**DARPA ASKE DCC – Milestone 8, 2019 (Month 12)**

Authors: Akrotirianakis Ioannis, Amar Viswanathan Kannan, Fradkin Dmitriy, Roy Aditi, Tugba Kulahcioglu, Canedo Arquimedes, Mohammad Al Faruque

**Contents**

[1. Introduction 2](#_Toc23433377)

[2. Run all modalities and generate knowledge graphs 2](#_Toc23433378)

[2.1 Deep Code Curator Demo Installation 2](#_Toc23433379)

[*2.1.1* Create the Python environment 2](#_Toc23433380)

[*2.1.2* Grobid installation 3](#_Toc23433381)

[*2.1.3* Tesseract 3](#_Toc23433382)

[*2.1.4* Other files 4](#_Toc23433383)

[2.2 Notebook description for running all modalities 4](#_Toc23433384)

[*2.2.1* Run text2graph 5](#_Toc23433385)

[*2.2.2* Run code2graph 5](#_Toc23433386)

[*2.2.3* Run image2graph 5](#_Toc23433387)

[3. Deep Code Curator - Queries demo notebook 6](#_Toc23433388)

[3.1 Setup Python environment 6](#_Toc23433389)

[3.2 Virtuoso installation 6](#_Toc23433390)

[3.3 Download and import data 6](#_Toc23433391)

[3.4 Run\_queries demo notebook 7](#_Toc23433392)

# Introduction

The purpose of this report is to provide an overview of the work that has been done during **October 1st** to **October 31st, 2019**. The report is based on the feedback obtained by the DARPA Program Manager, Dr. Joshua Elliott, during the demo of our system during October 18, 2019.

Currently, our [demonstration](https://github.com/deepcurator/DCC/tree/master/demo) consists of 2 jupyter notebooks. In the [first notebook](https://github.com/deepcurator/DCC/blob/master/demo/run_all_modalities/Deep%20Code%20Curator%20(DCC).ipynb), we have updated the documentation and organized our multimodal knowledge graphs (text2graph, image2graph and code2graph) in a sequential fashion so that they can be run easily and efficiently.

In the [second notebook](http://localhost:8888/notebooks/run_queries/Queries.ipynb) we have create several queries that show the value of the knowledge contained in the merged Knowledge Graph and obtained from the three modalities – text2graph, image2graph and code2graph.

In should be noted that each notebook comes with detailed ReadMe files that provide guidance on how to install different 3rd party packages needed for the execution of the notebooks.

In the remaining of this report we provide more details about the demo installation and the individual notebooks.

# Run all modalities and generate knowledge graphs

In this section we describe how to run all the modalities in a sequential mode. We provide details how to install the environment required and the different steps involved during the execution of the modalities.

## Deep Code Curator Demo Installation

### Create the Python environment

To create the Python environment needed to run the demo the file dcc.yml has to be used. The following steps are recommended:

* Run  conda env create -f dcc.yml in a **Windows Command** window. The reason we recommend to use Windows Command window is that some versions of Conda Command window (with Anaconda command prompt) has a bug installing 'pip' requirements. If you get the error '*conda not found*', add the following lines (or corresponding locations from your computer) into your 'path' system variable:
  + C:\Users\YOUR\_USERNAME\AppData\Local\Continuum\anaconda3
  + C:\Users\YOUR\_USERNAME\AppData\Local\Continuum\anaconda3\Scripts
* Switch to Anaconda command prompt and activate the environment typing: activate dcc
* Add your new environment to Python ipykernel by running the following command
  + python -m ipykernel install --user --name=dcc
* Try running jupyter notebook using: jupyter-notebook.
* Note: If you run into a "*DLL not found*" error during the above steps, run the following commands:
  + pip uninstall pyzmq,
  + pip install pyzmq
  + and then try the step again.

### GROBID installation

GROBID[[1]](#footnote-2) is an open source machine learning library for extracting, parsing and restructuring raw documents such as PDFs into structured XML/TEI encoded documents with a particular focus on technical and scientific publications. We use GROBID in the text2graph module to conveniently and efficiently extract the text from Deep Learning papers.

