we can provide detailed error analysis that string and image-based representations cannot capture. The most common error is

'missing' symbols (see Absent nodes in Figure 3.3). This happens because symbols are identified by their absolute path. Therefore, errors in structure lead to errors in symbol detection and classification. Note that this also means that correctly detected symbols at the incorrect position in a symLG are identified as invalid.

CROHME 2019 competition. The stroke-based LG files used in previous CROHMEs allow all segmentation, classification, and structural errors to be identified unambiguously, even when segmentations disagree [71]. However, with the success of encoder-decoder-based systems that generate LaTeX output, a new representation is needed - these systems do not output information about stroke segmentation or the location of symbols in the input, instead producing Symbol Layout Trees directly.

To compute the similarity of two Symbol Layout Trees in our symLG representation, we use an adjacency matrix. Labels on the diagonal define symbol labels, while off-diagonal elements represent spatial relationships between parent and child symbols. Using this representation, we can determine how formulas in an SLT representation differ in structure and symbol labels, but not the correspondence between symbols and relationships in a symLG file and the input data (i.e., strokes/images). Still, the symLG representation allows existing metrics and tools designed for evaluation of stroke-level LG files at the symbol level to be used directly. We note that symLG is closely related to the tree-based symbolic representation from earlier CROHME competitions [71], but permits more detailed error analysis.

Recognition systems that produce LaTeX strings or stroke-level Label Graphs (LGs) both have their outputs converted to symLG. This allows systems producing stroke-level and symbol-level results to be compared directly, albeit with a loss in the stroke segmentation information provided in the stroke-level representation. The symLG representation allows us to identify errors as symbol classification errors, relationship classification errors, or structure errors.

3.4 Summary

We introduced the tools and materials we use to evaluate QD-GGA and LPGA. We have also presented a new technique that allows string and tree-based formula structure representations to be meaningfully compared at the level

of recognized symbols and relationships. Further, this permits fine-grained evaluation of recognition results at the individual symbol and relationship level, as well as at the expression level. Finally, we discussed how we used SymLG format in evaluating IM2TEX system and ICDAR 2019 CROHME + TFD competition, as this format is simple to understand, and provide useful global performance metrics and automated error analyses.

In the next Chapter, we discuss LPGA system and the noteworthy behaviours we observed while designing this system which later on inspired designing our more complex model QD-GGA.

Chapter 4

Line-Of-Sight Parsing with Graph-based Attention (LPGA)

In this Chapter, we briefly present a series of systems that are improved to shape our final design QD-GGA. These systems are an extension and generalization of the works done by Hu et al. [39] and Condon [15]. In Hu and Zanibbi's work, the math visual parsing problem is defined as converting an input graph computed on the expression image to a Symbol Layout Tree (SLT) as shown in Figure 4.1. Line-Of-Sight (LOS) graphs are chosen to represent input equations as they have a higher maximum formula tree expressivity than many other graph representations (e.g., Delaunay, geometric MST, k-NN for $k \in \{1 \dots 6\}$) for CROHME dataset, while also reducing the search space for parsing [38]. The features set in their work is a combination of geometric features for spatial relations and visual density features e.g., the modified shape context feature with Parzen window density estimation. These Parzen shape contexts are used for symbol segmentation, symbol classification and symbol layout analysis with Random Forest classifiers.

Later, Condon applies LOS parsing approach on typeset formula images [15] using the same set of features and classifiers with a different pre-processing methods in which typeset images are converted into trace points in contours.

These works are then extended and generalized by us using Convolutional

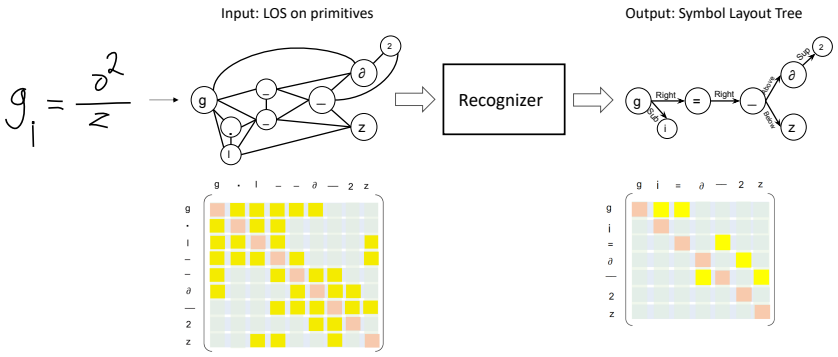


Figure 4.1: Input and output graphs for $g_i = \frac{\delta^2}{z}$. Adjacency matrices for stroke-level input and symbol-level output are shown.

Neural Nets (CNNs) for feature extraction. Line-Of-Sight Parsing with Graph-based Attention (LPGA) was our first attempt in using CNNs for graph parsing. Visual features for inputs are learned and extracted automatically by a two channel multi-layer convolutional neural network, using the standard VGG-16 architecture in each channel [79]. In [65], both Condon system [15], LPGA_{RF} and our modified system, LPGA_{CNN} is described. Throughout this document we refer to our improved system as LPGA for short.

4.1 LPGA

LPGA (Line-Of-Sight Parsing with Graph-based Attention) is a feed-forward approach that solves tasks sequentially similar to the work presented by Hu et al [39]. A hierarchical approach is used to first segment nodes into symbols with a Convolution Neural Net (CNN), and then generate a second LOS graph on symbols , see Figure 4.2. Two additional CNNs are trained to learn symbol classes and spatial relationships in the symbol-level graph [65].

Visual features for inputs are learned and extracted automatically by a two channel multi-layer CNN, using the standard VGG-16 architecture in each

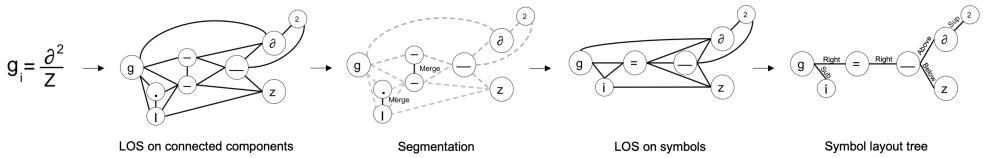


Figure 4.2: Hierarchical approach in LPGA. After segmentation, symbols are connected in a second LOS graph, and the spatial relationships represented by edges are classified. Edmonds' algorithm then selects a maximum spanning tree to produce a Symbol Layout Tree (SLT).

channel [79]. Figure 4.3 shows our CNN classifier for segmenting connected components into symbols, by classifying each edge in a LOS graph over connected components as 'merge' or 'split'. In Figure 4.3, the input is the edge between the two connected components in equal (=). We observed that it is beneficial to have separate channels for input and visual context (attention). Thus, for each target image, a parallel branch provides contextual information around the target for better decision making. The target image contains the merged bounding box of the parent-child pair cropped from the input image. The context image is another cropped image which is computed by scaling the merged box dimension with the context factor: $r_{context} = \alpha \times r_{target}$. The context factor of 1.75, 1.5 and 4.0 produced the best results for segmentation, parsing and classification respectively. The features from these two channels are concatenated before being passed through a softmax layer. For segmenting connected components and classifying spatial relationships, in addition to visual features the geometric features described above are added directly to the final feature vector before classification (see Figure 4.3). All features values are normalized between [-1,1].

4.2 System Design Experiments

In LPGA, for each task, an individual CNN is trained and each query (nodes or node pairs) is cropped from the input image and passed to the corresponding CNN. Following we are going to discuss some of the noteworthy observations in designing different modules of LPGA system.

- Context Module: aimed to study context in feature extraction.
- ALL vs ELSE in context channel.

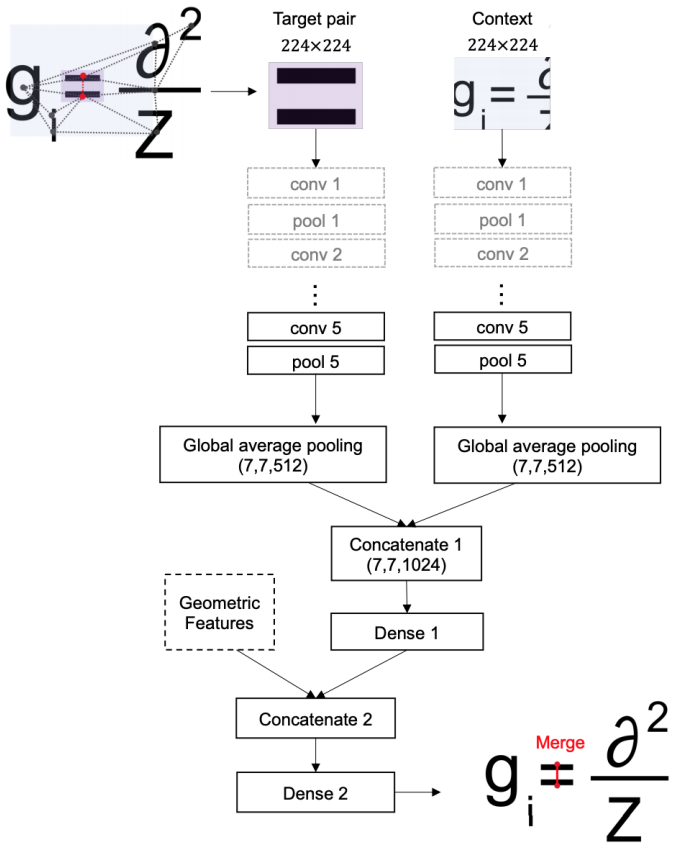


Figure 4.3: LPGA segmentation example. The red edge between connected components in the equals sign (=) is being classified for symbol segmentation. Two independent branches based on VGG-16 architecture represent the input (left side) and the attention context (right side). Geometric features are added before the final dense layers.

- Early Fusion vs Late Fusion in the main stream and the context stream.
- Multi-Resolution in Context Channel.
- Geometric Features.

A series of experiments were conducted to refine our CNN classification model for segmentation, symbol relationships, and symbols. For each set of experiments, parameters and structure designs which give the best result in a 5-fold cross validation were used for testing.

Context. These experiments aimed to study context in feature extraction. Experiments showed that having context is helpful in all three tasks. It is expected that contextual information will be beneficial for labelling edges, but it is also useful for symbol recognition. Adding context resolves class ambiguities for some visually similar classes as shown in Table 4.1. Symbol classification accuracy on the test set improves from 98.13% to 99.43% when having a second branch for context.

How much context should we use? Previous experiments by Condon in [15] suggest that increasing histogram size to capture more visual context from surrounding symbols improves classification accuracy. We did a grid search to optimize the radius of the context window in the second channel for each task. We find a radius factor (α), which is multiplied with original target radius and produce the context radius: $r_{\text{context}} = \alpha \times r_{\text{target}}$. A radius factor of 1.75, 1.5 and 4.0 produced the best results for segmentation, parsing and classification respectively. It is worth noting that the input includes two connected components or symbols for segmentation and parsing and only a single target symbol for classification (r_{target} is larger for the first two tasks); so the chosen radius for all three tasks provide similar contextual information.

ALL vs ELSE in context channel. The next set of experiments, studied whether we should keep the target in the context window (ALL) or just pass everything around it (ELSE). We hypothesize that ALL should perform better, as at some depth in the context channel the field of view will capture the target, and since the two channels are mutually trained, the network might learn the target shape. Hence, by not removing the target from the context channel, the model can potentially capture some relative positions. Experiments show insignificant difference between these two situations, e.g., less than 0.5% for segmentation, so we decided to pass the entire image region into the context branch.

Table 4.1: Effect of visual context attention on classification of similar symbols. Shown are the top-5 most frequent confusions for visually similar classes before using context, and then after.

		Errors	
Ground Truth	**Prediction**	**No Att.**	**w. Att.**
overline (e.g., $\overline{x}$)	minus	192	27
fractionline	overline	122	10
minus	overline	94	9
cdots ($\cdots$)	ldots ($\ldots$)	21	8
letter l	one (1)	18	10

Early Fusion vs Late Fusion. We also tested alternative ways of representing contextual information. We initially did source separation in the inputs by having the target in one channel, and context in the other channel (early fusion). This works well when using visual density features in LPGA_{RF}. Instead of having one visual density histogram, there are three histograms for parent, child, and context. In the convolutional layers of LPGA_{CNN}, the three channels for parent, child, and context will collapse into one 2D feature map once it convolves with the first kernel. The kernel moves along the height and width with the defined stride and its depth is the same as the inputs, e.g., in first layer kernels have depth of three if the model accepts RGB images. In other words, RGB channels of an input image are merged very early and basically no individual weight is trained for target and context individually as we hoped.

To have separate weights for the input and context, and in order to train both mutually, we decided to have a separate branch for context and merge the branches later in the pipeline. This allows the network to see the context while focusing more on the target. It worth mentioning that another advantage of late fusion is faster convergence during training.

We also explored a three-branch structure in which the parent and child were fed to the system separately and a third branch to capture context, but the performance deteriorates compared to having parent and child as target in one channel. That could happen due to the fact that we are freezing weights in the first four convolutional blocks in all three branches, and thereby not back

propagating through all the layers during training. Without this, training would not converge.

Multi-Resolution Context. Low resolution context improves the result when using visual density in LPGA_{RF}. That encouraged us to investigate different resolutions for the context channel. Experimental results showed that coarser context lowers the accuracy. We investigate three different resolution, 224 pixels height and width which is the default size for VGG16, 112×112 pixel resolution and 48×48. Decreasing the context input resolution in symbol classification task with a late fusion structure and context radius factor of 4 decreases the accuracy from 99.34(%) into 99.03(%) for halving the input size and 98.24(%) for the smallest inputs (48×48).

Geometric Features on Side Channel. For defining the relation between connected components in segmentation or symbols in parsing, relative position of parent and child plays a key role. We studied whether embedding the structural information extracted from the input graph into classification is useful. We concatenate the geometric features extracted from the node pairs in LOS graphs with the final feature vectors before passing them through the softmax layer. Feature vectors are normalized using a tanh activation function before concatenation meaning the last ReLu function is also replaced with tanh for domain adaptation. The result shows geometric features can improve the performance of relation classifiers, e.g.,from 98.69% to 99.34% for the segmentation task.

