Linking Knowledge Graphs and Images Using Embeddings

Carl Edwards

Abstract

Knowledge graphs and images offer contrasting yet complementary sources of information. Images offer visuospatial data lacking in the structure of knowledge graphs. By integrating these two data types, their complementary information can improve the ability of various prediction tasks. In this report, I investigate creating an aligned image-knowledge graph dataset and the implementation of a joint-embedding model using the KADE method. I use the Inception-resnet v2 architecture to create an image embedding. I also use TransE for experiments involving knowledge graph embeddings.

Introduction

As the volume of collected data continues to grow, tools such as large knowledge graphs (KGs) have been created to provide knowledge about relationships between real-world concepts. The underlying structure of a KG consists of entities which are connected by various relationships; for example, a commonly given example would be the link between Douglas Adams and his book *Hitchhiker's Guide to the Galaxy* via the relationship "Author of." These relationships are relevant for applications such as querying. However, KGs lack other types of complementary information, such as visual data. Visual information can provide useful properties to complement the entity-relationship triples present in a KG. However, linking visual information to knowledge graphs without human input is difficult due to the wide variety of concepts covered in modern knowledge graphs and the dissimilarity between data types. Embeddings are a useful tool for representing high-dimensional and potentially sparse data in a condensed manner. Embeddings are low-dimensional translations of higher dimensional data which seek to retain the semantic property of the data in a dense low-dimensional vector. Commonly used examples include Word2Vec [1] and TransE [2]. Embeddings allow data which might normally have a sparse representation like one-hot encoding, such as an entity in a knowledge graph or a word in a document, to be expressed in a compact yet semantically useful format.

Image data and knowledge graphs have previously been tied together in papers such as [3] and [4], who create multi-modal spaces by concatenating unimodal embeddings, process these using various methods, and then measuring semantic similarity between embeddings using semantic relatedness tasks such as WordSim353 and MEN. In addition, works such as [5] attempt to combine images with descriptive sentences using embeddings.

In order to obtain a dataset for this project, I investigate two commonly used KGs: DBPedia and Wikidata. I initially elected not to focus on Freebase due to its retirement. In addition, I investigated image databases which could potentially be linked to a knowledge graph. I was unable to find any dedicated, human-verified joint image-knowledge graph datasets. Imagenet [6] stood out as a likely possibility since it was based on Wordnet [7], which is a large lexical database of "synsets" that group together equivalent meaning words. Imagenet is a very popular and important benchmark dataset. The mapping of images to synsets as well as the popularity of the dataset made it desirable to use for this task. To this effect, I attempted to create a dataset connecting Imagenet and Wikidata or DBPedia.

This report investigates the connection between knowledge graphs and images. Particularly, it focuses on the joint creation of embeddings of aligned images and knowledge graphs. This allows the preservation of the very different semantic information contained in images, which contain visuospatial information, and that of knowledge graphs, which structure information graphically. The project investigates the application of the KADE method [13] to this problem of tying the very differing data structures of KGs and images together.

Dataset

I initially investigated DBPedia, since many of its entities are linked to Wordnet 2.0. I used official Wordnet mappings to convert this to Wordnet 3.0, which Imagenet is based on. According to [8], the quality of mappings is reasonably well-preserved. Finally, these are mapped to Imagenet. I also investigated linking Wikidata to Imagenet by means of the YAGO ontology [9]. This is a manually verified ontology which provides mappings including connecting Wikidata entities to Wordnet 3.0 synsets. In addition, Imagegraph [10], a knowledge graph which consists of search engine scraped images mapped to FB15K was downloaded for the purpose of developing and testing the model.

a. Imagenet

Imagenet [6] is an extremely popular image classification database based on Wordnet, which relates the meaning of words. Imagenet provides tens of thousands of image files in various categories. In particular, the Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) along with the debut of Alexnet [14] has resulted in a widely used benchmark dataset, Imagenet1k, containing 1000 image categories. This dataset is used both as a benchmark for classification and to train new models on for transfer learning. In Imagenet, images are sorted into underlying Wordnet [7] synsets, which are groups of related words. An example of this might be “timber” and “lumber” are in the same category. The Wordnet basis allows the image database to be adapted to language-based datasets more easily. The underlying linguistic backbone along with the large image database and historical popularity of the dataset make it a strong candidate for being mapped to knowledge graphs. Unfortunately, the authors of Imagenet failed to respond to attempted communication for the purpose of downloading the original dataset. Thus, images were downloaded with wget directly from their URLs, many of which were outdated and broken links. From the URLs that were reachable, roughly 75.2% of the images that were downloaded were actual image files.