#### GROBID Server

GROBID needs Gradle[[2]](#footnote-3) to be built, and based on our tests we found Gradle 4.10 and Grobid 0.5.5 works smoothly together, and hence we recommend using these versions. The following steps should be followed:

* Install [gradle-4.10](https://gradle.org/next-steps/?version=4.10&format=bin) using the instructions [here](https://docs.gradle.org/current/userguide/installation.html)
* Download and extract [grobid-0.5.5](https://github.com/kermitt2/grobid/archive/0.5.5.zip)
* cd into grobid-0.5.5, and run the command gradle clean install
* At this point, Gradle server is built. Before running the demo, make sure to run gradle run in the grobid-0.5.5 folder to start the server.

#### GROBID client

Download [Grobid client](https://github.com/kermitt2/grobid-client-python/archive/master.zip) (the whole project as a zip) and extract it to a folder of your choice. You will be specifying its path as an input to the [demo notebook](https://github.com/deepcurator/DCC/blob/master/demo/run_all_modalities/Deep%20Code%20Curator%20(DCC).ipynb).

### Tesseract

[Tesseract](https://github.com/tesseract-ocr/tesseract) is an open source Optical Character Recognition library. In our project it isused by the image2graph module.

For Windows only: Tesseract needs to be installed using the installer from the following link: <https://github.com/UB-Mannheim/tesseract/wiki> Please make sure that the following line of code is uncommented in the notebook, and has the correct path to the tesseract executable in your local

* pytesseract.pytesseract.tesseract\_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

### Other files

Additional files needed to run the demo are available through the Open Science Foundation[[3]](#footnote-4) (OSF) in the corresponding [OSF project](https://osf.io/jdhw8/). You can click on the "demo" and then download it as a zip file to obtain all required files at once.

#### PDFFigures 2.0

[PDFFigures 2.0](https://github.com/allenai/pdffigures2) is used by the image2graph module to extract the images from PDF files. To make its installation more convenient, we provide compiled jar files in the zip named pdffigures.zip. These jar files need to be placed **directly under the dcc\demo\run\_all\_modalities folder**, without any additional folders e.g. dcc\demo\run\_all\_modalities\pdffigures2\_2.12-0.1.0.jar

#### Models

Model files for text2graph (text2graph\_models.zip) and image2graph (image2graph\_models.zip) modules are provided. These zips need to be extracted into seperate folders of your choice, and their paths need to be updated in the corresponding lines from the [demo notebook](https://github.com/deepcurator/DCC/blob/master/demo/run_all_modalities/Deep%20Code%20Curator%20(DCC).ipynb) as shown below:

text2graph\_models\_dir = "YOUR\_PATH\_TO\_THE\_TEXT2GRAPH\_MODELS\_DIR"

image2graph\_models\_dir = "YOUR\_PATH\_TO\_THE\_IMAGE2GRAPH\_MODELS\_DIR"

#### Ontology

Download and place the ontology file named DeepSciKG.nt into a folder of your choice and update its path in the corresponding line (given below) from the [demo notebook](https://github.com/deepcurator/DCC/blob/new_demo/demo/run_all_modalities/Deep%20Code%20Curator%20(DCC).ipynb) .

ontology\_file = "YOUR\_PATH\_TO\_THE\_ONTOLOGY\_DIR\DeepSciKG.nt"

## Notebook description for running all modalities

The notebook that runs all modalities and generates the corresponding graphs can be found in our [GitHub repo](https://github.com/deepcurator/DCC/blob/master/demo/run_all_modalities/Deep%20Code%20Curator%20(DCC).ipynb).

Initially all the necessary packages and modules are imported. Next the user needs to create two input directories and one output directory. Those directories can be simply created in the directory where the notebook exists, or in any other place as long as the complete path is specified. More specifically,

* inputFolder/, is the first input directory and contains the PDF file (of files) that will be processed by text2graph, image2graph.
* code\_repository\_path/ is the second input directory and stores the source code of the algorithm described in the PDF paper
* outputFolder/ is the output directory and it stores the KGs of each individual modality.