4.3 Motivation For Designing QD-GGA

Some of the limitations of feed-forward approaches are listed below: (1) Each of these networks can predict one query per input image. Therefore, for each expression having N nodes and E edges (N+E) target and (N+E) context images are generated and classified through corresponding networks to score the input graph. Our new design QD-GGA scores the input graph in one iteration as it is capable of handling multiple queries in a multi-task learning framework. (2) Errors in previous steps are inherited by the structural analysis as hard decisions are applied at each step in a hierarchical approach whereas integrated approaches, like QD-GGA, do not rely on previous steps, meaning that structural analysis interacts directly with symbol segmentation and symbol classification. (3) Moreover, learning these tasks separately on

isolated symbols and pair of symbols not only causes the lack of global attention and spatial information, but also is not efficient in terms of memory and speed. Thus, by sharing the features between the tasks, training the model end-to-end and also extracting the query features from a single input image rather cropping, centering and re-scaling each node and edge sub-image, we make the recognition much faster and computationally less expensive in our latest design: QD-GGA.

4.4 Summary

We have presented a graph-parsing approach for recognizing math formula images, in which we extract convolutional features using a simple attention model organized around a line-of-sight graph over connected components. Constraining the search space from pixels to nodes and edges of the attention graph appears to be effective (especially for scanned typeset expressions, see Results in Chapter 6), and we obtain strong results using relatively simple classifiers and maximum spanning tree extraction. The contextual features used for each of the recognition tasks helped in their classifications.

LPGA does all the recognition tasks in isolation with a predefined attention. A promising direction for improvement is training the classifiers jointly on all edges and nodes in expressions. This could solve the errors that happen locally by providing a global view of the entire expression. We used this insight and design as an integrated system which is discussed in the next Chapter.

Chapter 5

Methodology

Limitation of hierarchical approaches such as relying on previous steps and training multiple neural nets in isolation and success of global approaches (e.g., auto-encoders) in math recognition inspired us to design a new approach for LOS parsing which can be trained end-to-end and provide global information from the equation. We call this approach: Query-Driven Global Graph Attention (QD-GGA). QD-GGA predicts all node and edge classes at each feed-forward pass with the help of a query-driven attention mechanism.

QD-GGA is capable of Multi Task Learning (MTL) and predicting multiple queries at each iteration. Therefore, the three tasks in math recognition (segmentation, classification, and parsing) trained in isolation in previous models, can be trained simultaneously with weights get modified according to a joint loss counting all the errors in nodes end edges of the input LOS graph. In this design, CNN features are shared between tasks and extracted using SE-ResNext blocks which combines Squeeze-And-Excitation [37] with a ResNext block [91]. This grants an end-to-end training of all tasks and queries (nodes and edges) with traditional CNNs allowing for global feature-learning. In execution, before extracting a Maximum Spanning Tree (MST) on the scored LOS graph, segmentation results are applied converting the output primitive-level LOS into symbol-level LOS (see Figure 5.1).

The main strategy in both LPGA and QD-GGA is computing an input LOS graph from handwritten strokes or connected components in images. Then, class probabilities generated by the CNN classifiers are assigned to nodes and edges in the input graph, symbol segmentation are applied, and

then Edmond's arborescence algorithm [23] is used to extract a maximum spanning tree from the weighted graph over symbols. In the remainder of this Chapter, we overview the input graph computation in 5.1 which is similar in both LPGA and QD-GGA. We present the main modules and architecture of QD-GGA in Section 5.2. Finally, a summary of this Chapter is given in Section 5.3.

5.1 Graph Representation

For dependency parsing such as done in QD-GGA and LPGA, we need to identify a sub-graph with minimal cost or maximum probability. For math recognition, the final subgraph is usually a Symbol Layout Tree. Ideally, we would like to reduce the size of the search space by using a graph that only has edges between the primitives having a spatial relationship in the original setting (perfect precision) and avoid the miscellaneous edges which add confusion to the problem.

A study by Hu et al. [38] on parsing handwritten formulas proposes using Line of Sight (LOS) graphs [19], which represents whether nodes can 'see' one another. LOS graphs can represent roughly 98% of CROHME 2014 dataset formulas, while reducing the number of edges to $3.3n$ for n strokes, many fewer than the $n(n-1)/2$ edges in a complete graph.

In this work, we also use the Line of Sight (LOS) graphs over primitives. Figure 5.1 shows edges in a complete graph for $g_i = \frac{\delta^2}{z}$ and the remaining edges after computing the LOS graph. Nodes sharing an edge are marked with yellow in the adjacency matrix and the diagonal elements of the matrix shown in red are class labels of each primitives. The adjacency matrix $\mathbf{A}$ on the input graph shows nodes and edges such that the element a_{ij} represents the relationship between vertex i (parent) to vertex j (child). Given the n proposed object nodes, there are $O(n^2)$ possible connections between them; however, most object pairs are unlikely to have relationships. We use LOS graphs to prune many edges and reduce the hypothesis space of relationships, while preserving enough to ensure that valid expressions can be recovered most of the time. In our initial work, we use eight possible labels: Right, Superscript, Subscript, Above, Below, Inside, NoRelation and Merge for a_{ij} when node i can be 'seen' from the bounding box center of node j and zero when the line-of-sight between nodes i and j is blocked. We leave matrices and

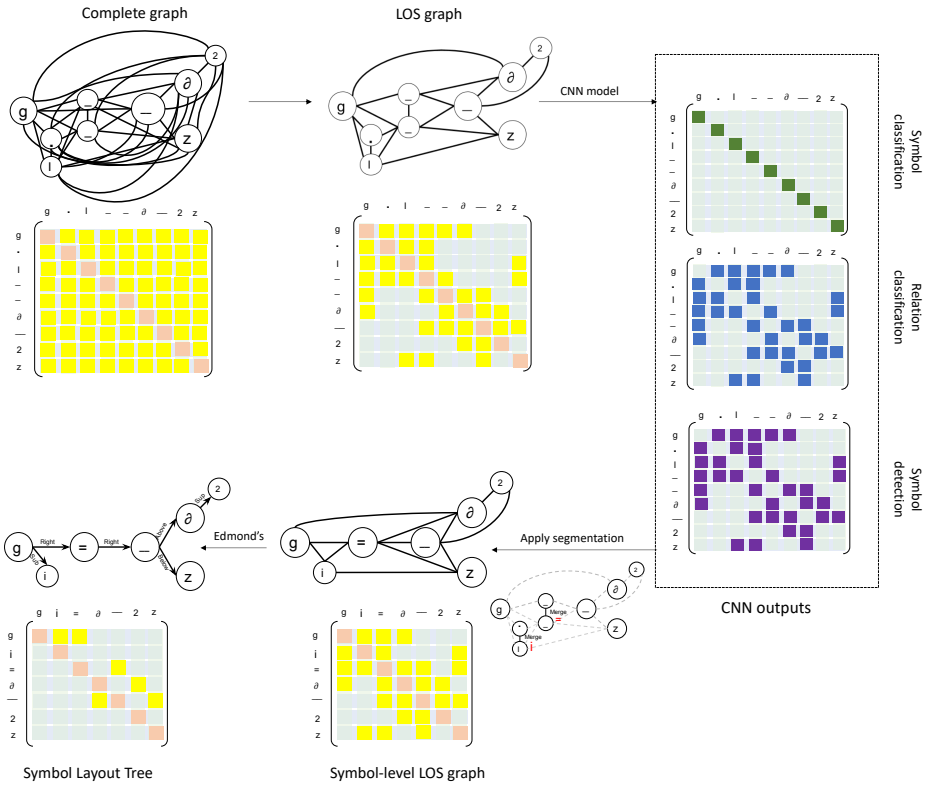


Figure 5.1: Parsing $g_i = \frac{\delta^2}{z}$. Graph edges are yellow adjacency matrix entries; nodes appear in red along diagonals. A primitive-level line-of-sight graph is constructed, after which nodes are classified as symbols, and edges between nodes classified for 1) symbol detection (merge/split), and 2) relation classification. Symbol detection decisions are applied to convert the graph into a symbol level graph, followed by averaging symbol (node) and relationship (edge) scores to obtain a new weighted graph. Finally, Edmond's arborescence [23] algorithm extracts a maximal symbol layout tree.

formula derivations as future work.

5.2 Query-Driven Global Graph Attention Model (QD-GGA)

QD-GGA utilizes a multi-task learning framework and has a graph-based attention trained along with the main branch by defining a joint loss. A joint classification loss is calculated from the output matrix, which back-propagates through all three classifiers responsible for segmentation, classification and relationships, as well as the attention layers, enabling us to train them simultaneously with shared features. QD-GGA model is made of four modules:

1. **Feature extractor:** stack of convolutional layers that takes the expression image and generate a feature map.

2. **Attention module:** a simple convolution block that takes binary spatial masks of queries (nodes and edges of input graph) and generates relevance maps. These relevance maps are downsized and multiplied with the input feature map and produce a distinct feature vector for each query from the single representation of the input expression.

3. **Context module:** A simple approach to expand the local context for queries by convolving each query feature with its immediate neighbors, previous and next queries, using a 1-by-3 filter, see Figure 5.4. This is done for node and edge streams separately.

4. **SLT generation:** Extracting the final tree from the weighted graph using Edmond's algorithm.

The mechanism we propose requires only a single pass through the input image to generate features, and it can be trained end-to-end. Our architecture is modular (each module can be updated individually), independent of the CNN feature model (any feature extractor such as VGG or ResNet might be used), easy to implement, and faster than recurrent approaches in training and execution (see Section to 6.7).

QD-GGA Architecture. The QD-GGA architecture is shown in Figure 5.2. The network contains a convolution block (shown in red) followed by two SE-ResNext block groups. SE-ResNext blocks combine Squeeze-And-Excitation [37] with a ResNext block [91]. To compute image features, each

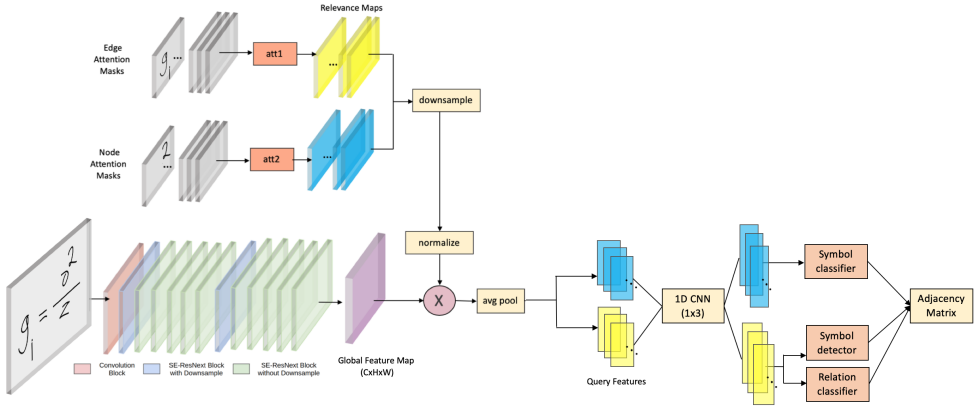


Figure 5.2: QD-GGA Architecture. The dimensions of the 2D features are channel, height and width. The dimensions of the linear features are batch size and channel. The batch size is N which is the number of binary masks or queries (all nodes and edges) to answer (shown in gray). The final adjacency matrix has class distributions for symbols on the diagonal, and two distributions for segmentation and relationship labels for each edge.

SE-ResNext group contains six SE-ResNext blocks with the first SE-ResNext block having a down sample layer (shown in blue).

In the following, we discuss the attention module (5.2.2, 5.2.3) that allows for multi-query learning from one input image. We explain how the final Symbol Layout Tree is extracted in Section 5.2.4 and finally provide details on implementation and training in Section 5.2.5.

5.2.1 Feature Extractor

This architecture is based on ResNet. A squeeze-and-excitation (SE) block is applied at the end of each non-identity branch of residual block. SE is an architectural unit designed to improve the representational power and enhance the quality of spatial encodings throughout the network's feature hierarchy by enabling it to perform dynamic channel-wise feature re-calibration. These units recalibrate channel-wise feature maps by explicitly modelling interdependencies between channels. This is achieved through an Squeeze operator: global average pooling and an Excitation operator: introducing nonlinearity with ReLU or tanh.

Receptive field calculations. The receptive field of a convolutional neural net is the key idea that most of the object recognition method is built around it. It is important to pay attention to receptive fields at each layer when designing a neural net. We care about the receptive field of our network as it defines how much context from the input image is encoded in the final feature map that create query features. Receptive field at each layer defines the region in the input image that a unit at that layer depends on [63]. Areas in the input image outside the receptive field of a unit does not affect the value of that unit. It should be noticed that not all pixels in a receptive field contribute equally to an output unit's response and pixels at the center of a receptive field have a much larger impact on an output [63]. The receptive size of our last layer in the network is 35 pixels which is calculated as explained below [22].

$$n_{out} = \left[\frac{n_{in}+2p-k}{s}\right] + 1 \tag{5.1}$$

This equation calculates the number of output features in which n_{in} is the number of input features, n_{out} is the number of output features, k is the convolution kernel size, p convolution padding size, and s convolution stride size. For simplicity, it is assumed that CNN input is symmetric and input image is square. In case of having non-square inputs, like our equation images, one can calculate the feature map attributes separately for each dimension. We did our calculation on the fixed symbol height size which is 64 pixels.

$$j_{out} = j_{in} \times s \tag{5.2}$$

Jump refers to the distance between two consecutive features. This calculates the jump in the output feature map (j_{out}), which is equal to the jump in the input map times the number of the stride size in the convolution showing how many input features are skipped when applying the convolution.

$$r_{out} = r_{in} + (k-1) \times j_{in} \tag{5.3}$$

Equation 5.3 calculates the **receptive field size** of the output feature map, which is equal to the receptive field size of the input feature map plus the area covered by k input features.

$$start_{out} = start_{in} + \left(\frac{k-1}{2} - p\right) \times j_{in} \tag{5.4}$$

Start is the center coordinate of the first feature. This equation calculates the center position of the receptive field at a certain layer, which is equal to the center position of the input feature plus the distance from location of the first input feature to the center of the first convolution minus the padding space. Note that jump of the input feature map is multiplied in both cases to provide the actual distance/space.

