Multimodal Knowledge Graph for Deep Learning Papers and Code

Amar Viswanathan Kannan
Dmitriy Fradkin
Ioannis Akrotirianakis
Tugba Kulahcioglu
Arquimedes Canedo
Aditi Roy
firstname.lastname@siemens.com
Siemens Corporate Technology
Princeton, NJ, USA

Shih-Yuan Yu
Malawade Arnav
Mohammad Abdullah Al Faruque
shihyuay@uci.edu
malawada@uci.edu
alfaruqu@uci.edu
Department of Electrical Engineering and Computer Science
University of California, Irvine, USA.

ABSTRACT

Keeping up with the rapid growth of Deep Learning (DL) research is a daunting task. While existing scientific literature search systems provide text search capabilities and can identify similar papers, gaining an in-depth understanding of a new approach or an application is much more complicated. Many publications leverage multiple modalities to convey their findings and spread their ideas - they include pseudocode, tables, images and diagrams in addition to text, and often make publicly accessible their implementations. It is important to be able to represent and query them as well. We utilize RDF Knowledge graphs (KGs) to represent multimodal information and enable expressive querying over modalities. In our demo we present an approach for extracting KGs from different modalities, namely *text*, *architecture images* and *source code*. We show how graph queries can be used to get insights into different facets (modalities) of a paper, and its associated code implementation. Our innovation lies in the multimodal nature of the KG we create. While our work is of direct interest to DL researchers and practitioners, our approaches can also be leveraged in other scientific domains.

CCS CONCEPTS

• **Computing methodologies** → **Information extraction**; **Knowledge representation and reasoning**; **Ontology engineering**.

KEYWORDS

Knowledge graphs, Multimodal Information Retrieval, Deep Learning, Scientific Knowledge Graphs, Scientific Knowledge Graph Exploration

ACM Reference Format:
Amar Viswanathan Kannan, Dmitriy Fradkin, Ioannis Akrotirianakis, Tugba Kulahcioglu, Arquimedes Canedo, Aditi Roy, Shih-Yuan Yu, Malawade Arnav, and Mohammad Abdullah Al Faruque. 2020. Multimodal Knowledge Graph for Deep Learning Papers and Code. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3417439

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3417439

1 INTRODUCTION

Deep Learning (DL) has experienced a remarkable growth in recent years with thousands of researchers working on new models, architectures and applications every day. DL models are complex to express and implement, and keeping up with all the latest publications and their accompanied source code becomes a great challenge both for researchers looking to develop new methods and for practitioners who want to use existing methods to solve real-world problems. The main goal of our project - Deep Code Curator (DCC)[1] - is to address this issue by extracting the information from scientific publications and the accompanying source code and representing it as a unified knowledge graph. This allows researchers and practitioners to pose expressive queries such as *Show me all the tensorflow functions used by CNNs in papers from CVPR between the years 2013 and 2018* or *Show me all the CNN publications which have average pooling in their architecture diagrams* . This can dramatically decrease the time, effort and resources spent curating DL literature and algorithms. We focus on three modalities that can provide useful information: *text, images,* and *source code*. We have developed modules to process each of these modalities: *text2graph, image2graph,* and *code2graph*. We show how KGs are extracted for each modality from each paper, and how they are aligned and merged into a single multimodal KG, based on the RDF framework. We further show that the resulting KG can be used to identify, compare and contrast deep learning techniques across different publications and implementations.

2 RELATED WORK

There has been a lot of work in scientific literature on knowledge base construction. Most of it focused on extracting information from text and then representing it as knowledge graphs. For example, [7] built knowledge graphs for degenerative diseases. The *ScienceIE* task competition from Semeval 2017 [2] led to development of multiple systems covering different aspects of knowledge graph generation. Another example of KG creation from text alone is [4]. Semantic