b. DBPedia

DBPedia is a knowledge graph based on many Wikipedia projects, primarily through extraction of Wikipedia infoboxes. It is frequently used as a dataset for knowledge graph applications, such as embeddings. I initially worked to connect DBPedia to Imagenet due to its connections to Wordnet 2.0. I obtained a file containing the Wordnet 2.0 links in the 2017 release of DBPedia. DBPedia maps to 124 synsets consisting of approximately 425,000 entities. URLs for the DBPedia entities and the Wordnet synsets were extracted from the linking file. I take the DBPedia entity name directly from its URL, and I access the Wordnet URL hosted by w3 in order to obtain the Wordnet ID (wnid). Next, the Wordnet 2.0 wnid is mapped to Wordnet 2.1. using the sensemaps published online by Wordnet. Due to the noun-only structure of Imagenet, I only mapped nouns. The polysemous mappings consisted of several potential mappings to new synsets with an assigned confidence in the mapping. Among the synsets that had polysemous mappings, I randomly used one of the potential mappings which had above a 10% confidence. This resulted in the majority of the synsets being mapped. 70% of the desired synsets mapped between version 2.0 and 2.1 were monosemous, and the rest were mapped when polysemous mapping was introduced. I followed the same procedure to map Wordnet from 2.1 to 3.0, and I find that the same amount of synsets map monsemously. Selecting the highest confidence mapping among polysemous synsets provides the most semantic coverage. However, the final entity file only has 425,008 entities mapped to Wordnet before being mapped to Imagenet specific synsets, so I instead focus on Wikidata, which had a considerably larger amount of mapped entities.

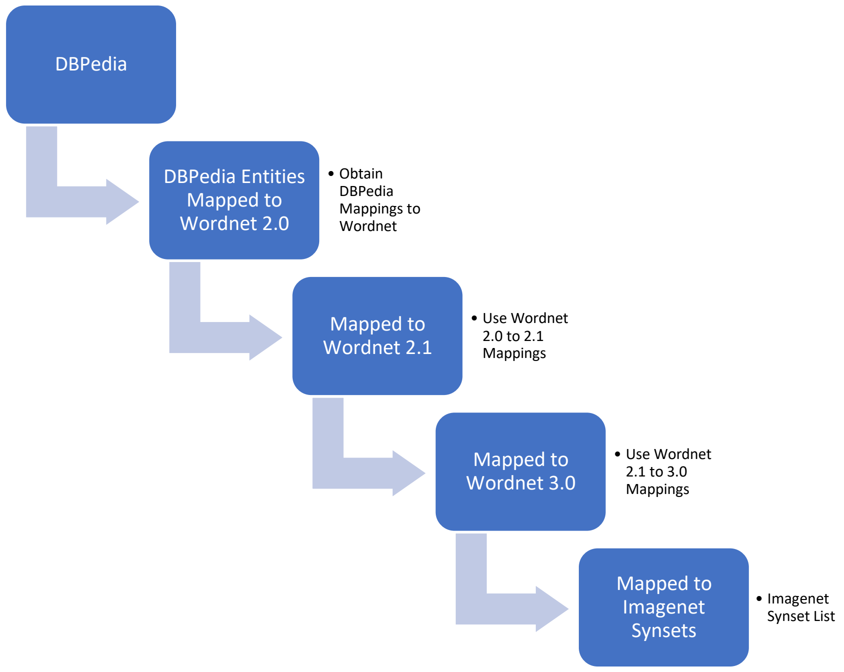
DBPedia
DBPedia Entities Mapped to Wordnet 2.0
• Obtain DBPedia Mappings to Wordnet
Mapped to Wordnet 2.1
• Use Wordnet 2.0 to 2.1 Mappings
Mapped to Wordnet 3.0
• Use Wordnet 2.1 to 3.0 Mappings
Mapped to Imagenet Synsets
• Imagenet Synset List