The first layer is the input image, which always has n = image size, $r = 1, j = 1$, and start = 0.5. Applying this on our architecture we get $n_{out} = 8$ for symbol height of 64, jump = 8, receptive size = 35, and start = 9.5. That implies each point in the final feature map corresponds to a 35×35 region in the input image, with the minimum symbol height set at 64 pixels. Considering in practice, the center part has the most contribution into later units, this receptive field is not large enough to capture the global context and we need to have a system that incorporate all nodes and edges when making decision for each query (global context).

5.2.2 Attention Module

The attention module queries a shared CNN feature map to efficiently obtain inputs for all classification, segmentation, and relationship decisions. As seen in Figure 5.2, there is a side branch consists of two attention layers which takes binary masks for nodes and edges separately as inputs and passes them through convolution layers. We performed extensive experimental analysis explained in Chapter 6 to understand the performance trade-offs amongst different combinations of shared and task-specific representations in the main and side stream. The best configuration has 3 convolutional blocks with each block having 4 kernels of size of 7×7, 5×5, and 5×5. The final relevance maps are 2D ($H \times W$) similar to input binary masks, see Figure 5.3.

Spatial masks provide attention, comprised of either individual node binary masks, or masks from pairs of nodes sharing an edge in the input graph. The attention module takes binary masks and then generates relevance maps (i.e., continuous masks) by convolving binary masks with kernels trained for each task (per [24]). Figure 5.3 shows binary masks and their corresponding relevance masks for node (single stroke) x and stroke pair $(x, 6)$. Relevance

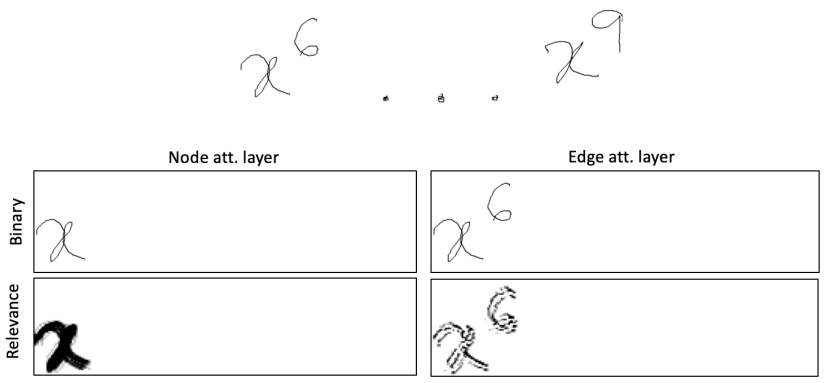


Figure 5.3: Attention Masks. Input binary masks for node x and edge $(x, 6)$ are shown in the first row. Corresponding relevance masks applied to features are shown in the second row.

maps are downsized and then multiplied with the global feature map. In this way, the downsampled and normalized relevance mask weights the feature map to focus on ('query') the relevant input region. Finally, the weighted feature map is average pooled to a one dimensional feature vector which can be regarded as the extracted feature for the primitive (stroke) or primitive pair (stroke pair).

We normalize query features after average pooling using the 1D Instance Norm [83] with $\epsilon = 0.001$. This converts values for each feature into a measure of standard deviations from the mean ($E[x]$) making each individual feature distribution look Gaussian.

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \tag{5.5}$$

5.2.3 Context

We explained the role of field of view in providing local context for query features in Section 5.2.1. To provide more context, given a sequence of primitive feature vectors (generated by attention module from the main feature map), the context module first concatenates the feature vectors along the length dimension of a 1D feature tensor. Then, the module performs a 1-by-3 convolution along the length dimension treating each primitive as an individual element for node stack and edge stack separately, similar to time series classifi-

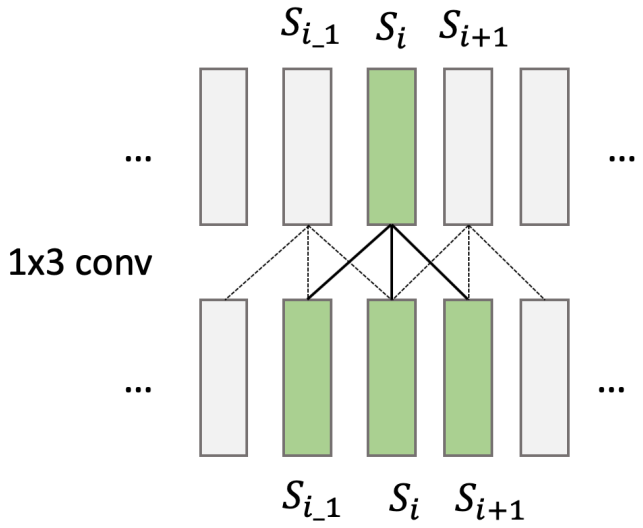


Figure 5.4: Visualizing 1-by-3 convolution in context module for stroke feature sequence.

cation with 1D-CNNs [27]. The convolution operation consolidates features of a primitive neighborhood by considering the $i-1-th$ and $i+1-th$ primitives for the $i-th$ primitive, see Figure 5.4. This simple way of adding context is clearly dependent on the sequence of queries fed to the system. The node stack is sorted based on the left to right order of strokes for online data and bounding box coordinates for the offline data (images). The edge stack is sorted based on the parent primitives with the most left one comes first and then for each parent the children are sorted the same way from left to right. This is our initial idea to test if more context would be helpful for segmentation, classification and parsing.

5.2.4 SLT Generation

We use Edmond's algorithm [23] to extract a Maximum Spanning Tree (MST) from class distributions associated with the LOS adjacency matrix output. Experiments demonstrate that it is more accurate to apply symbol segmentation results before extracting relationships (see Figure 5.5), rather than extract an MST directly from the stroke-level matrix.

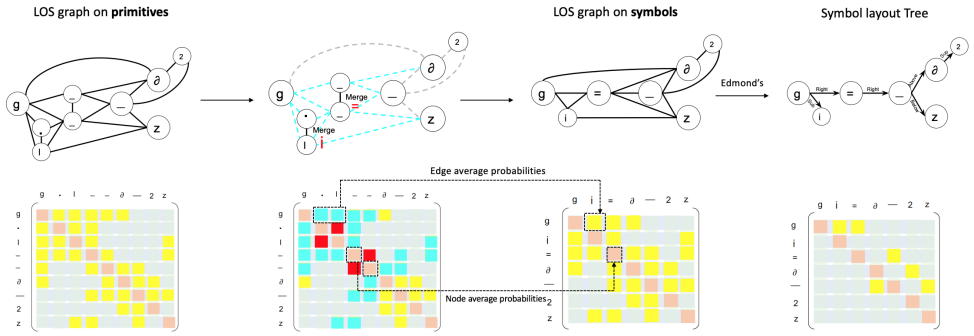


Figure 5.5: SLT Extraction. A primitive-level graph is converted to symbol-level after applying segmentation results, and then an SLT is extracted using Edmond's arborescence algorithm. Red patches show primitives to be merged into symbols. Blue patches show edges that should be updated after merging nodes. Merged probabilities for symbols and symbol relationships are averaged over constituent stroke-level elements.

Algorithm 1 provides the steps in converting a stroke-level graph to a symbol-level graph. First, primitives that belong to a single symbol are merged based on the segmenter predictions. Symbol class distributions are computed by averaging symbol classifier probabilities over all primitives belonging to a single symbol. Then, all incoming and outgoing edges attached to primitives grouped into a symbol are merged into one incoming and one outgoing edge. Again, probabilities over merged edges are averaged to generate the symbol-level edge probability distributions.

SLT generation is illustrated in Figure 5.5. In the example, primitives belong to i and $=$ are merged into symbols. All edges connected to these four primitives, shown with blue patches, should be updated in the symbol-level graph. Average probabilities for stroke-level edges provide the distributions for symbol-level merged edges.

5.2.5 Implementation and Training

Loss. The loss designed for an Multi-Task Learning (MTL) model should allow tasks contribute to training equally, without letting the easier tasks

Algorithm 1 SLT Extraction from Adjacency Matrix

1. Use 'Merge' edges to group primitives into symbols
2. Classify symbols by max. mean score for member primitives
3. Score symbol relationships by avg. stroke pair distributions
4. Apply Edmond's algorithm to obtain maximal SLT

dominate the learning. We use the cross entropy loss (CE) with a softmax layer to normalize the network outputs. The final loss is the sum of all the errors in segmenter, parser and symbol classifier. The loss function δ is defined in equation 5.6. Here N is the stroke set (nodes), and E is the set of line-of-sight edges in the adjacency matrix. D is the set of detection ground truth labels for edges, R is the set of relationship ground truth labels for edges, and S is the set of ground truth symbol labels for nodes.

$$\delta(N, E) = \sum_{e=1}^{|E|} (CE(e, D) + CE(e, R)) + \sum_{n=1}^{|N|} CE(n, S) \tag{5.6}$$

Tree Loss. To reduce the effect of edges not contributing to the final tree we introduce "tree loss," where only ground truth edges and false positive edges in the final SLT are counted in loss calculations. Therefore, only hard negatives mistaken for real relations in the output are include in the loss computation. We tried other loss designs, e.g., harmonic mean of individual loss types, weighted combinations of individual losses, loss defined on MST edges, but experiments showed that the tree loss and the linear combination loss (Eq. 5.6) work better. For our main experiments described in Chapter 6 the linear combination loss has been used (since this worked well, we did not consider more sophisticated alternatives).

Training process. Since math expressions have different sizes, we replace the conventional batch normalization layers with group normalization [90] for all the blocks, as this is more robust for small batch sizes. Group Normalization divides the channels (in feature maps) into groups and computes the mean and variance within each group for normalization making the computation independent of batch sizes, and its accuracy stable for a wide range of batch sizes.

The QD-GGA CNN has 13,854,478 parameters. We use an SGD (Stochastic Gradient Descent) optimizer to learn the parameters. The batch size was set to 1, momentum to 0.9 and the learning rate was initially set to 10^{-2},

and then decreased by a factor of 10 when the validation set accuracy stopped improving. The training was regularized by weight decay set to 0.004. The system is built using PyTorch and experiments were run on an 8GB Nvidia 1080 GPU. Experiments were run on a server with an Intel Xeon E5-2667 processor (3.20 GHz per core), and 512 GB RAM was available. The time complexity of our model is $O(|N| + |E|)$ for an input graph with E edges and N nodes. In the worst case (complete graph) $|E| = |N| \times (|N| - 1)$. It should be mentioned that for tree extraction with Edmonds' algorithm, the complexity is $O(N^3)$ in the worst case [23]. Training our network does not include the tree extraction, but we use Edmonds' algorithm in execution.

Recurrent Training. In QD-GGA, classifiers are trained with primitive level inputs. For instance, the symbol classifier is trained with the visual information extracted from the strokes making an 'x' separately rather that seeing the whole symbol at once. This is due to the fact that we would like to train all tasks simultaneously without allowing the segmentation to be applied earlier as in traditional hierarchical approaches.

In order to exploit the symbol level information, inspired by hierarchical approaches, we proposed a multi-step training in which the segmentation predictions in the first step are used to update the attention masks, hence generate new query features. This technique keeps input/output space in primitive level, but provide symbol level attention and features. This model has the ability to calculate the loss based on what system predicted as symbols at each iteration without changing the input space. We call this a recurrent training method. Operating this in 2-steps has the obvious benefit of converting primitive attention masks into symbol attention masks as shown in Figure 5.6, but one can do that for more than two step recursively. We experimented with two and three steps (see Chapter 6).

The equation image goes into the main branch and the binary stroke (N) and pair (E) masks go into the attention layers similar to the previous designs making the input dimension: $H \times W \times (1 + N + E)$. The input image generates the global feature map ($(H/8, W/8, 512)$), shown in blue in Figure 5.6, and the stroke masks output the relevance maps of the same size (H, W, N) in the stroke attention layer. Pair masks also go through the similar process in the edge attention layer. Next, the relevance maps are resized into the shape of the main feature map $(H/8, W/8)$. The resized maps generate the query features by multiplying with the main feature map, shown in pink. We stack node

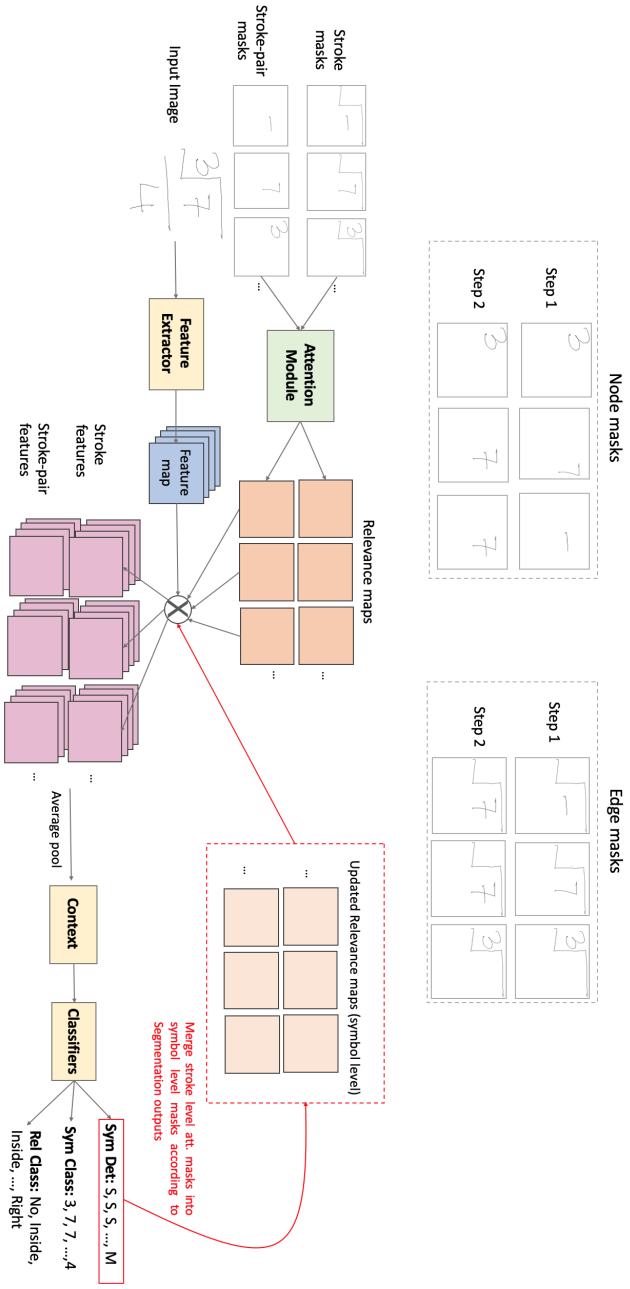


Figure 5.6: QD-GGA recurrent mechanism. Segmentation results at each iteration can be used to update stroke-level attention masks into symbol-level attention masks. These updated masks can query new features from the global feature map. The red arrows shows the operations added at the end of the network that can be repeated recursively (see Algorithm 2). The final predictions are used to calculate the loss.