In addition, a number of dependency files and directories need to be specified

* ontology\_file contains the merged ontology
* text2graph\_models\_dir contains the statistical models for Named Entity Recognition (NER) and Relation Extraction (RE), used to generate the KG based on the information extracted from the text of the PDF.
* Image2graph\_models\_dir contains the deep neural network models used to recognize and classify different deep learning architectures defined by images/plots within the PDF paper.

### Run text2graph

In that cell we run the entire text2graph pipeline (described in previous Milestone reports). In particular, the following steps are performed:

* Using Grobid we extract the XML format of the PDF paper
* From the XML we then can select any part of the paper and extract its text – currently we select the Abstract of the paper
* We use the extracted text as input for our NER and Relation Extraction statistical models to generate the entities and relations.

The entities and relations are used to generate the RDF graph, text2graph.ttl, based on the text modality, and stored in the demo\_output/text2graph/ folder.

### Run code2graph

In this cell the Lightweight Approach is demonstrated. In particular, a static AST-based analysis of code is performed, which can allow us to acquire graphs without actually executing the source code that corresponds to the PDF paper we analyze. The Python modules ast and pyan help us extract the ASTs from Python programs and generate the call graphs. Afterwards, we generate tree-like structures (known as Call Trees) that contain both sequential and hierarchical information, needed for the creation of the RDF graphs of the main components of the source code. The generated RDF graph is stored in the code2graph.ttl file which resides in the outputFolder/ directory. In addition, a visualization of the graph is created for each source Python file in the GitHub repository where the source code exists. For simplicity we only show the visualization of the graph for one file, but the user can specify a different filename and visualize other files in the source code.

### Run image2graph

The entire pipeline for the image2graph is run in this cell. In particular the following main steps are performed

* All the images in the PDF paper are extracted
* Initially these images are classified in two main classes: (i) images that describe Deep Learning models/architectures, and (ii) other images (e.g., performance comparisons of different methods). Next, the images describing DL methods are classified to five different categories which are mostly used in the DL literature (2D Box, Stacked2D Box, 3D Box, Matrix Box, Neurons plot, Pipeline plot)
* The identified DL images are analyzed by performing the following steps: (1) identification of the building blocks, (2) identification of the text present around those blocks, (3) identification of the edges connecting the nodes, and finally (4) determination of the spatial and logical relations among the nodes, edges and text candidates in order to generate the final interpretation of the DL architecture
* After detecting deep learning architecture and the flow of the computations within it, a computational graph is created with the following relationships describing each node or pair of nodes: “is type”, “has description”, “connected to”, “followed by”, “has input”, and “has output”.

# Deep Code Curator - Queries demo notebook

## Setup Python environment

The instructions are identical as those described in section 2.1.1.

## Virtuoso installation

Virtuoso is the database used to store the graph data for DCC. Below we provide a summary of the installation steps. For detailed instructions, please check [the corresponding Virtuoso page](http://vos.openlinksw.com/owiki/wiki/VOS/VOSUsageWindows).

* Pre-built binaries of Virtuosofor Windows require the Microsoft Visual C++ 2012 Redistributable Package. If you do not already have it in your system, it can be downloaded from [the corresponding Microsoft page](https://www.microsoft.com/en-us/download/details.aspx?id=30679).
* Next, [download](https://sourceforge.net/projects/virtuoso/files/virtuoso/7.2.5/Virtuoso_OpenSource_Server_7.20.x64.exe/download) and install the pre-built Virtuoso (for Windows).
* Setup the necessary environment variables. Determine the location for your Virtuoso installation (e.g. C:/Program Files/OpenLink Software/VOS7/virtuoso-opensource/). Using this path, create a new system environment variable called VIRTUOSO\_HOME.
* Finally, add the following string to the end of the existing PATH system variable: ;%VIRTUOSO\_HOME%/bin;%VIRTUOSO\_HOME%/lib.