Figure 1 - Steps for mapping DBPedia and Imagenet

c. Wikidata

The technique applied to Wikidata can be applied to the mapping created above as well. Wikidata is a knowledge graph which is part of the Wikimedia foundation. In order to map this dataset to imagenet, I used the YAGO ontology [9]. In particular, this required several mappings from YAGO. First, I extracted the YAGO simple types containing Wordnet mappings. This resulted in roughly 3.1 million mappings. Next, I mapped these to YAGO's Wikidata instances. This mapping contains simple YAGO types which map to both Wordnet and Wikidata. Next, I map the simple type to the Wordnet ID (wnid) rather than a synset name and restrict it to Imagenet synsets. I use two files from Imagenet's website which provide a list of synsets. These files differ, but my tests show that they ultimately contain the same synsets and produce no difference in the final result. This yields a graph containing roughly 1.6 million entities. I use this file and replace the simple types with QIDs, creating a mapping from the Wikidata QID to the Imagenet synset.

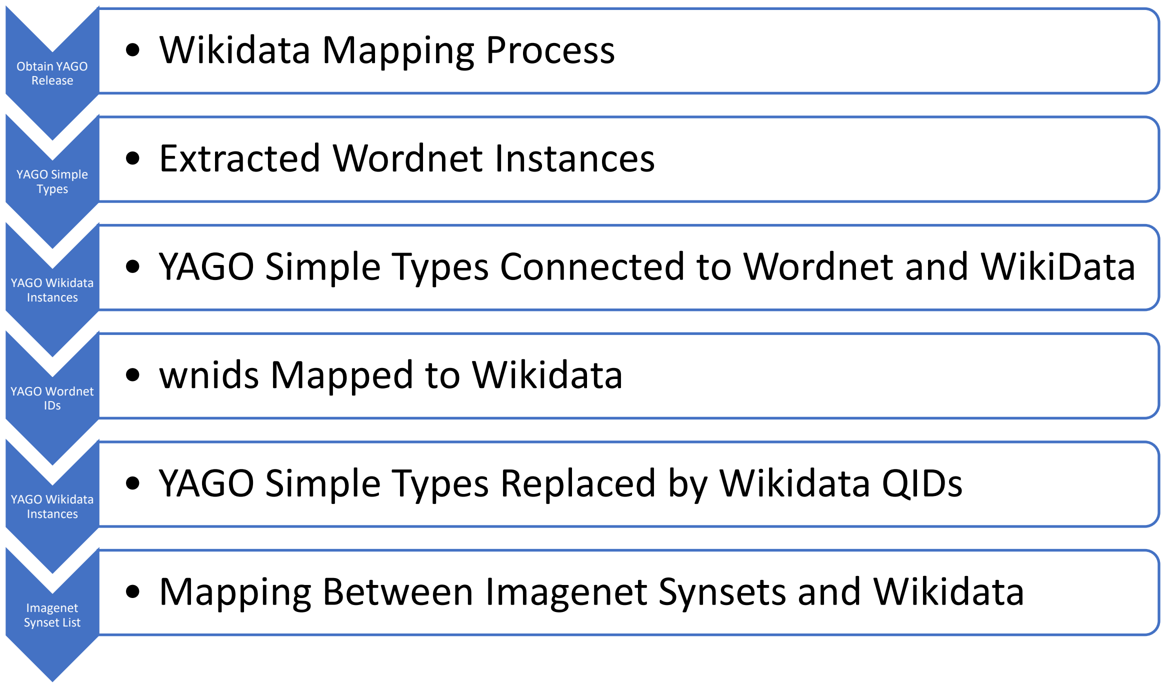
Obtain YAGO Release
• Wikidata Mapping Process
YAGO Simple Types
• Extracted Wordnet Instances
YAGO Wikidata Instances
• YAGO Simple Types Connected to Wordnet and WikiData
YAGO Wordnet IDs
• wnids Mapped to Wikidata
YAGO Wikidata Instances
• YAGO Simple Types Replaced by Wikidata QIDs
Imagenet Synset List
• Mapping Between Imagenet Synsets and Wikidata