Algorithm 2 Pseudo code for n-step training approach ($n \geq 1$).

$i = 0$
$sym_{ref}, seg_{ref}, rel_{ref} = \text{Net}(images)$
while $i < n$ and $Merge$ in seg_{ref} **do**
  $updated\ images = \text{UpdateMasks}(images,\ seg_{ref})$
  $sym_{new}, seg_{new}, rel_{new} = \text{Net}(updated\ images)$
  $sym_{ref}, seg_{ref}, rel_{ref} = sym_{new}, seg_{new}, rel_{new}$
  $i = i + 1$
end while
loss = $\text{CE}(sym_{ref}, sym) + \text{CE}(rel_{ref}, rel) + \text{CE}(seg_{ref}, seg)$
loss.backward

and edge features separately and make sure they are keeping their incoming sequence as we want them to correspond to the ground truth labels. Each stream goes into a 1-by-3 convolutional layer, to add local context for each target query. Edge stack goes into the segmenter and parser, node stack goes into the symbol classifier. We refer to the predictions at this step as First predictions. At this step, instead of computing the loss and do a backward propagation, we apply the segmentation results and update the input masks.

To update masks, first we group the primitives that belong to the same symbol, and then for all those primitives we update their input binary mask to be a symbol level mask containing all the primitives in the group, see the examples of masks in first and second steps in Figure 5.6. These new masks are generated by getting the maximum of all the primitive masks. So if in the input equation we have a 7 made of two strokes and the segmenter in the first step predicts they have to be merged into one symbol, both stroke masks will be updated to be the maximum value of stroke masks belong into a symbol. For pair masks, we use the updated masks for parent stroke and child stroke: Sym Pair Masks = max(updated parent , updated child). These updated masks and the original image make a new input ($H \times W \times (1 + N + E)$) which is passed through the model again to produce the second step predictions. This time the predictions are used to compute the loss.

Algorithm 2 describes the recurrent training in which n can be a value above 1 for multi-step approach training. In the first step ($i = 0$), we get our reference predictions from the network. If $n = 0$, then the loss will be

computed with these predictions. But if $n \geq 1$ and there are Merge cases in the reference segmentation, these predictions are used to update the masks and compute new predictions. The reference prediction are updated as they are going to be used for loss calculation and the next step operations.

5.3 Summary

In our proposed approach, nodes and edges in the input graph get scored at each iteration with probabilities generated by a CNN-based model allowing to extract a directed tree from the weighted graph. The class probabilities generated by the CNN classifiers are assigned to nodes and edges as weights. In LPGA, vgg16 is used for feature extraction and classification in a hierarchical approach while QD-GGA utilizes SE-ResNext architecture and is equipped with an attention module allowing for classifying all nodes and edges in LOS graph in a single iteration.

We introduced recurrent training in which the primitive masks that filter the feature map will be updated to symbol level masks, for both nodes and edges, per segmenter decisions at each iteration.

In the next Chapter, we overview the experiments designed for QD-GGA approach and benchmark our system for both handwritten and typeset recognition.

Chapter 6

QD-GGA Experiments

This chapter overviews the design experiments in QD-GGA, including Graph Representation, Attention Module, Feature Extraction, Context Module, CNN Classifiers, and Tree Extraction. These experiments are designed to answer the research questions we would like to study in support of our thesis hypothesis: fast and accurate recognition of typeset and handwritten mathematical formulas can be obtained by an end-to-end CNN-based model equipped with query-driven global graph attention. Table 6.1 shows the mapping between some of these research questions and experiments and Table 6.2 summarizes the experiments that are presented in this chapter and provide a brief description for each.

We group experiments based on the research question they are designed to answer and present the results on CROHME 2019 test set [68]. Results are compiled using the LgEval library [70] created for the CROHME competitions. We report recognition rates for formulas indicating how many of equations recognized with zero errors (in symbols and relations) given all the equations in the test set. **F-scores** are reported for detection and classification of symbols and relationships. Results for correct symbol/relationship locations (*Detection*), correct symbol/relationship detections and classes (*Det.+Class*), unlabeled SLT structure, and SLT structure with correct labels (*Str.+Class*) are presented for each experiment to better understand the behavior. Please note that symbol and relationship detection results are reported across all formulas in CROHME, while formula recognition rates are reported for complete formulas (i.e., input files) in these Tables. For better comparisons, Table

Table 6.1: Summary of the main experiments designed to answer research questions arises in support of the thesis statement.

Research Questions	Experiments
Input Graph	Graph representation for training (e.g., Complete vs LOS)
Attention & Context	Attention module (soft vs hard) Global context
Features & Tasks	Directed edge feat. Multi-scale feat. Task-specific vs shares feat. Tasks design
Generalization	Typeset math recognition (e.g., Infty)

Table 6.2: Description of the experiments designed to answer the research questions supporting the thesis statement.

Experiments	Description
Graph Representation	studying input graphs (e.g., Complete vs LOS).
Attention Module	Studying the best way to query features at each iteration.
Features	Studying the context and resolutions of shared features for different tasks.
Classifiers	Defining the tasks we want to train in the MTL framework.
Context	Exploring different approaches for including global context.
Tree Extraction	Methods on extracting the final tree from the Adj. matrix.
Recursive Training	A new method for updating primitive masks into symbol masks (local context).
Typeset Formulas	Evaluation Qd-GGA on typeset math recognition datasets (e.g., Infty).
Isolated Symbol Classification	studying the symbol classification tasks from primitives.
LPGA Design Experiments	Listing important observation in designing the baseline system.

6.3 presents all design experiments. Please note these rates cannot be used

directly for comparing performance across the table as the experiments were conducted under different configurations rather to understand what method works better under each sub-experiment and to discuss the logic behind our design. Detailed information on configuration of each experiment are provided in their corresponding sections.

Finally, we benchmark our system against state of the arts in Section 6.7 for both CROHME and INFTY datasets and suggest a plan for further investigation in the future based on the results and error analysis.

6.1 Input/Output

These experiments concern the input graph and the output tree representations.

6.1.1 Graph Representations

We would like to study whether we can improve structure learning from adjacency matrices over primitives by incorporating parse results directly in a modern CNN learning framework. We propose to define input formulas by graphs. Score nodes and edges of the input graph with CNN classifiers and extract the maximum spanning tree from the scored directed graph. Thus, the first step in our approach is computing a graph on formula primitives.

This sets of experiments aimed to study the input graph representations. Experiments designed for this purpose should essentially investigate the followings: Do we need to have all the elements in the adjacency matrix scored to be able to extract a valid tree? This means all the possible relationships between the primitives should be labelled (complete graph). Can we prune miscellaneous relations and reduce the input queries to CNN classifiers without sacrificing the performance?

In the previous studies on handwritten math recognition, [38] Line-Of-Sight (LOS) graph has shown promising results. Results on our baseline model using LOS graphs [65] support this observation. However, error analysis in both models shows that the most frequent relation classification errors are due to missed edges in the LOS graph, e.g., a long superscript may block the field of view of two symbols with a connection and cause a missing ground-truth edge in LOS graphs.

To address this, we experimented with complete graphs. Table 6.3 shows our results. We believe LOS graphs are better representations since the complete graph contains more edges, resulting in more variation in features and a larger search space for spatial relationships. It is intuitive that having a larger search space makes the decision making harder, whereas applying some pre-filtering method to reduce the search space, like pruning the miscellanies edges with LOS, is making the task easier for classifiers. In some of the spatial parsing studies, a similar idea implemented with a side low-cost CNN in [93]. This small network is responsible for removing the miscellaneous edges and pass a smaller subset for classification to main network which support our observation. According to [38] LOS recall rate on CROHME2014 dataset is 98%. In the future, one might make this rate improved by training a small network for it. We stick to LOS in this study as it is faster to compute and its behavior is predictable.

Another case to consider is using only the edges given in the ground truth for training. This way "NoRelation" will be removed from the sets of ground truth relations as all the samples in training set are a valid spatial relation. Table 6.3 shows the results. The test results shows the best input graph representation is still LOS. Training the system with only positive examples from the ground truth tree cause the model to perform poorly on "NoRelation" edges in execution time.

In future, another interesting idea to explore for the input representation is computing undirected graphs rather than directed edges we studied so far. In this case, the final output would also be an undirected tree and some rules must be applied to find the parent-child relations. This will decrease the input edges to half as we only need to label either upper or lower triangle in the adjacency matrix (allowing to work with complete graphs). Also, we would have one edge per primitives pair, so we do not need to concatenate the pair masks with the parent mask to introduce a visual difference for different directions of an edge.

The results shown for these experiments are computed on the CROHME 2019 test set using a CNN that has three classifiers for symbol classification, symbol detection, and relation classification. For edge features, parent mask in each pair is concatenated with the edge mask.

Table 6.3: Results of experiments studying QD-GGA modules computed on the CROHME 2019 test set. F-scores are reported for Symbols and Relationships. The highest rates are shown in bold for each task. Our best system achieves 37.38 recognition rate which can be boosted to 38.81 by transfer learning on INFTY.

	Symbols		**Relationships**		**Formulas**		**Config***
	Detection	Det.+Class	Detection	Det.+Class	Structure	Str.+Class	
Input Graph							
Complete graph	97.89	87.27	84.41	83.21	46.10	24.81	B-00-2-3-p
LOS graph	97.81	87.35	89.55	88.16	58.84	31.35	B-00-2-3-p
GT tree	95.75	80.94	35.17	34.92	19.72	14.27	B-00-2-3-p
Output Tree							
Primitive LOS	82.21	73.68	41.78	40.83	14.53	8.24	B-00-2-2-Np
Symbol LOS	73.38	66.18	51.25	50.13	24.06	13.5	B-00-2-2-Np
Classifiers							
2 class.	73.38	66.18	51.25	50.13	24.06	13.5	B-00-2-2-Np
2 class. + p	82.21	74.54	66.99	66.19	40.15	23.30	B-00-2-2-p
3 class. + p	97.81	87.35	89.55	88.16	58.84	31.35	B-00-2-3-p
4 class. + p	97.94	87.01	90.02	88.66	59.09	29.92	B-00-2-4-p
Att. Module							
Binary Masks	97.81	87.35	89.55	88.16	58.84	31.35	B-00-2-3-p
1block-1kernel	98.49	89.00	91.35	90.03	62.45	35.21	T-11-2-3-p
1block-4kernel	98.70	89.13	92.23	90.73	63.12	35.21	T-14-2-3-p
1block-8kernel	98.06	86.55	88.84	87.11	58.68	31.01	T-18-2-3-p
2block-1kernel	98.49	89.00	91.35	90.03	62.45	35.21	T-21-2-3-p
3block-1kernel	98.61	89.44	92.14	90.58	63.87	35.21	T-31-2-3-p
3block-4kernel	97.43	88.72	91.18	89.83	62.53	36.13	T-34-2-3-p
Att. Layers							
2Branch	97.43	88.72	91.18	89.83	62.53	36.13	T-34-2-3-p
3Branch	97.73	89.10	90.74	89.27	61.94	36.88	T-34-3-3-p
Query Feat.							
Task-specific feat.	98.44	88.38	**93.12**	**91.7**	**67.39**	35.37	T-11-2-3-p
Multi-scale feat.	95.59	87.10	87.46	85.81	55.49	33.19	T-34-2-3-p
Rel-specific feat.	97.42	88.77	92.57	91.09	66.81	**37.38**	T-34-3-3-p
Transfer learning	97.30	89.09	92.19	90.76	65.55	**38.81**	T-34-3-3-p
Geometric feat.	96.67	87.93	90.84	89.46	63.87	34.12	T-34-3-3-p
Context							
Opposite Masks	83.68	48.35	40.79	37.89	10.39	7.89	B-00-2-3-p
Recurrent train							
2-step training	99.44	**89.90**	92.20	90.51	64.96	36.13	T-31-2-3-p
Adj. Matrix Class.	**99.45**	87.85	92.83	91.07	65.63	32.44	T-31-2-3-p
3-step training	99.36	89.19	92.28	91.10	66.31	36.21	T-31-2-3-p

Config: a string is used to encode the network configuration for each experiment which is described in Table 6.4.

Table 6.4: The **config** strings in Table 6.3 are made of five parts described below.

	Description
Attention	
B/T	If the attention masks are binary (B) or trainable (T)
NN	The number of convolution blocks and kernels in the attention module
N	The number of layers in the attention module (2 or 3)
Classifiers	
N	The number of output classifiers
Parent Feat.	
p/Np	If the edge features are concatenated with parent feature (p) or not (Np)

6.1.2 Tree Extraction

Experiments depict extracting a tree directly from primitive-level LOS is not providing a consistent tree all the time. Using Edmond's algorithm for MST extraction does not guarantee that all the Merge edges would get chosen for the final tree (Table 6.3). This results in missing relations in the final tree. So we need an extra step to take care of this by either (1) finding nodes that have a "Merge" relation to the nodes of the final tree and group them or (2) applying segmentation prediction results on the primitive-level adjacency matrix and convert it to symbol level as shown in Figure 5.5. We employ the later to ensure the output trees are consistent. The edges connected to those primitives merged into symbols in the new graph need to be updated. The scores of the updated edges would be the average of old scores.