[1] https://github.com/deepcurator/DCC
[2] https://scienceie.github.io/

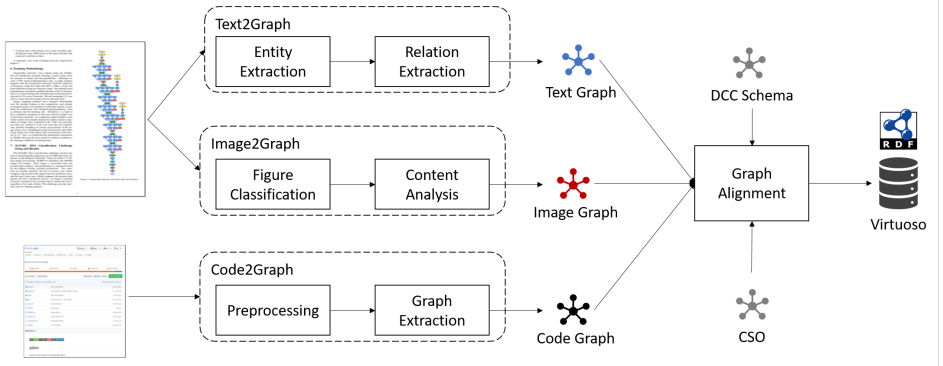

Figure 1: System Architecture

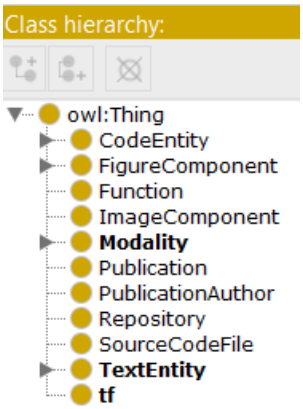

Figure 2: Schema Hierarchy

scholar [2] also allows searching and some statistics about similar papers. Recently, [1] described generation of a KG from source code. Our system is different from the existing systems in that we want to represent the entire information of a Deep Learning model from different modalities. The KGs extracted separately from text, diagrams and source code are aligned into a single KG for each paper and code, which are in turn aggregated into a single multimodal KG representing the field of Deep Learning.

3 SYSTEM OVERVIEW

The architecture of our system is shown in Figure 1. The input to the system is a set of papers and their associated code repositories. From the paper abstracts and introductions the system extracts entities and relations. Similarly, if the publication has an image of a deep learning architecture, different components of the image are extracted and an image knowledge graph is constructed. Finally, a code knowledge graph is extracted from the referenced code implementation. These graphs are then converted to a standard RDF format [9] using the DCC schema ontology. This conversion step also includes an alignment phase, which takes into account how same or similar entities are represented in each of these modalities. After alignment we get a RDF knowledge graph which can be stored and queried in any triple store. All our code and documentation can be found online[3]. In the following sections we describe the different components of our system from the ontology design to the different types of knowledge graph extractors.

3.1 Knowledge Graph Ontology

The knowledge graph ontology schema is the glue that unifies the different modalities into a single representation. The top level classes of our knowledge graph ontology [4] are shown in Figure 2. Figures 3 and 4 show lower levels of the ontology. Our ontology consists of 278 object classes, 25 Object properties, 18 data properties and 2 annotation properties. The object and data properties are used to link the instances together according to the linked data principles.