Figure 2 Steps for mapping Wikidata and Imagenet

Following the creation of this list of files, the 1.6 million entities mapped to synsets are downloaded. These entities are downloaded by directly querying the online SPARQL endpoint using Gastrodon. Gastrodon is a toolkit used for working with RDF graphs and SPARQL queries. I queried the Wikidata query endpoint using the SPARQL query seen in Figure 3 which is designed to find the links to other Wikidata entities. It ignores statements, which are not a useful feature for the graph, and other languages besides English. Finally, I store these triples with the specific Wikidata entity as a head in an individual file. Each is stored in its respective synset's directory. Note that relationships where the entity is the tail are not queried; these will be stored if the head entity of that relationship is also linked to Imagenet.

```
SELECT ?pred ?obj WHERE {
  wd:Q42 ?pred ?obj .
  FILTER(STRSTARTS(xsd:string(?obj), "http://www.wikidata.org/entity/"))
  FILTER(!STRSTARTS(xsd:string(?obj), "http://www.wikidata.org/entity/statement")
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
}
```

Figure 3 SPARQL query for downloading a subgraph of Wikidata entities. This particular query returns all triples where Q42 is the head and another Wikidata entity is the tail.

Next, I process the triples I have downloaded and extract a non-disjoint subgraph to provide a suitable environment for embedding models. I create a subgraph from these entities using the Gastrodon Python toolkit. Unless otherwise stated, code and libraries used in this work are implemented using Python 3.6. I first parsed the file structure to obtain all the entities. Next, I eliminated synsets lacking any pictures downloaded. Although there may be pictures in

Imagenet, the download was incomplete due to downloading via old URLs. I then loaded the entities into a complete knowledge graph, which contains 596,743 entities of which 290,394 are the subject of a triple (have a synset). Next, I trimmed this graph to only entities which exist in my mapping of QIDs to synsets. Finally, I use the following algorithm to create a non-disjoint subgraph of this data to create embeddings with. The property path syntax in SPARQL queries returns connected entities. Specifically, "+" is used to find other entities connected to an entity via one or more paths. However, it only finds connected entities along the same relationship, so to circumvent this I use an identical placeholder relationship "<>". A copy of the trimmed graph is created and all the relationships in the triples are replaced with the identical placeholder. This allows the use of the "+" in the SPARQL query. This modified copy of the trimmed graph is uploaded to blazegraph. This is necessary because the following query using "+" is often implemented recursively and fails in various Python-based implementations such as the Gastrodon toolkit using RDFlib. The SPARQL query returns all interconnected entities. I select a random entity and run this SPARQL query on it, which returns a list of connected entities—my subgraph. Entities that are assigned a subgraph are marked. A new random, unassigned entity is selected and its subgraph is queried. The entities in the placeholder graph are iterated over until they are all assigned a subgraph. To select a subgraph the following heuristics are performed: first, the largest subgraph is selected. Second, from subgraphs of identical size, the subgraph with the most triples is selected. Finally, the selection is random.

```
prefix wd: <http://www.wikidata.org/entity/>

prefix wdt: <http://www.wikidata.org/prop/direct/>

prefix : <>

SELECT * WHERE {

  wd:Q1131681 <>+ ?o

}
```

Figure 4 SPARQL query for identification of subgraphs

Two list of of Imagenet synsets were used: an XML file of synsets from the fall 2011 release consisting of 428 unique Wikidata-mapped synsets and another file of synsets provided by ImageNet on its download page which contains 280 unique synsets. Some of these synsets, however, contain no pictures upon downloading. I compared the useful synsets and found that after several steps of mapping, they eventually became equivalent, so I used the former list for my mapping.

The trimmed graph contains 141,255 triples. After creating the subgraphs, the largest subgraph contains 55,360 entities and 126,101 triples, which represents 63 synsets. This result is fairly sparse. This subgraph was used to create training-test splits. In addition, the subgraph was formatted so that OpenKE can be used to run TransE on the data.