Table 6.3 shows the results of extracting a tree directly from adjacency matrices over primitives and extracting a tree from a converted symbol level adjacency matrix. As we expected, extracting a tree directly from primitive level matrix does not always generate a valid output and causes a lower recognition rate. Results here are reported on CROHME 2019 test set using a two classifier CNN taking nodes and edges from earlier designs of QD-GGA.

6.2 CNN classifiers

In LPGA, we trained three classifiers to do segmentation, symbol classification, and symbol parsing. In QD-GGA, we do not have a separate step to first

segment input primitives to symbols as in traditional hierarchical approaches. Therefore, in the initial designs, we had two classifiers one responsible for classifying nodes and the other one for classifying edges, with "Merge" and "NoRelation" be in the same relation pool. Results are shown in the third section of Table 6.3.

In this set of experiments, we explore separating decisions by assigning new classifiers to them. In the first experiment, we add a third classifier responsible for symbol detection (segmentation). It is intuitive that merge/split decisions are easier to make in comparison to classifying spatial relations altogether. To do that, the edges once are fed to a segmenter for a binary classification and also fed to a parser for relation classification. Separating the "Merge" relation from spatial relations and training a separate classifier that only collects information regarding this decision improves the segmentation results as shown in Table 6.3. The three classifiers we have in this configuration are (1) detecting symbols with a segmenter (2) classifying symbols (3) classifying relations for all LOS edges. We include "NoRelation" labels in our set trying to detect true spatial relationships from miscellaneous edges.

Observing the improvement in the previous experiment leads us into assigning a fourth detector for the binary task of detecting true relations (Edge/NoEdge). We hypothesize that it would be beneficial as it breaks the process of finding relations into two steps: detecting the correct edges and classify them, similar to what we were already doing in node classification. So the four tasks to solve in this experiment would be: (1) detecting symbols (2) classifying symbols (3) detecting relations (4) classifying relations.

Table 6.3 shows the result on dividing the main two tasks (node classification and edge classification) into more specific tasks using new classifiers. By applying the relation detection results before SLT generation similar to what we do for nodes (first applying segmentation results), the recognition rates dropped to 29% and no improvement was observed regrading relation detection and classification. We hypothesize this behaviour could be a result of our Symbol Layout Tree (SLT) extraction approach. Applying the relation detector decisions to prune early and not considering edges that are detected as miscellaneous in the SLT generation step inserts the binary rel detector errors directly to the results. We decided to not eliminate any of the edges before tree extraction and keep the fourth detector to introduce a better separation of features while training. Last row of Classifiers section in Table 6.3

presents the results of this approach.

Separating segmentation task by introducing a binary classifier to collect information regarding "Merge" or "Split" of edges improves the performance, substantially. However, dividing the relation classification into two steps in the same manner, did not help the model performance. If the binary rel detector results are not applied directly, the relation detection and structure learning slightly improve. We choose the three classifier configuration for future experiments as the overall performance is better. The results shown below are computed on CROHME 2019 test set. It should be mentioned that the system used for evaluation was equipped with a binary attention module.

6.3 Features

We propose a multi-task framework which learns to predict nodes and edges in one feed-forward pass. We would like to study if the proposed design with shared features between tasks improves the learning and makes it more accurate than individually trained detectors (while also making it faster)? To answer this, we discuss three key aspects in designing a multi-task-learning model: Attention 6.3.1, Query Features 6.3.2, and Context 6.3.3. The following presents the experiments we conduct to design each module.

6.3.1 Attention Module

Attention module produces feature vectors for nodes and edges (query features) from a single global feature map generated from the input formula image. To do that, we first use the graph information and generate spatial binary masks for all nodes and edges. Node masks have a single stroke and edge masks have the two nodes sharing an edge on them. In the preliminary design, the attention module takes the global feature map and the query binary mask as inputs and do an element-wise multiplication generating a new feature map with zeros everywhere except for the target query. In this method, the downsampled and normalized binary mask weights the feature map to focus on the relevant ('query') input region. This filtered out feature maps will then be used to generate query features. This is called a hard attention approach. Results of this experiment is shown in the first row of At. Module section in Table 6.3.

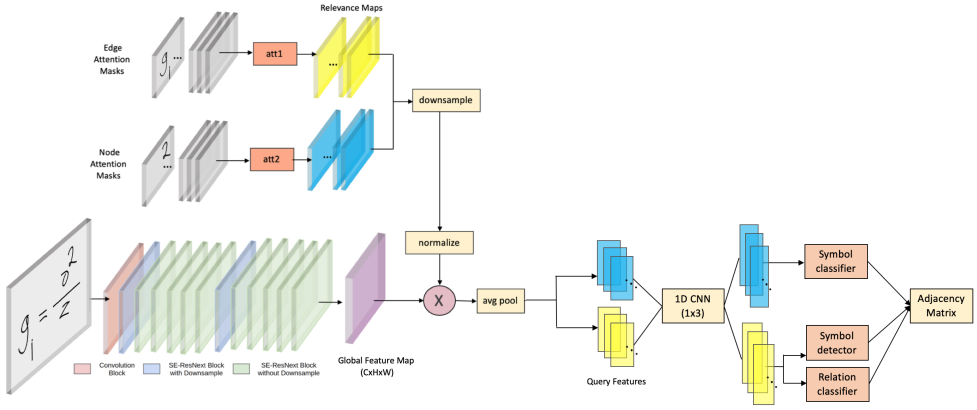


Figure 6.1: QD-GGA Architecture with two layers in the attention module taking node and edge masks separately.

We try both static binary masks (hard attention) and trainable attention masks (soft attention). Trained masks are more flexible and can learn where else to look in the feature map in order to gather more information for each query as they are trained end-to-end with the main branch in the QD-GGA.

Soft attention module contains two convolutional branch; one taking node masks and the other one taking edge masks. Each binary mask goes into the corresponding branch to get convolved and generates a relevance map as shown in Figure 6.1. This approach is inspired by [24]. Relevance maps are the convolved binary masks coming out of the attention layers which are later downsized to the shape of the global feature map. The downsized relevance maps are applied to the final feature map and the resulting query features are fed into their corresponding classifiers.

To improve the relevance maps contribution, we try different convolutional blocks and kernels in attention layers. After experimenting with different combinations shown in Table 6.3, we observed that having more than 3 convolution layers and 4 kernels per layer is not beneficial, so we use this as the best configuration in the attention layers. It should be mentioned that F-measures are calculated on CROHME 2019 test dataset using a CNN model with three output classifiers. Attention module contained of two identical layers that learn weights for nodes and edges separately.

As we generate three sets of features for symbol detector, symbol classifier,

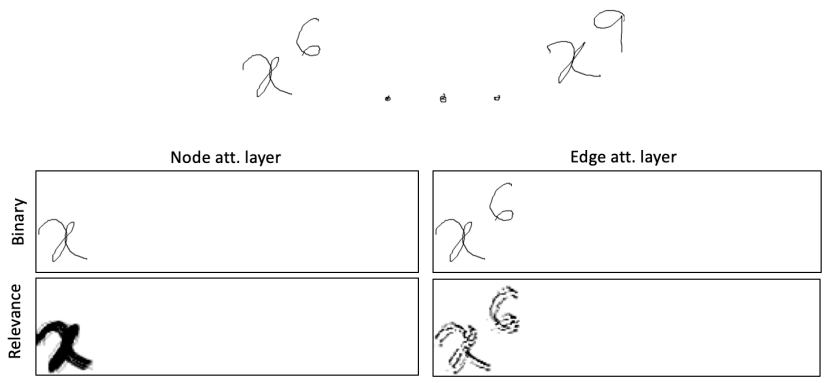


Figure 6.2: Attention Masks. Input binary masks for node x and edge $(x, 6)$ are shown in the first row. Corresponding relevance masks applied to features are shown in the second row.

and relation classifier, we decided to treat their input binary masks separately by adding another layer to the attention module. In this new configuration, weights in attention layers are not shared between the tasks. results are shown in Table 6.3 under Att. Layers. Each branch has the best convolutional configuration from earlier experiments regarding attention module (3 blocks and 4 kernels).

It is interesting that with only training one kernel (7×7) per binary mask, shown in second row of Att. Module section in Table 6.3, formula recognition rates are boosted by almost 4%, and symbol classification and relationship detection both are improved. Additional experiments reported in Att. Module section of this table show that the attention module with three convolution blocks provides the best configuration for attention layers, and stacking more blocks does not improve the performance.

Relevance Maps. We visualized the relevance maps to study what attention layers learn to query node and edge features better. Figure 6.2 shows binary masks and their corresponding relevance masks for node (single stroke) x and stroke pair $(x, 6)$. We observed the kernels trained for nodes are acting like a dilatation operation and the ones trained for edges are operating similar to contouring. We hypothesize this might happen as it tries to reach for far edges to extract more information on the two target nodes that appear at each

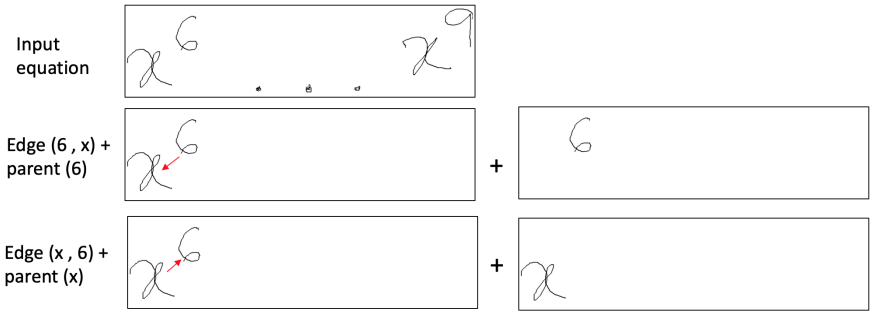


Figure 6.3: Concatenating parent features with edge pairs to introduce visual difference for directed edges shown with their corresponding binary masks.

edge mask.

6.3.2 Query Features

In the first setup, we did not differentiate features generated for directed edges between nodes (i.e., visual features for (A, B) matched those for (B, A)). Since features were identical for both edge directions, we decided to add features to signal which node is the parent for each edge, by concatenating the stroke pair masked features with the parent stroke masked features when classifying edges. Figure 6.3 shows that each feature vector for edges is concatenated with the features generated by parent mask. The feature vector for edge classification then becomes 1024 elements, while the remaining 512 elements become for the node classifier. This makes the parsing more accurate as seen in the first two rows of Classifiers section in Table 6.3. In this experiment the system had 2 classifiers (early designs). P in the table shows that the parent features are added for edge features.

Task-Specific Features. The query features are all extracted from the main feature map of the input image and all three classifiers use the same feature set originally. This means features are shared between tasks and no task-specific features are developed. In the next experiment, we designed an additional convolutional block for each task that takes query features separately. This allows for kernels in the final convolutional block to extract additional task-specific features at top level. Results in Table 6.3 - Query Feat. section show these features are mostly helpful with relation detection and classification tasks. We hypothesize having a larger field of view (context) provided by the new layer is more informative in case of edge features. By

going deeper, resolution is reduced and receptive field is increased. In symbol classification, resolution is important in learning the shape of symbols, therefore the additional block with lower resolution feature maps did not improve symbol classification. On the other hand, when deciding the spatial relations the shape of each node is not important rather their relative positions. It should be mentioned that we had trainable attention modules (3block-4kernel) made of two branches in this configuration.

We used this insight and designed another experiment extracting relation-specific features. In this design, the edge features go into an extra convolution block. This is the strongest result we got on CROHME 2019 dataset. The model is equipped with a three-branch attention module.

Multi-Scale Features. To find the correct balance of resolution and context in shared feature for different tasks, we implemented the multi-scale feature extraction approach. We hoped that the features from earlier layers (higher resolution) will contribute more in symbol classification and features from the last layer (larger context) will contribute more on relation detection.

We extract the features after each layer; the first feature map is extracted after the first convolutional layer and the other two are extracted after each SE group (total of three extracted feature maps). The output feature map dimensions are: (128, h/2, w/2), (256, h/4, w/4), (512, h/8, w/8) with h and w be the height and width of the input image. Masks from attention layers are resized to the shape of each feature map and mask the target in three scales.

There are mainly two ways to merge different feature maps together: concatenation and element-wise summation. Element-wise summation requires feature maps to have the same dimensions which means we have to convert the feature maps to have the same number of channels. Also according to FSSD [55] concatenation can get a better result than element-wise summation. So we use concatenation to combine the features. The features from different layers are concatenated and passed through the classifiers. The details of implementation is explained below:

1. Apply 1×1 convolution to reduce the feature dimensions.

2. Choose a fixed size for features to upsample (bilinear interpolation) and downsize (pooling) feature maps to it. We choose 256 (from the middle layer).

3. Apply batch normalization as features from different scales have different ranges (0.7% mAP improvement in FSSD [55]).

In Section 6.5, we observed that having the temporal layer in this configuration decreases the recognition rate by 5% for isolated symbol classification experiments, whereas removing it provides the similar performance. This means the context provided by multi-scale feature extraction can replace the temporal layers. So we removed the temporal layers meaning no additional module adds more context to the features.

As shown in results table, we could not prove this is the case for all three tasks trained simultaneously. This experiment showed that although having the multi-scale features without temporal layers could get the same symbol classification rate in isolation, the overall formula recognition rates is not as good.

Geometrical Features. Although visual features are helpful, geometrical features play a key role in parsing and structure learning. In this experiment, we propose to extract geometrical features for edges using bounding box information of pairs. We use the same set of geometric features used in [39]. These features include distance measures, area overlaps and differences, size ratios, and angles, distances and differences based on the bounding boxes around each stroke/CC. These distances include: distance between center points, difference in vertical position of bounding box tops and bottoms, difference in horizontal positions of left and right edges, difference in area, and amount of overlapping area. These spatial features are normalize and concatenated to the visual features before going into the classifiers.