[3]https://github.com/deepcurator/DCC
[4]https://osf.io/bq5h6/

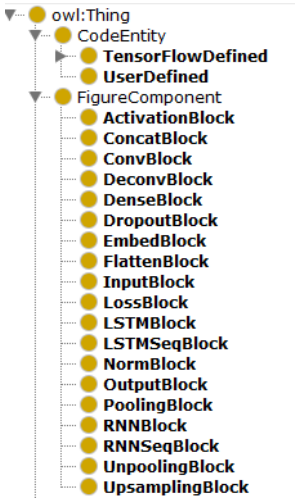

Figure 3: Code and Image schema classes

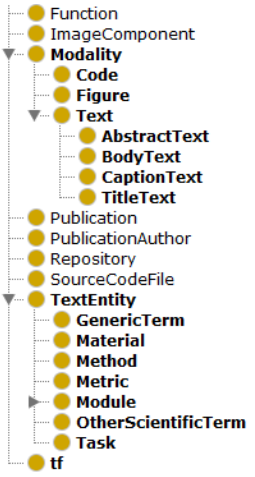

Figure 4: Text schema classes

3.2 Text2Graph

This module uses Named Entity Recognition (NER) to detect DL based entities and Relation Extraction (RE) to determine relations between two or more entities occurring within a sentence. The graph is then built by constructing two nodes from each of the entities and linking them by the named relationship edge. The details of NER and RE are given below. Since this is a new task, we created an annotated dataset of DL models and relations. We used the web-based annotation tool *brat*[5] to perform text annotation. The entity and relation types follow those suggested in [5].

- **Named entity extraction:** For NER we used spaCy[6] with our own annotations to train the model. The entity types that we used are (1) Task - e.g., information extraction, forecasting, image analysis, (2) Method - e.g., Neural Network,

[5]https://brat.nlplab.org/
[6]https://spacy.io

Attention, CRF, CNN, RNN, (3) Evaluation metric - e.g., F1, Precision, Recall, ROC curve, (4) Material - e.g., data, datasets, corpus, (5) Other scientific terms - e.g., dbpedia, Wikipedia, CoNLL, and (6) Generic - e.g., model, approach, algorithm .

- **Relation extraction:** For RE we leveraged a Bidirectional LSTM model along with a neural attention mechanism to capture the relationships [6]. We extract the following relationships (1) Used for – e.g., B is used for A, B models A, (2) Feature of - B belongs to A B is a feature of A, (3) Part of – e.g., Our system includes models A and B, (4) Compare - Comparing two works, (5) Conjunction - Symmetric relation, (6) isA – e.g., DNN is a type of artificial neural network, and (7) sameAs : ex. NMT, otherwise known as neural machine translation.

3.3 Image2Graph

Our approach to converting DL architecture diagrams to KGs is described in detail in [8]. Briefly, image2graph pipeline (shown in Figure 5 consists of four main steps: (1) extraction of all the figures from a research paper in PDF form, (2) selection of figures showing DL model diagram, (3) analysis of the diagrams depicting DL models, and (4) construction of a graph representing the information extracted from the DL diagram. For each DL image we extract the following relationships *isA, foundIn, hasCaption, partOf, isType, hasDescription, hasFlow, followedBy.*

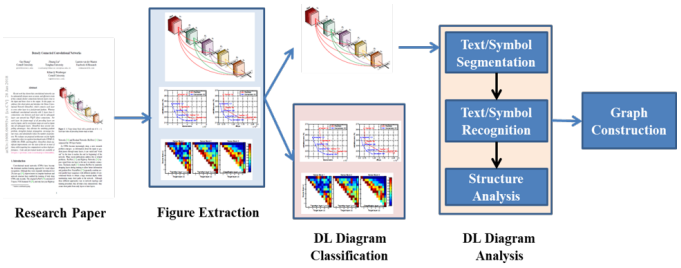

Figure 5: Pipeline for image2graph

3.4 Code2Graph

The details of our approach to code2graph are described in [10]. We analyze the static syntactic structure of the code since this does not require compilation and execution of code, making this approach efficient and scalable to large code bases (i.e., many github repositories representing the source code of DL papers). By utilizing the ontology and vocabulary, we extract information on the DL architecture implemented (e.g., layer types, activation functions, optimizers, etc.) and construct a static call graph for the project code. Child nodes in the generated tree structure can be used to identify whether the function calls are TensorFlow or UserDefined. Based on the call tree structure, the corresponding relations (such as calls and followedBy) between functions are then added. By inspecting all the source files in a project, we generate an RDF graph to represent the project code.