TransE obtains a best mean rank of 2576.132812 after r filtering given a random 70-30 training test split. This mean rank result matches those of TransE when applied to other sparse knowledge graphs such as DB50. Several varying train-test splits were used, and TransE had similar results on various data splits. There are not enough relationships between the entities to provide the information to create embeddings that are as effective as those from dense graphs. It is likely that there are entities with many connections and others with very few. This results in a test-train split where an entity may only be present in one or the other since it only has one connection. This dataset provides a sparse graph mapped to Imagenet for testing embedding or other approaches which may require graph sparsity.

d. FB15K and Imagegraph

FB15K is a commonly used benchmark knowledge graph used for evaluating KG embedding methods. It is a subset of the Freebase knowledge graph, which is based on Wikipedia; however, Freebase is no longer supported due to the creation of Wikidata. However, the small benchmark dataset FB15K is still widely used for benchmarking embedding models. I initially focused my investigation of image-KG linking on DBPedia and Wikidata. In addition, I also focused on finding a dense graph for testing. I found a recent paper [10] with a dataset using the converse method: it uses the existing FB15K graph and scrapes images from search engines to match it. This resulted in roughly ~920,000 images of which ~600,000 were viable images spread among 14,854 entities, covering the large majority of FB15K. These images are not human verified and may be ambiguous, but it provides a dense, benchmark knowledge graph which can be used to test potential embedding models. In [10], Oñoro-Rubio et al. use the VGG image classification model to create embeddings which they train (starting with pretrained weights) using the TransE loss function by using the FB15K relationships to describe triples between the images. They attempt to answer various types of basic matching SPARQL queries using images.

Model

The goal of the model is to create a joint embedding which links images and knowledge graph entities. The model uses the KADE method [13] to create these embeddings in an iterative process. The model seperates into two components: a knowledge graph embedding such as TransE and an image embedding model. The goal of the model is to iterate between the two embedding methods and use regularizers to create a joint embedding space.

Image embeddings are typically extracted from existing image classification networks. Popular networks include the variety of Inception networks [11] and VGG [12]. The Inception-Resnet v2 convolutional neural network image classification model using pretrained weights is used to create embeddings in this report.

a. Inception-resnet v2

This image classification model [11] is a fusion of the resnet architecture originating from Microsoft [15] with Google’s Inception architecture. It has obtained state of the art results on the ILSVRC. The model uses residual blocks where some connections skip layers in order to avoid the vanishing gradient problem and create deeper networks. Resnet models also train in fewer epochs than previous inception models. By using this state of the art model, we can obtain embeddings which appropriately represent images. In addition, this architecture produces a 1,536-dimensional embedding which is smaller than the 2,048-dimensional embeddings produced by earlier Inception models.

The model has been implemented in both tf.slim and Keras; it has weights that have been pretrained with ILSVRC 2012 for both implementations. The Keras implementation of Inception-resnet v2 was used due to its implementational simplicity. Using Keras, the network can be trained by using data generators to feed the image data. To obtain the embedding, the last categorizing softmax layer was removed from the network, causing it to output an embedding from the dense 1,536-dimensional layer from before the softmax layer. Similarly to [10], another layer was added as a smaller embedding with 256 dimensions. A smaller embedding reduces the quality of the image embedding, but it matches the typical embedding sizes used in approaches such as TransE. This greatly increases the time required to create embeddings using those models. In tests on the Wikidata graph, this time difference was on the order of changing from 20-30 minutes for a 100-dimensional embedding to nearly six hours for the 2,048-dimensional one. In order to use the typical categorical cross-entropy cost function along with the unlabeled images from Imagegraph, another softmax layer with 1,000 categories is added to this small embedding layer. To obtain the labels for the web-scraped images, the labels were computed using the pretrained resnet implementation. This allows the smaller embedding layer to be trained using the same categorical cross-entropy loss function to compare against desired labels. In addition, this allows the main resnet network to be frozen, which significantly speeds up training time. The large embeddings of the images are pre-computed by the resnet and then are given as input to the new embedding layer. The resulting performance increase is significant. On a typical laptop processor, the smaller model computed over 1,000 times as many predictions in the same time that a single prediction of the resnet can be computed.