Results show that adding geometric features for edges is not informative in QD-GGA design, especially for segmentation. The lower segmentation F-score (96.67) causes lower relation and symbol classification rates. This might be due these features are designed carefully for a different processing in which each target pair is cropped, centered and scaled into a fixed size so distance measures can be compared meaningfully across all samples, whereas, we avoid such preprocessing in our system.

Transfer Learning. In another experiment, we used INFTY dataset (scanned typeset formulas) and pre-processed images differently to be used as additional data in handwritten formula recognition. We extracted the skeleton of each character as shown in Figure 6.4. The model pre-trained on these samples and re-trained for CROHME. This improved the recognition rate to

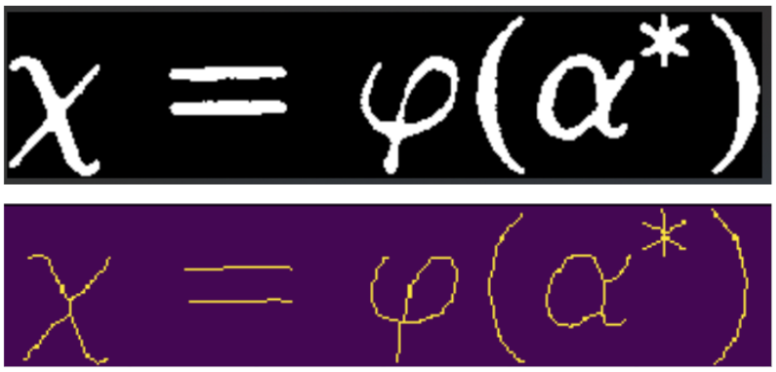


Figure 6.4: Extracting skeleton of typeset formulas in INFTY to pre-train a model for handwritten recognition.

38.81 % which is the best results we got so far. This should be noted that the most improvement is in symbol recognition.

6.3.3 Context

These experiments aimed to study local and global context in feature extraction. Having context always shown to improve the performance [15, 24, 38, 65]. Experiments in the baseline approach (LPGA model) showed that having context is helpful in all three tasks [65]. Context in symbol classification helps networks to differentiate between characters that are visually very similar, e.g., minus and hyphen. In this work, although all nodes and edges are predicted together and weights of the network get updated based on a joint loss introducing a global view to the network, context is not included in feature extraction directly. Using binary/relevance masks to generate the query features from the main global feature map is basically removing everything else except for the target.

Local Context. It should be mentioned that each pixel in the final feature map is looking at a larger area in the input image (receptive field) providing some context for the query features. We also apply a 1×3 convolution, to convolve each query feature with its immediate neighbors providing more local context which improved the results substantially. Moreover, in Section 5.2.5, we explain how 2-step training can provide even more local context by

updating initial stroke/cc masks into symbol level masks.

Global Context. We would like to provide a simple global context for each query feature. In this experiment, we generate a negative binary mask (opposite masks) for each target binary mask and multiply the main feature map with both of these templates and finally concatenate the results and feed it to the classifiers. As shown in Table 6.3, including context in this way is not helpful for our model and adds more confusion, especially in symbol classification, as the context masks are cluttered. This approach is mostly helpful for tasks containing one class label per image, meaning the negative mask will provide back ground context for one target in the image. For the case of labelling nodes and edges, the negative masks are very similar to the target itself as it contains other nodes and edges in the expression. We explored some variation of the same idea, e.g., weighting feature maps generated from the target masks differently than the ones from the context masks, but again the general idea is not suited for multiple target images and results in crowded background masks.

In another attempt to include the global context given by the input graph for individual queries, we decided to embed the whole adjacency matrix for all three classifiers. We classify all queries for all classifiers meaning the symbol detector and relation detector that previously was fed with edge features, will take node features as well and similarly the symbol classifier that previously would label the node features, will take edge features as well. In this design, the parent label is the desired predicted label for edges going into the symbol classifier. For nodes that go into symbol detector and relation classifier the ground truth label would be “Merge” and “NoRelation” respectively. Results shown in Table 6.3 computed with a 2-step training method (see Chapter 5), hence the high segmentation rate, on the CROHME 2019 test set. We group this experiment under recurrent training section in the table so it can be compared meaningfully.

A common approach to give the network more global context is a temporal layer, which is the key part in RNN-based models and Graph-CNNs. Temporal attention modules usually keep a memory of previous/next sequences and consider those decisions when making a prediction. Our model can benefit from having a knowledge of what are the previous nodes and edges. To experiment that we propose adding a sequential layer which allows for walking the graph and predicting nodes and edges at each step while keeping a memory of

previous nodes and edges, instead of batch prediction.

6.4 Loss and Training

Recurrent Training. We proposed a training method allowing to use intermediate segmentation results (first step) to update the input binary masks to symbol level and use those masks in a second round to generate updated features for primitives (e.g., strokes). This recurrent training can be done for more than two steps. We explored going another step using the second step segmentation results and generate a new symbol level masks for a third round. The symbol level masks from the second step would be mostly similar to the symbol masks we updated according to segmentation results in the first step as the segmentation decisions are similar in most cases, see Table 6.3.

Results of this experiment are shown in Table 6.3. The system description: a three classifier (no binary rel detector) CNN equipped with a trainable attention module made of three convolutional layers (1 kernel per layer). Although the difference is not significant in the overall recognition rate and we have reached a higher rate with one step training and three branch attention, segmentation rates are improved in this design due to selective updates in local context. We hypothesize that updating the symbol masks and adding local context helped mostly in segmentation as it is more reliant on local information. It is improving the symbol recognition rate as well, but could not solve the most frequent errors which is confusing the visually similar classes, e.g., (S,s), (X,x), (2,z). To solve these kinds of confusions and make the symbol classification better we need to exploit the global context better.

6.5 Isolated Symbol Classification Task

Symbol classification is the bottleneck of our design, so we decided to study it further individually.

Results of having only a single classifier in QD-GGA with trainable attentions are shown in the first row of Table 6.5. We pre processed the stroke points by (1) removing the duplicated points, (2) adding missing points, and (3) applying Catmull smoothing. This produces similar results to the baseline experiment as the smoothing step added decimal values which finally converted

to integers when drawing the input images (pixel values). We also tested the system by cropped and centered stroke images which reduces the context and increases the resolution. Cropping symbols eliminates the spatial information of strokes in the original setting. This results in decreasing the recognition rate as it makes it even harder to identify visually similar classes.

Another attempt to exploit higher resolution features was Early Masking. In this setup, attention masks are applied into intermediate feature maps instead of the last one. Then query features goes into the rest of the layers. We applied the masks on the feature maps from the first convolutional layer. Early masking reduces the context given by network field of view and filter the feature map in higher resolution. Results from Table 6.5 depicts that, similar to cropped strokes experiment, higher resolution cannot compensate for the lost context.

Table 6.5: Stroke classification results.

	Accuracy (%)
Isolated Strokes	**89.23**
Pre-processing	**89.01**
Cropped Strokes	76.09
Early Masking	71.27
Multi-scale Masking	85.61
Multi-scale Masking + noTemp	**89.29**

Therefore we designed experiments allowing the network to learn from both the higher resolution features from earlier layers and the later features that have more information from the input equation by multi-scale masking. The details of implementing these experiments are explained above. Important observation in Table 6.5 is removing the 3×1 convolutions (temporal layer) when having the multi-scale features provide a higher rate compared to having both multi-scale features and temporal layers. This also proved that the amount of context provided by temporal layer can be replaced by extracting features from different layers.

Table 6.6: Results of recognizing typeset formulas in INFTY with top QD-GGA architectures. The attention module has three blocks in all configurations.

	Symbols		**Relationships**		**Formulas**	
	Detection	Det.+Class	Detection	Det.+Class	Structure	Str.+Class
Binary Masks	99.07	97.03	95.29	95.01	89.38	81.84
1kernel	99.44	97.64	95.33	94.97	88.78	81.68
4kernel	99.50	97.82	95.71	95.43	89.46	82.27
4kernel-3branch	99.53	97.85	95.39	95.10	89.37	82.84
Pre-process	99.25	97.83	95.69	95.39	90.03	83.75
Pre-process + recursive train.	99.71	97.69	97.06	96.73	92.56	85.94

6.6 Typeset Formulas

QD-GGA can be used with images (offline recognition). We believe our method can be used for other visual parsing tasks, e.g., scene understanding, chemical diagrams, table extraction, etc. as long as the input data can be defined by graphs.

All the experiments so far were designed and evaluated on handwritten formulas from CROHME dataset. In this experiment we recognize scanned typeset formulas from INFTY dataset without any major changes. For raster images, the input LOS graph is computed on connected components instead of strokes in handwritten formulas. The input masks are ordered from left to right when passing into the attention module. If two boxes have the same horizontal location, we sort them from top down. The output layers are also modified for the INFTY dataset classes. 101 symbol classes in CROHME to 207 classes in INFTY and 7 relation classes into 9 relations in INFTY (RSUP, HORIZONTAL, RSUB, UNDER, LSUB, UPPER, LSUP, NoRelation, and PUNC).

This set of experiments are focused on testing if our method can be generalized on images of scanned typeset formulas in INFTY. The only parameter that is changed in training is the learning rate which is decreased from 0.01 to 0.001. Table 6.6 shows all the configurations tested for this dataset. We explored binary masks filtering the features directly and then tried the best configuration of attention modules (4 kernel, 3branch and 4 kernel). We then applied the pre-processing steps Condon suggested in [15] to use the smooth image contours which was helpful. In the pre-processing step, (1) single CC

per image is generated by applying CC bounding box information from the ground truth, (2) each CC image is converted into contours, (3) contours are treated as traces in the handwritten data and re-scaled to 64-pixel symbol height. This improves results slightly. Finally, we train the model in two steps with symbol masks updating (Section 5.2.5) which provides the best results for INFTY.

6.7 Benchmark

We benchmark our proposed model against state-of-the-art systems. Table 6.7 shows the results of the best configuration of our model (Relation-Specific features in Table 6.3) against state of the arts on CROHME 2014 and 2016 datasets. In this configuration we have three output classifiers (segmenter, classifier, parser), a three branch attention module and edge features go into and additional convolutional layer (early symbol feature extraction). Since encoder-decoder networks generate LaTeX string outputs, in most of these systems, they need to apply constraints in the decoder to make sure the output string is a valid tree. Our simple design gets comparable results with encoder-decoder systems that does not apply grammar rules (e.g IM2TEX). It should be mentioned we trained our system using only CROHME 2019 train set whereas, IM2TEX, WAP, TAP, and MAN trained or pre-trained on much larger datasets. It is important as we observed the effect of larger dataset when pre-training our system on skeletons extracted from INFTY dataset (Section 6.3). The QD-GGA achieves almost 10% higher recognition rate compares to LPGA supporting the idea of feature sharing and multi-task learning.

We present recognition rates and structure rates on CROHME 2019 test set in Table 6.8. Results show that the structure rates (unlabeled) in QD-GGA using only visual features is comparable to state-of-the-arts in which temporal and spatial attention and grammars are applied on the output strings. This is aligned with our observations on symbol classification task which is currently the most challenging task for our system. Improvement in symbol classification can be obtained by optimizing the features and exploiting the global information.

Table 6.9 compares our results on CROHME 2019 against the winners of the competition. Please note that recognition rates are not comparable with the other two tables as these results are computed using SymLG [66] format.

Table 6.7: Comparison against State-of-the-art math recognition models on CROHME 2014 and 2016. Expression rates are reported for comparisons.

	CROHME 2014			CROHME 2016					
System	ExpRate	≤ 1	≤ 2	ExpRate	≤ 1	≤ 2	Spatial att.	Temporal att.	Grammar
IM2TEX	35.90	-	-	-	-	-	Yes	Yes	No
TAP	48.47	63.28	67.34	44.81	59.72	62.77	Yes	Yes	Yes
WAP	48.38	66.13	70.18	46.82	64.64	65.48	Yes	Yes	Yes
MAN	54.05	68.76	72.21	50.56	64.78	67.13	Yes	Yes	Yes
LPGA_{RF}	26.88	36.63	42.50	-	-	-	Yes	No	No
QD-GGA	37.23	51.27	58.39	36.84	50.39	58.16	Yes	No	No

Table 6.8: Comparison against State-of-the-art math recognition models. Expression rates are reported for comparisons.

	CROHME 2019			
System	ExpRate	≤ 1	≤ 2	StruRate
TAP	44.20	58.80	62.72	63.64
WAP	48.12	63.47	67.22	67.97
OnSCAN	46.46	62.47	66.14	66.31
OffSCAN	47.62	63.14	67.06	67.81
QD-GGA	37.38	51.71	59.09	66.81

Evaluation metrics are explained in Chpater 3.

Table 6.9: Benchmarking QD-GGA against CROHME 2019 participating systems. Models evaluated using SymLG metrics.

	Structure + Symbol Labels			**Structure**
CROHME 2019	ExpRate	≤ 1	≤ 2	**Correct**
USTC-iFLYTEK	**80.73**	**88.99**	**90.74**	**91.49**
Samsung R&D	79.82	87.82	89.15	89.32
MyScript	79.15	86.82	89.82	90.66
QD-GGA	43.40	63.09	67.81	66.96

The training time reported for Tap [99] system is 780 sec/epoch for the base

model and it is even longer to train the ensemble models, whereas the training for our best configuration takes 254 sec/epoch. Execution time reported on CROHME2014 for TAP model is 377 sec, the WAP system [100] and the ensemble of TAP+WAP each takes 196 and 564 sec respectively. The execution time for QD-GGA on the same dataset is 59 sec which is much faster. These results display that our approach is computationally less costly and training and execution both are faster in our system due to our simpler approach.

Our system is built using PyTorch and experiments were run on an 8GB Nvidia 1080 GPU. Experiments were run on a server with an 8-core Intel Xeon E5-2667 processor (3.20 GHz per core), and 512 GB RAM was available. For TAP model, the experiments are all implemented with Theano 0.10.0 and an NVIDIA Tesla M40 12G GPU is used.