Publications #	**539**
DL Entities #	**7999**
DL Figures #	174
DL Repositories #	256

Table 1: Knowledge graph statistics

3.5 Knowledge Graph Alignment

The knowledge graphs generated from different modalities represent complementary information. For example, a paper talking about *convolutional neural networks* may also include architecture diagram of the same. A code block referencing *convolutional network* may also be present in the associated implementation. Each of these concepts would be present as a separate node in the atomic knowledge graphs. In the knowledge graph alignment step we identify these complementary nodes and then *align* or *link* them together. This results in a richer multimodal knowledge graph representation of each paper. Knowledge graph alignment for the three modalities involves the following two phases:

- **Local graph alignment**: During the knowledge graph mapping, we map respective modality instances of data to their conceptual schema elements. For example the text entity *convolutional neural network* is mapped to the schema element Entity. The image entity convolutional neural network is mapped to a *ConvBlock* and code entity would be mapped to the respective subclass of *TensorflowDefined*. Since all of these elements are connected to a single paper through the Publication entity, we perform a local alignment by looking for entities that are named similarly across modalities and then linking them together.
- **Global graph alignment**: We have decided to use an external well-curated knowledge graph called the Computer Science Ontology (CSO) - a large scale research ontology that categorizes 16 million publications, mainly in the field of computer science as a taxonomy [3]. Out of the 14K topics and 163K relationships, we mapped all our instances of concepts from the text extraction (i.e., the entities of the text extraction to their respective CSO components). The entities that were mapped were instances of Material, Method, Metric and Task.We also performed string matching to link entities from image and code to CSO. This resulted in a total of 978 entities mapped to CSO.

The final statistics of the knowledge graph is shown in Table 1.

4 DEMONSTRATION OVERVIEW

In our demonstration, we first focus on how the atomic knowledge graphs of each modality are extracted and then visualized as a single Paper2Graph. While such a graph is useful for exploration of an individual paper, data driven systems rely on large scale analysis results and comparisons. KGs collected and unified over multiple papers can provide insights into the state of the art in the field. Our querying demo shows how our knowledge graph can be utilized to perform data-driven analysis and model exploration.

4.1 Paper2Graph Demo

We merge and align the three different modality-specific KGs into a single merged KG, which we refer to as as Paper2Graph, as described in the previous section. Figure 6 annotates the different subgraphs by numbers, the nodes by different colors based on the entities they represent, and the edges by different arrows. Specifically, sub-graphs shown in boxes 1, 6 and 7 represent code2graph, text2graph and image2graph, respectively. Boxes 4, 3, and 2 include our overall Ontology. Box 4 covers Ontology concepts related to the individual modalities of DCC (text, image and source code). Box 2 covers TensorFlow defined functions (such as tf.constant, tf.placeholder, tf.nn, tf.nn.conv2d, tf.nn.relu, tf.nn.tanh, tf.nn.softmax, etc.) and box 3 covers the TensorFlow classes (such as tf.Graph, tf.Tensor, tf.Module, tv.Variable, etc).

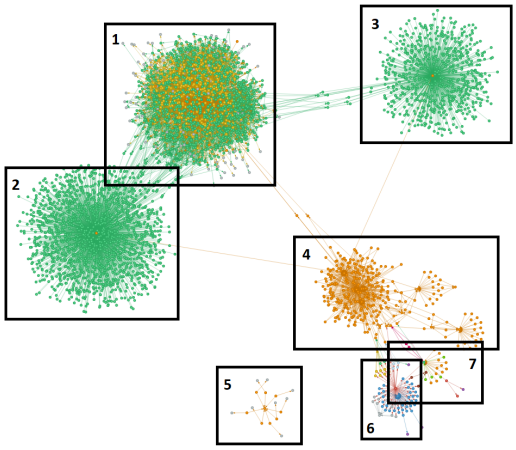

Figure 6: The Paper2Graph KG combines all the modalities: text (box 6), image (box 7) and source code (box 1). Our ontology is visible in boxes 2, 3 and 4.

We will demonstrate how, given an input pdf of a DL paper and a link to a Tensorflow code repository, our system generates KGs for individual modalities and a combined one for the paper. This demo is available online in a Jupyter notebook[7].