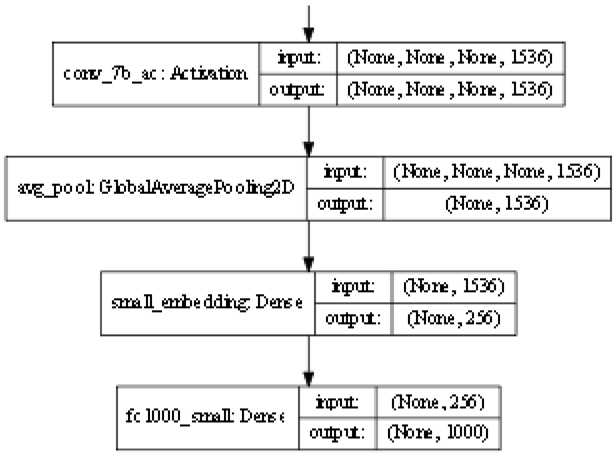

Figure 5 Final layers of Inception-resnet v2 with smaller embedding layer and softmax. The original softmax layer is removed from the network.

Keras is not able to feed input into intermediate layers of functional models. This is because the layer which should receive the intermediate input could potentially receive input from different sources in the model. In order to overcome this restriction, the new layers added to the model are sequential. This allows them to be incorporated into a new sequential model. The following code shows how this short model is created. It uses the functional model, `small_softmax_model`, built onto the existing resnet implementation and accesses the two new layers at the end (the small embedding and new softmax). An input layer is not required for the sequential model. I have tested training the network using both the functional model—the full network consisting of the resnet and additional embedding layers—and the sequential model—the additional embedding portion only. Training using either model successfully updates the network weights (of the small embedding portion), therefore affecting the output of either model. For training, the layers of the initial resnet model are frozen. After training, both models predict the same thing when given either image input or a pre-computed embedding input, for resnet and the small embedding model respectively. This indicates that the layers and their weights are shared between the two models and training can be performed on either if the embeddings of an image are pre-computed.

```
end_model = Sequential();

end_model.add(small_softmax_model.layers[-2]);

end_model.add(small_softmax_model.layers[-1]);

end_model.compile(optimizer='sgd', loss = SOME_LOSS_FUNCTION, metrics = ['accuracy'])
```

Figure 6 Code snippet of sequential model creration from existing functional model in Keras

b. KADE

KADE [13] is an embedding method which seeks to create aligned embeddings of KGs and documents. It uses a regularization process during the training of both embedding types to create a joint embedding space which seeks to preserve the properties of both original embedding methods used. It uses the original loss function of both embedding techniques in order to preserve their semantic properties and adds a regularization term to each loss function based on the difference between an embedding and the embedding of the aligned data type. KADE uses an iterative procedure to build the embeddings of both models concurrently, ensuring that each embedding technique has an influence on the resulting joint embedding space, preserving the properties of both techniques.

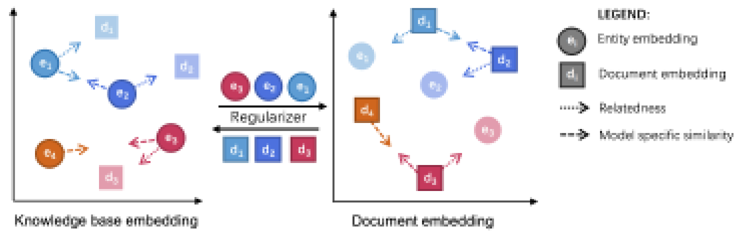

Figure 7 Illustration of KADE [13]

The Inception-resnet model can be implemented into the KADE system as a class. The Keras implementation uses Tensorflow as a backend. The modified resnet model is created during the construction of the class. These are compiled using a custom loss function which adds a regularization term based on matrix of image embeddings and KG embeddings. The initial embeddings matrix is created and added to the Tensorflow graph. For training, a custom data generator implementation is used to load data to feed Keras' fit_generator function which trains the network for the desired number of epochs. This allows the custom data order that KADE requires in order to switch between training the two embedding techniques.

In order to implement KADE's regularization function into the loss function, a Keras regularizer was initially considered. Keras has three types of regularizers: kernel, activation, and bias. The kernel regularization is a replacement for weight regularization (and essentially does the same thing). A regularizer can be a function or a callable class. It receives a tensor argument consisting of the layer's weights, activations, or bias. It also returns a tensor. Unfortunately, this regularization function is only called once; the tensor value it returns is incorporated into the Tensorflow graph. Thus, other code cannot be run or calculated from functions in the regularizer function.