## Download and import data

We provide the data file consolidated.ttl through the demo\_queries/ folder in the corresponding [OSF project](https://osf.io/jdhw8/). After you download this file into your computer, the following steps will guide you to import it to the Virtuoso:

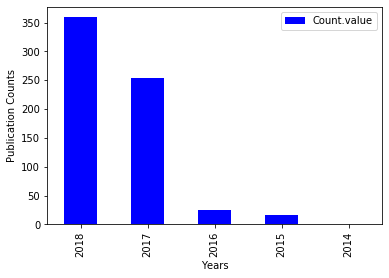
* Open <http://localhost:8890/conductor> in your Web browser. If you do not use your localhost for Virtuoso, or if you use another port, update the address accordingly.
* Enter account username dba and password dba to login.
* Navigate to the window Linked Data → Quad Store Upload
* Choose the downloaded consolidated.ttl file to upload, and specify the Named Graph IRI as https://github.com/deepcurator/DCC/. Then click 'Upload".
* After seeing the upload completed message, click on the "Graphs → Graphs" tabs to verify the existence of the uploaded database. You would be seeing the new named graph (https://github.com/deepcurator/DCC/) in that list.

Now you can start using the [Queries demo notebook](http://localhost:8888/notebooks/run_queries/Queries.ipynb). You will need to update your Virtuoso address in the notebook if you use something other than the default http://localhost:8890.

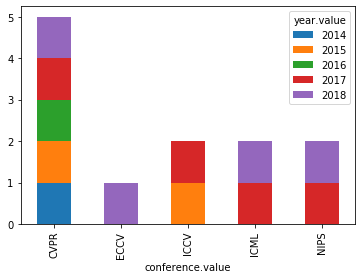
## Run\_queries demo notebook

In out [Queries demo notebook](http://localhost:8888/notebooks/run_queries/Queries.ipynb) we want to demonstrate the value of the knowledge extracted from the Knowledge Graphs (KG) that is created by merging the individual KGs obtained from the three modalities – text2graph, image2graph and code2graph. We should note that this is our current focus and more queries will be added in the near future. The users can create their own queries following the example queries we have provided. We have used SPARQL which is a query language and can retrieve and handle data stored in RDF format. SPARQL allows us to generate queries that consist of triple patterns, conjunctions, disjunctions, and optional patterns.

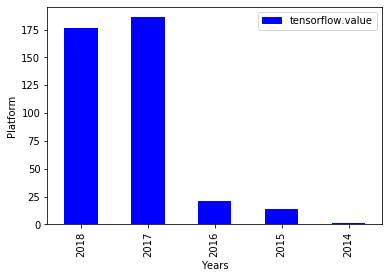
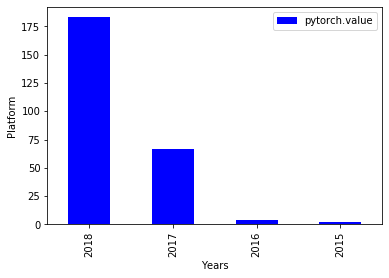
The first query we have implemented tries to find the number of Deep Learning publications organized by year across all conferences[[4]](#footnote-5). As can be seen by the plot, the number of publications in those conferences increases dramatically.



The second query determines the number of Deep Learning publications and sorts them by conference series and year of publication. As can be seen CVPR contains Deep Learning papers for all years we consider in our study (2014-2018), On the other hand, in the dataset we consider the ECCV conference is only included from 2018.



The third query we formulated tries to identify trends between the two mostly used Deep Learning platforms - TensorFlow and PyTorch. As can be seen from the plot, TensorFlow has been the leader platform in the DL community. However, PyTorch has been consistently gain acceptance and in 2018 reached a almost equal number of developers as TensorFlow.



1. https://github.com/kermitt2/grobid [↑](#footnote-ref-2)
2. https://gradle.org/ [↑](#footnote-ref-3)
3. https://osf.io [↑](#footnote-ref-4)
4. currently we consider the following conferences: NIPS, ICML, CVPR, ECCV, but we plan to incorporate more conferences in the future. [↑](#footnote-ref-5)