Figure 6.5 presents recognition results on CROHME 2019 including correctly recognized equations along with the examples of the most frequent errors. Most structure recognition failures are caused by missing edges in LOS graphs, or incorrect baseline detection as a result of size variations in handwritten characters, e.g., the "subscript" relation between a and β is classified as "right." Most symbol classification errors are among visually similar classes such as (X,x), (m,n), (α, a), (d,a), (z,2), etc. Lastly, the second row shows one of the few segmentation errors in which two strokes in an X did not merge and each is recognized as parenthesis individually. We noticed that the correctly recognized equations can be more complex than some of the occurring errors. We hypothesize that although images are denser, since strokes are closer the attention masks may capture more context locally. We are figuring out how to better use the context provided by the attention masks which is a key challenge to improve QD-GGA performance.

Recurrent training error analysis. In Figure 6.6, we present confusion histograms generated with LgEval tool to visualize the effect of recurrent training on 1-node and 2-node subgraphs. We tried to focus on symbols made of one stroke vs multiple strokes (e.g., m, n vs x, i). Please note for each type of errors, shown in blue columns, only the most frequent misclassifications are shown, hence some have three, some have less or more. Figure 6.6-a, shows both multi-stroke and single-stroke symbol errors are happening for visually similar classes (m,n), (i,j), (X,x). Similar behaviour observed in 2-step training (Figure 6.6-b) meaning the additional local context provided by updated symbol masks in recurrent training does not really help with confusion in vi-

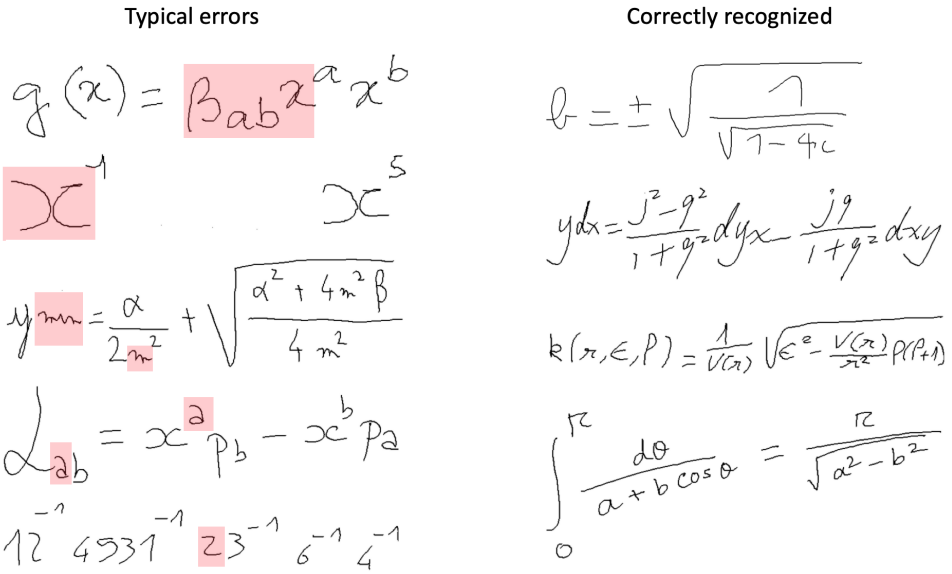


Figure 6.5: Example QD-GGA Results for CROHME 2019. Column at left shows the typical error cases; large subscripts mistaken with baseline (first row), visually similar symbols classified incorrectly (d instead of a and z instead of 2).

sually similar classes and a more global operation is needed to help with these cases.

Infty MCCDB-2. We present QD-GGA preliminary results on Infty MCCDB-2 dataset against LPGA systems. This should be mentioned that the only learning parameter modified for this dataset is the learning rate (from 0.01 to 0.001) and additional grid search for tuning learning parameters for this dataset might improves the results. We achieved higher segmentation rate with the recurrent training and the structure detection rate is comparable.

Context is included in these two systems differently. In LPGA, context region is cropped for each query meaning the surrounding is encoded directly for each node and edge. However, the context module in QD-GGA encodes neighboring queries for each target query (previous and next queries in the stack) without considering their spatial positions in the input image. We hypothesize that the dependency of the context module on the order of incoming feature streams in QD-GGA system might cause the lower recognition rates for images.

Table 6.10: InftyMCCDB-2 test set results for correct symbol/relationship locations (*Detection*), correct symbol/relationship classes (*Det.+Class*), unlabeled formula SLT structure, and structure with correct labels (*Str.+Class*). Percentage of correct formulas are shown.

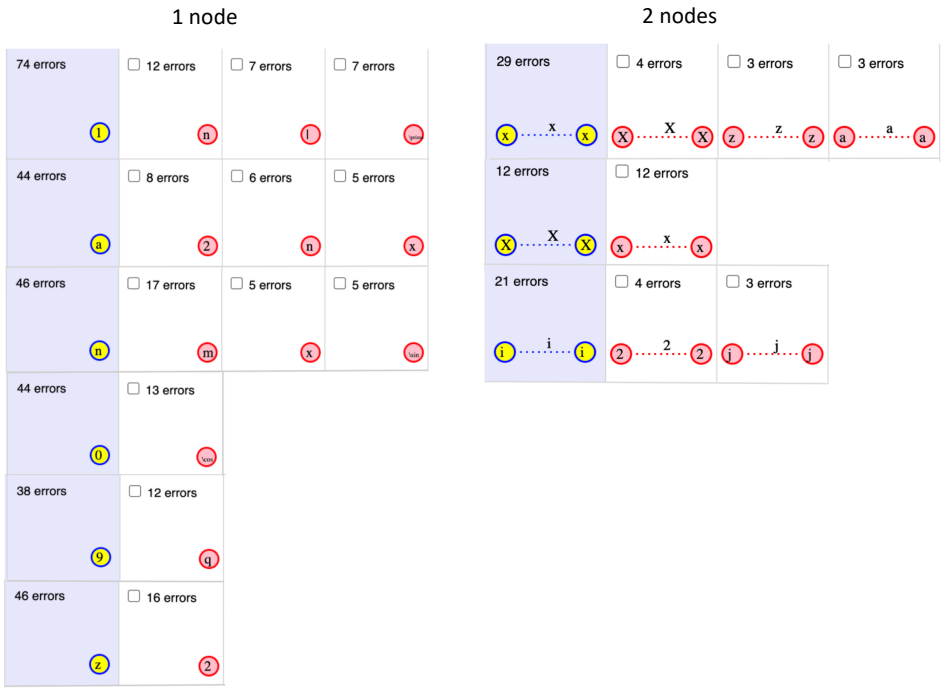
	Symbols		**Relationships**		**Formulas**	
	Detection	Det.+Class	Detection	Det.+Class	Structure	Str.+Class
QD-GGA	**99.71**	97.69	97.06	96.73	92.56	85.94
LPGA_{RF}	99.34	98.51	97.83	**97.56**	**93.81**	90.06
LPGA_{CNN}	99.35	**98.95**	**97.97**	97.74	93.37	**90.89**

6.8 Summary

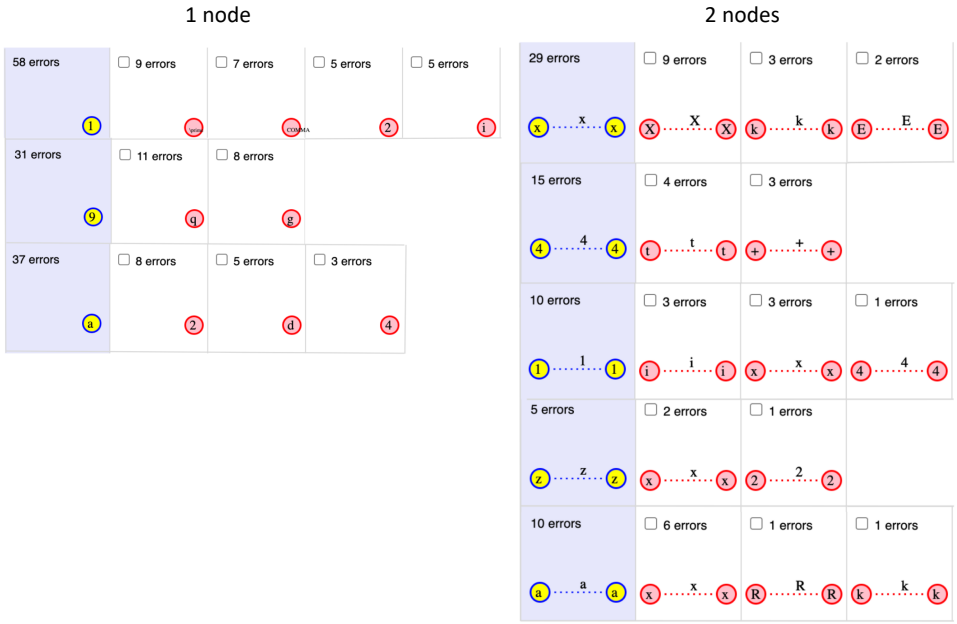
We have presented an approach for recognition of handwritten and typeset math formulas. Features in our approach are shared between classifiers and the query features for all strokes/CCs and their relations are computed in one iteration. It is worth to mention that our initial experiments suggested that it is better to have individual kernels for each task in the attention module (3 attention layer) and individual classifiers assigned to symbol detection, symbol

and relation classification. We compared the accuracy of our approach to the state of the arts and obtained comparable results with other approaches given the fact our method is much faster and does not apply grammar constraints. The structure detection (unlabeled) in QD-GGA is competitive to encoder-decoder models.

The main experimental results show that Line-Of-Sight Parsing with Query-Driven Global Graph Attention (QD-GGA) is effective for math expressions recognition and needs further improvement in symbol classification to be both faster and more accurate than sequential approach. This can be obtained by optimizing the features and exploiting the global information.



(a) Frequent patterns in 1-step training errors



(b) Frequent patterns in 2-step training errors

Figure 6.6: Comparing Frequent errors in single-stroke vs multiple-stroke subgraphs for 1-step training (a) against 2-step training (b). Error analysis is done for CROHME 2019 test set. Note that for each error type (yellow references), only the frequent misclassifications are shown.

Chapter 7

Conclusion

Through all of our research work we could observe that a simple model based on convolutional neural networks tends to be enough to produce satisfactory results for math expression recognition, Chapter 6. Our proposed approach is faster compared to sequential models both in training and execution and is able to do very accurate segmentation through recurrent execution. The structure detection in QD-GGA is comparable with encoder-decoder systems trained on much larger datasets. Adding a temporal attention, enhancing symbol classification through a better use of global context and spatial features, or using a larger dataset can make QD-GGA results competitive to state-of-the-art systems while keeping it faster.

7.1 Contributions

In this work, we introduced a Query-Driven Global Graph Attention (QD-GGA) parsing model, a new CNN-based variant of graph-based visual parsing. A summary of our contributions is listed below.

- **LPGA:** a new structure is proposed to add contextual visual information with a two branch CNN that takes target and context images for segmentation, classification and parsing. The LPGA architecture and methodology is discussed in Chapter 4. We benchmarked LPGA system on both CROHME and InftyMCCDB-2 datasets in Chapter 6.

- **QD-GGA:** this approach generalizes our previous work using CNN-based features [65], with features and attention modules trained concurrently for multiple tasks as described in Chapter 5. our novel graph-based attention module allows multiple classification queries for nodes and edges to be computed in one feed-forward pass resulting in faster training and execution, see Chapter 6. By using a Multi Task Learning (MTL) framework, it is possible to train our CNN for different tasks simultaneously from an output adjacency matrix containing class distributions at each entry. This provides generalization for feature representations, and a more global view for classifiers through a shared joint loss.

- **Graph-Based Attention in QD-GGA:** the input graph used both for parsing the formula and query the features through attention module. We introduce an attention module that queries CNN features from a single image to efficiently obtain inputs for multiple classification, segmentation, and relationship decisions by defining a graph on input image, see Section 5.2.2 for more details on attention module.

- **Recurrent updating of masks in QD-GGA:** segmentation results at each iteration is used to update primitive-level masks to symbol-level masks. This method keeps the input and output spaces on primitive level but makes it possible for query features to access symbol level information. The additional local information provided through this approach makes the segmentation task very strong, see Table 6.3.

7.2 Future Work

In the future, we would like to improve symbol and relation classification tasks by adding a graph sequential learning [93, 94] module instead of the simple 1-by-3 convolution. This provides a memory of seen nodes and edges and therefore adds a more global view to the system. So, the input graph is used both for having a memory of all nodes and edges as well as guiding a spatial and temporal attention model. This setup will eliminate the impact of both the input sequence in the current context module and the hard decision making on how many neighbors should contribute (e.g., two neighbors) for each target query. In this method, the surrounding characters are included

for each target if the input graph has an edge connecting them that addresses the lack of encoded surroundings in QD-GGA. The sequential attention can be applied on query features meaning it learns how much of other neighbors should contribute for each target query (by convolution) or it can be applied on the adjacency matrix. In the latter, the scores of nodes and edges will be fed into a GCN that can learn to extract a tree from the weighted graph. The advantage of extracting a tree in this way instead of MST extraction with Edmond's algorithm is that all the four modules can be trained end-to-end and weights will be back propagated through the network based on the final tree.

In another attempt to better exploit the global information in this integrated approach, we would like to expand our primitive studies on embedding to include adjacency matrix probabilities in feature extraction. For this purpose, each input binary mask is multiplied with all the class probabilities, e.g., the 1-D masks are transforms to 101-D masks based on the symbol classifier probabilities in handwritten recognition. The input spatial masks are used to represent embedding masks by getting multiplied with class probabilities. For instance, each binary node mask is converted into 101 masks with background pixels being zeros but the pixel values multiplied with probability of each class. Next, the binary mask stacks with 101 channels are collapsed to a single stack by getting the maximum pixel values across the channels. These embedding masks are normalized across the channels and then encoded into a lower dimension (e.g., 101 symbol classes to 3) using 1-by-1 convolution. These new embedding masks with lower dimensions will be passed to the network in the next iteration to query global features from our general feature map. Note that in the first iteration, the embedding masks are all zeros.