4.2 Graph Querying Demo

```
Select count(?type) as ?counttype ?cso ?type where {
?s <https://github.com/deepcurator/DCC/hasFigure>?f .
?component <https://github.com/deepcurator/DCC/partOf>?f .
?component <https://github.com/deepcurator/DCC/hasCSOEquivalent>?cso.
?s <https://github.com/deepcurator/DCC/hasRepository>?repository .
?repository <https://github.com/deepcurator/DCC/hasFunction>?y.
?y a ?type .
FILTER(!(STR(?type) = "https://github.com/deepcurator/DCC/UserDefined")).
}group by ?cso ?type ORDER by DESC(?counttype)
```

Table 2: Sample Multimodal SPARQL query showing frequency of tensorflow modules across architecture diagrams

[7]https://github.com/deepcurator/DCC/tree/master/demo/run_all_modalities

KGs from individual papers are combined into a large knowledge base, which is stored a Virtuoso triple store backend. We demo a Jupyter notebook for visualizing query results on this triple store. We show example queries that explore the different relationships between DL models, their architectures, source codes and other aspects of the publication. For this demo, we create a DL knowledge graph from paperswithcode [8]. This demo is available online in a Jupyter notebook[9]. More examples of KG queries can be found in our repository.[10]

4.3 Target Audience

The demonstrations are aimed at Deep Learning researchers and practitioners. It also targets both experts and knowledge graph enthusiasts. The enthusiasts can easily query the knowledge base to look for advances in Deep Learning and also look for trends in published papers. On the other hand the scientists could use this system as a way to summarize the advances in the field. In addition they could also dive deep into model specific implementations. The demonstration is also of interest to the larger scientific community since the approaches we developed are applicable to other scientific domains.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) Automating Scientific Knowledge Extraction (ASKE) Program under contract no. HR00111990010.

REFERENCES

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2017. Learning to Represent Programs with Graphs. arXiv:cs.LG/1711.00740
[2] Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, et al. 2018. Construction of the literature graph in semantic scholar. *arXiv preprint arXiv:1805.02262* (2018).
[3] Davide Buscaldi, Danilo Dessì, Enrico Motta, Francesco Osborne, and Diego Reforgiato Recupero. 2019. Mining scholarly publications for scientific knowledge graph construction. In *European Semantic Web Conference*. Springer, 8–12.
[4] Natthawut Kertkeidkachorn and Ryutaro Ichise. 2017. T2KG: An End-to-End System for Creating Knowledge Graph from Unstructured Text. In *AAAI Workshops (AAAI Workshops)*, Vol. WS-17. AAAI Press. http://dblp.uni-trier.de/db/conf/aaai/aaai2017w.html#KertkeidkachornT17
[5] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. 2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. *arXiv preprint arXiv:1808.09602* (2018).
[6] Makoto Miwa and Mohit Bansal. 2016. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 1105–1116. https://doi.org/10.18653/v1/P16-1105
[7] Anderson Rossanez and Julio Cesar dos Reis. 2019. Generating Knowledge Graphs from Scientific Literature of Degenerative Diseases. (2019).
[8] Aditi Roy, Ioannis Akrotirianakis, Amar V. Kannan, Dmitriy Fradkin, Arquimedes Canedo, Kaushik Koneripalli, and Tugba Kulahcioglu. 2020. Diag2Graph: Representing Deep Learning Diagrams in Research Papers as Knowledge Graphs. In *IEEE International Conference on Image Processing*.
[9] Guus Schreiber and Yves Raimond. 2014. RDF 1.1 Primer. *W3C Working Group Note* 25 (2014).
[10] Shih-Yuan Yu, Ahmet Salih Aksakal, Sujit Rokka Chhetri, and Mohammad Abdullah Al Faruque. 2020. *Deep Code Curator – code2graph Part-II.* Technical Report TR-20-01. Center for Embedded and Cyber-Physical Systems University of California, Irvine, Irvine, CA 92697-2620, USA. http://cecs.uci.edu/files/2019/05/TR-19-01.pdf

[8]https://paperswithcode.com/
[9]https://github.com/deepcurator/DCC/tree/master/demo/run_queries
[10]https://github.com/deepcurator/DCC/blob/master/queries.py