It is also possible to implement custom loss function to be used by Keras when a model is compiled. Like the regularization function, this is only called to create a tensor object for the graph. As an example, it is possible to take a copy of Keras' categorical cross-entropy function and add a regularizer to it. Keras calls the Tensorflow backend version of the function, so a tensor object can be added to that. An untested method to implement the KADE regularization is to have the two embeddings matrices as instance variables in a class. Then, in the loss function compute the desired regularization and add it to the typical loss. Add the other tensors used to the Tensorflow backend so the regularization term continues to be updated.

Conclusion

This work has explored techniques for the creation of a joint embedding space between images and knowledge graphs using KADE. It examines the creation of a dataset for the task consisting of human-verified image classifications mapped to knowledge graphs. First, it investigates the linking of standard image datasets, particularly Imagenet, to knowledge graphs. Wikidata and DBPedia are both able to be mapped to Imagenet synsets. Wikidata is found to have a substantial number of entities that can be mapped to Imagenet. These entities are used to create a non-disjoint subgraph of Wikidata which is very sparse and shows poor results for the TransE model. In addition, this work also explores the use of a dataset created via search engine scraping (the Imagegraph dataset [10]) in order to provide an implementation of the proposed model. The image embedding is created using a smaller layer attached to the Inception-resnet v2 model. Due to time constraints on the project, this implementation has been unable to undergo testing. Future work can examine the hyperparameters of the model, test other embedding methods, and evaluate the performance of the model on other datasets.

During my time working on my project during my study abroad program, I have been fortunate to have had the opportunity to learn several new skills, consisting of both hard and soft skills that will surely be useful in my future career. I have been able to learn about embedding methods for a variety of data types, such as knowledge graphs, documents, images, and words. I have explored state of the art deep learning image classification models, have learned about the semantic web, RDF format, and SPARQL queries, and have also become acquainted with several important benchmark datasets such as Wikidata and Imagenet.

In addition to hard skills, I have also been able to better observe the lives of Ph.D. students and to have a better understanding of the requirements and challenges of obtaining the degree which will be useful when I enter graduate school myself. I have been able to experience the European academic environment, and I also was able to experience taking an oral exam for a class.

Acknowledgments

I have enjoyed the time I have been able to spend at the University of Zürich and am grateful for the opportunity to work with in Prof. Bernstein's DDIS group. I am also grateful to Matthias

Baumgartner for his mentorship and assistance on my project. I’d also like to thank Cristina Sarasua for her discussion on downloading Wikidata and the rest of the member of the DDIS group for their enriching discussion and interesting conversation.

References

1. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
2. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems* (pp. 2787-2795).
3. Thoma, S., Rettinger, A., & Both, F. (2017, October). Towards holistic concept representations: Embedding relational knowledge, visual attributes, and distributional word semantics. In *International Semantic Web Conference* (pp. 694-710). Springer, Cham.
4. Kiela, D., & Bottou, L. (2014). Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*(pp. 36-45).
5. Karpathy, A., Joulin, A., & Fei-Fei, L. F. (2014). Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems* (pp. 1889-1897).
6. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (pp. 248-255). Ieee.
7. Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, *38*(11), 39-41.
8. Kafe, E. (2017). How Stable are WordNet Synsets? *LDK Workshops*.
9. Suchanek, F. M., Kasneci, G., & Weikum, G. (2007, May). Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web* (pp. 697-706). ACM.
10. Oñoro-Rubio, D., Niepert, M., García-Durán, A., González-Sánchez, R., & López-Sastre, R. J. (2017). Representation Learning for Visual-Relational Knowledge Graphs. *arXiv preprint arXiv:1709.02314*.
11. Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI* (Vol. 4, p. 12).
12. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
13. Baumgartner, M., Zhang, W., Paudel, B., Dell'Aglio, D., Chen, H., & Bernstein, A. (2018, October). Aligning Knowledge Base and Document Embedding Models Using Regularized Multi-Task Learning. In *International Semantic Web Conference* (pp. 21-37). Springer, Cham.
14. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).
15. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).