We would also like to study undirected graphs as input representations. In this case, the final output would also be an undirected tree and rules must be defined to find the parent-child relations. This will decrease the input edges to half as we only need to label either upper or lower triangle in the adjacency matrix. We are interested to experiment with undirected graphs as it allows to work with complete graphs but having a smaller search space. In terms of features, we would have only one edge per primitives pair (no direction), so we do not need to concatenate the pair masks with the parent mask to introduce the visual difference in different directions of the edge.

Another future direction to study the input representations is taking care

of the missing edges in LOS graph by training a small network which decides which edges are miscellaneous. The purpose of this low-cost network would be reducing the edges in the input graph and provide a smaller search space similar to what Line-Of-Sight Graph is responsible for in the current QD-GGA architecture.

We would like to study the joint loss and explore if the training can be improved with a different definition for shared tasks. We particularly would like to study if a weighted combination of the three loss could make the training faster/better as the edge samples and node samples are not balanced and the difficulty of the tasks varies. Focal Loss [57] is another candidate that can weights the edges differently in calculating the errors. Instead of removing the non-contributing edges in tree loss we can make soft decisions with focal loss.

Finally, we would like to apply QD-GGA to similar visual parsing problems e.g., parsing chemical diagrams. As our approach can be utilized for any visual parsing task in which inputs and outputs may be defined using directed graphs.

Bibliography

[1] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1012–1016. IEEE, 2013.

[2] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models. *Pattern Recognition Letters*, 35:58–67, 2014.

[3] Francisco Alvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. An integrated grammar-based approach for mathematical expression recognition. *Pattern Recognition*, 51:135–147, 2016.

[4] Ahmad-Montaser Awal, Harold Mouchere, and Christian Viard-Gaudin. Towards handwritten mathematical expression recognition. In *2009 10th International Conference on Document Analysis and Recognition*, pages 1046–1050. IEEE, 2009.

[5] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. A global learning approach for an online handwritten mathematical expression recognition system. *Pattern Recognition Letters*, 35:68–77, 2014.

[6] Abdelwaheb Belaid and Jean-Paul Haton. A syntactic approach for handwritten mathematical formula recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):105–111, 1984.

[7] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent

neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2874–2883, 2016.

[8] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):509–522, 2002.

[9] Dorothea Blostein and Ann Grbavec. Recognition of mathematical notation. In *Handbook of character recognition and document image analysis*, pages 557–582. World Scientific, 1997.

[10] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[11] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[12] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.

[13] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[14] Philip A Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Visual Communications and Image Processing IV*, volume 1199, pages 852–865. International Society for Optics and Photonics, 1989.

[15] Michael Patrick Erickson Condon. Applying hierarchical contextual parsing with visual density and geometric features to typeset formula recognition. Master's thesis, Rochester Institute of Technology, 2017.

[16] Bo Dai, Yuqi Zhang, and Dahua Lin. Detecting visual relationships with deep relational networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3076–3086, 2017.

[17] Kenny Davila, Stephanie Ludi, and Richard Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 323–328. IEEE, 2014.

[18] Brian Davis, Bryan Morse, Scott Cohen, Brian Price, and Chris Tensmeyer. Deep visual template-free form parsing. *arXiv preprint arXiv:1909.02576*, 2019.

[19] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. Computational geometry: introduction. *Computational Geometry: Algorithms and Applications*, pages 1–17, 2008.

[20] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-markup generation with coarse-to-fine attention. *arXiv preprint arXiv:1609.04938*, 2016.

[21] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017.

[22] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[23] Jack Edmonds. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240, 1967.

[24] Sagi Eppel. Setting an attention region for convolutional neural networks using region selective features, for recognition of materials within glass vessels. *arXiv preprint arXiv:1708.08711*, 2017.

[25] Yuko Eto and Masakazu Suzuki. Mathematical formula recognition using virtual link network. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 762–767. IEEE, 2001.

[26] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM, 2004.

[27] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[28] J Fitzgerald, Franz Geiselbrechtinger, and Tahar Kechadi. Mathpad: A fuzzy logic-based recognition system for handwritten mathematics. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 694–698. IEEE, 2007.

[29] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[30] Ryoji Fukuda, I Sou, and Fumikazu Tamari. A technique of mathematical expression structure analysis for the handwriting input system. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*, pages 131–134. IEEE, 1999.

[31] Felix A Gers, Jürgen A Schmidhuber, and Fred A Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

[32] Georgia Gkioxari, Bharath Hariharan, Ross Girshick, and Jitendra Malik. R-cnns for pose estimation and action detection. *arXiv preprint arXiv:1406.5212*, 2014.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Ann Grbavec and Dorothea Blostein. Mathematics recognition using graph rewriting. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 417–421. IEEE, 1995.

[35] Chaojun Han, Fumin Shen, Li Liu, Yang Yang, and Heng Tao Shen. Visual spatial attention network for relationship detection. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 510–518. ACM, 2018.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[37] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[38] Lei Hu and Richard Zanibbi. Line-of-sight stroke graphs and parzen shape context features for handwritten math formula representation and symbol segmentation. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 180–186. IEEE, 2016.

[39] Lei Hu and Richard Zanibbi. Mst-based visual parsing of online handwritten mathematical expressions. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 337–342. IEEE, 2016.

[40] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[41] Saumya Jetley, Nicholas A Lord, Namhoon Lee, and Philip HS Torr. Learn to pay attention. *arXiv preprint arXiv:1804.02391*, 2018.

[42] Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, and Nina ST Hirata. Top-down online handwritten mathematical expression parsing with graph grammar. In *IberoAmerican Congress on Pattern Recognition*, pages 444–451. Springer, 2015.

[43] Frank JulcaAguilar, Nina ST Hirata, Christian ViardGaudin, Harold Mouchère, and Sofiane Medjkoune. Mathematical symbol hypothesis recognition with rejection option. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 500–505. IEEE, 2014.

[44] Vadim Kantorov, Maxime Oquab, Minsu Cho, and Ivan Laptev. Contextlocnet: Context-aware deep network models for weakly supervised localization. In *European Conference on Computer Vision*, pages 350–365. Springer, 2016.

[45] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[46] Birendra Keshari and S Watt. Hybrid mathematical symbol recognition using support vector machines. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 859–863. IEEE, 2007.

[47] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[48] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 845–853, 2016.

[49] Andreas Kosmala, Gerhard Rigoll, Stephane Lavirotte, and Loic Pottier. On-line handwritten formula recognition using hidden markov models and context dependent graph grammars. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*, pages 107–110. IEEE, 1999.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417*, 2012.

[52] Stéphane Lavirotte and Loïc Pottier. Mathematical formula recognition using graph grammar. In *Document Recognition V*, volume 3305, pages 44–52. International Society for Optics and Photonics, 1998.

[53] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[54] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[55] Zuoxin Li and Fuqiang Zhou. Fssd: feature fusion single shot multibox detector. *arXiv preprint arXiv:1712.00960*, 2017.

[56] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[57] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[58] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

[59] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[60] Wei Liu, Andrew Rabinovich, and Alexander C Berg. Parsenet: Looking wider to see better. *arXiv preprint arXiv:1506.04579*, 2015.

[61] Yang Liu, Zhaowen Wang, Hailin Jin, and Ian Wassell. Synthetically supervised feature learning for scene text recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 435–451, 2018.

[62] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013.

[63] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pages 4898–4906, 2016.

[64] Scott MacLean and George Labahn. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *International Journal on Document Analysis and Recognition (IJDAR)*, 16(2):139–163, 2013.

[65] Mahshad Mahdavi, Michael Condon, Kenny Davila, and Richard Zanibbi. Lpga: Line-of-sight parsing with graph-based attention for math formula recognition. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 647–654. IEEE, 2019.

[66] Mahshad Mahdavi and Richard Zanibbi. Tree-based structure recognition evaluation for math expressions: Techniques and case study.

[67] Mahshad Mahdavi and Richard Zanibbi. Visual parsing with query-driven global graph attention (qd-gga): Preliminary results for handwritten math formula recognition. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[68] Mahshad Mahdavi, Richard Zanibbi, Harold Mouchère, Christian Viard-Gaudin, and Utpal Garain. Icdar 2019 crohme+ tfd: Competition on recognition of handwritten mathematical expressions and typeset formula detection. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1533–1538. IEEE, 2019.

[69] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3994–4003, 2016.

[70] Harold Mouchere, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the crohme competitions, 2011–2014. *International Journal on Document Analysis and Recognition (IJDAR)*, 19(2):173–189, 2016.

[71] Harold Mouchère, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014. *IJDAR*, 19(2):173–189, 2016.

[72] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

[73] Liang Peng, Yang Yang, Zheng Wang, Xiao Wu, and Zi Huang. Cra-net: Composed relation attention network for visual question answering. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 1202–1210. ACM, 2019.

[74] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *European Conference on Computer Vision*, pages 75–91. Springer, 2016.

[75] D Prusa and Václav Hlavác. Mathematical formulae recognition using 2d grammars. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 849–853. IEEE, 2007.

[76] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[77] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[78] Taik Heon Rhee and Jin Hyung Kim. Efficient search strategy in structural analysis for handwritten mathematical expression recognition. *Pattern Recognition*, 42(12):3192–3201, 2009.

[79] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[80] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *Graphics Interface*, volume 99, pages 84–91, 1999.

[81] Chunfeng Song, Yan Huang, Wanli Ouyang, and Liang Wang. Mask-guided contrastive attention model for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1179–1188, 2018.

[82] Masakazu Suzuki, Seiichi Uchida, and Akihiro Nomura. A ground-truthed mathematical character and symbol image database. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 675–679. IEEE, 2005.

[83] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[84] Ba-Quy Vuong, Yulan He, and Siu Cheung Hui. Towards a web-based progressive handwriting recognition environment for mathematical problem solving. *Expert Systems with Applications*, 37(1):886–893, 2010.

[85] Jiaming Wang, Jun Du, Jianshu Zhang, and Zi-Rui Wang. Multi-modal attention network for handwritten mathematical expression recognition.

[86] Jian Wang, Yunchuan Sun, and Shenling Wang. Image to latex with densenet encoder and joint attention. *Procedia computer science*, 147:374–380, 2019.

[87] Xiaolong Wang, David Fouhey, and Abhinav Gupta. Designing deep networks for surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–547, 2015.

[88] H-J Winkler, H Fahrner, and Manfred Lang. A soft-decision approach for structural analysis of handwritten mathematical expressions. In *1995*

International Conference on Acoustics, Speech, and Signal Processing, volume 4, pages 2459–2462. IEEE, 1995.

[89] Jin-Wen Wu, Fei Yin, Yan-Ming Zhang, Xu-Yao Zhang, and Cheng-Lin Liu. Handwritten mathematical expression recognition via paired adversarial learning. *International Journal of Computer Vision*, pages 1–16, 2020.

[90] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[91] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[92] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[93] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 670–685, 2018.

[94] Ting Yao, Yingwei Pan, Yehao Li, and Tao Mei. Exploring visual relationship for image captioning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 684–699, 2018.

[95] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.

[96] Richard Zanibbi, Dorothea Blostein, and James R Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.

[97] Richard Zanibbi and Awelemdy Orakwue. Math search for the masses: Multimodal search interfaces and appearance-based retrieval. In *Conferences on Intelligent Computer Mathematics*, pages 18–36. Springer, 2015.

[98] Richard Zanibbi, Amit Pillay, Harold Mouchere, Christian Viard-Gaudin, and Dorothea Blostein. Stroke-based performance metrics for handwritten mathematical expressions. In *2011 International Conference on Document Analysis and Recognition*, pages 334–338. IEEE, 2011.

[99] Jianshu Zhang, Jun Du, and Lirong Dai. Track, attend, and parse (tap): An end-to-end framework for online handwritten mathematical expression recognition. *IEEE Transactions on Multimedia*, 21(1):221–233, 2018.

[100] Jianshu Zhang, Jun Du, Shiliang Zhang, Dan Liu, Yulong Hu, Jinshui Hu, Si Wei, and Lirong Dai. Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. *Pattern Recognition*, 71:196–206, 2017.

[101] Ting Zhang, Harold Mouchère, and Christian Viard-Gaudin. A tree-blstm-based recognition system for online handwritten mathematical expressions. *Neural Computing and Applications*, pages 1–20, 2018.

[102] Yaping Zhang, Shan Liang, Shuai Nie, Wenju Liu, and Shouye Peng. Robust offline handwritten character recognition through exploring writer-independent features under the guidance of printed data. *Pattern Recognition Letters*, 106:20–26, 2018.

Appendices

Appendix A

Publications

1. Mahdavi, M., Sun, L., Zanibbi, R. (2020). Visual Parsing with Query-Driven Global Graph Attention (QD-GGA): Preliminary Results for Handwritten Math Formula Recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 570-571).

2. Mahdavi, M., Condon, M., Davila, K., Zanibbi, R. (2019, September). LPGA: Line-of-sight parsing with graph-based attention for math formula recognition. In 2019 International Conference on Document Analysis and Recognition (ICDAR) (pp. 647-654). IEEE.

3. Mahdavi, M., Zanibbi, R., Mouchère, H., Viard-Gaudin, C., Garain, U. (2019, September). ICDAR 2019 CROHME+ TFD: Competition on recognition of handwritten mathematical expressions and typeset formula detection. In 2019 International Conference on Document Analysis and Recognition (ICDAR) (pp. 1533-1538). IEEE.

4. Mahdavi, M., Zanibbi, R. (2019). Tree-Based Structure Recognition Evaluation for Math Expressions: Techniques and Case Study. In 2019 IAPR International Workshop on Graphics Recognition (GREC).

5. Mahdavi, M., Condon, M., Zanibbi, R. (2018). Applying Hierarchical Contextual Parsing to Recognize Isolated Typeset Math Formulas. In 2018 Western NY Image and Signal Processing Workshop.

6. Mali, P., Kukkadapu, P., Mahdavi, M., Zanibbi, R. (2020). ScanSSD: Scanning Single Shot Detector for Mathematical Formulas in PDF Document Images. arXiv preprint arXiv:2003.08005.

7. Mahdavi, M., Kanan, C., Salvaggio, C. (2017). Roof Damage Assessment using Deep Learning. In 2017 IEEE Applied Imagery Pattern Recognition (AIPR), 6403-6408.