Enhancing BIM Element Classification with a DNN hybrid model

Team Members:

Avishag Nevo - 324079763

Yogev Namir - 318880754

Introduction

This project focuses on improving Building Information Modeling (BIM) in the architecture, engineering, construction, and management sector by developing a deep learning-based method to automatically recognize and validate unknown BIM object information. Facing challenges like data conversion errors, human mistakes, and the use of incompatible objects, the current manual validation processes are inefficient. We propose a solution that merges CNNs and GNNs to enhance BIM object classification by leveraging both their geometric features and their contextual relationships within a building's design.

To be more concrete : We're developing a hybrid model that combines Graph Neural Networks and Convolutional Neural Networks for enhanced BIM object classification. This model harnesses CNNs to extract geometric features from 2D images of BIM objects (extracted from the objects 3D shapes) and GNNs to understand their contextual relationships within the building's layout (extracted from BIM projects).

The training involves two phases: first, the CNN learns from 2D rendered views to recognize geometric patterns (of 3D shapes), and then the GNN uses a graph representation of the building—where nodes are BIM objects and edges reflect their connections—to grasp the spatial and relational context. These two perspectives are integrated, possibly via a fusion layer, to accurately classify BIM objects by combining detailed geometry with contextual information, thus refining classification accuracy and providing deeper insights into model performance.

Shortly on BIM –

Outcomes brief -

We've successfully merged Convolutional Neural Networks and Graph Neural Networks to classify BIM objects. Despite having to work with very limited data and less computing power, our Hybrid Model still performed well, coming close to the results of models that had more training. These outcomes show that our approach works and that the model can effectively use the strengths of both CNNs and GNNs. This initial success sets the stage for further improvements.

Litrature Review

1. **Incorporating Context into BIM-Derived Data—Leveraging Graph Neural Networks for Building Element Classification** by Guy Austern, Tanya Bloch, Yael Abulafia, 2024:

- This study illustrates that GNNs, which incorporate contextual information through a graph data structure, outperform traditional machine learning models in classifying building elements.

- This paper directly supports the GNN aspect of our project by demonstrating the effectiveness of using contextual information for classification. The approach of constructing a graph dataset from BIM files and leveraging GNNs could guide the development of the GNN component in our model, especially in representing BIM objects and their relationships within the overall building structure.

2. **Multi-view Convolutional Neural Networks for 3D Shape Recognition** by Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, 2015:

- This paper demonstrates that CNNs trained on 2D images of 3D shapes can achieve high recognition rates, improving further with multiple views. It also introduces a novel CNN architecture that effectively combines information from multiple views into a compact shape descriptor.

- This research aligns closely with the CNN component of our project. The multi-view approach and architecture can be adapted to extract geometric features from BIM objects. Incorporating their methodology could enhance our CNN's ability to recognize BIM objects from 2D renderings of their 3D shapes, thereby improving the geometric feature extraction phase of our hybrid model.

3. **IFCNet: A Benchmark Dataset for IFC Entity Classification** by Christoph Emunds, Nicolas Pauen, Veronika Richter, Jérôme Frisch, Christoph van Treeck, 2021:

- The paper presents IFCNet, a dataset containing geometric and semantic information of IFC entities for BIM, showcasing that deep learning models can achieve good classification performance using geometric data.

-IFCNet can serve as an invaluable resource for training and evaluating our model. Access to a broad range of IFC classes with geometric information can help in creating and fine-tuning our CNN component and potentially the GNN component by providing a rich dataset that reflects the diversity of BIM objects.

4. **Recognizing and Classifying Unknown Object in BIM Using 2D CNN** by Jinsung Kim, Jaeyeol Song, and Jin-Kook Lee, 2019:

- It proposes a deep learning-based method for classifying building elements in BIM using 2D images, emphasizing the utility of CNNs for BIM object classification, including a practical application through a BIM plugin prototype.

- This research complements the CNN part of our project by showcasing an applied approach to classifying BIM objects using 2D images. The methodological insights and the practical demonstration of integrating the classification model into a workflow could inform how our hybrid model might be applied in real-world BIM applications and the development of related tools.

5.**SpaRSE-BIM: Classification of IFC-based geometry via sparse convolutional neural networks** by Christoph Emunds, Nicolas Pauen, Veronika Richter, Jérôme Frisch, Christoph van Treeck, 2022:

- Introduces SpaRSE-BIM, utilizing sparse convolutions for efficient IFC-based geometry classification, addressing computational overhead in BIM tools.

- The methodology can enhance our project by improving the CNN component's efficiency in classifying BIM objects, potentially offering a more computationally efficient approach to geometric classification. We most likely wont use it but it was an intresting moving forward approch.

6.**AN OPEN REPOSITORY OF IFC DATA MODELS AND ANALYSES TO SUPPORT INTEROPERABILITY DEPLOYMENT** by Robert Amor, Johannes Dimyadi, 2010:

- Highlights the importance of robust datasets for training BIM-related models and interoperability challenges.

- Access to such repositories (presented in the paper) could enhance our GNN component, providing relational data and insights for better handling IFC data models, supporting training and real-world application adaptability.

7.**ImageNet: A large-scale hierarchical image database** by Jia Deng; Wei Dong; Richard Socher; Li-Jia Li; Kai Li; Li Fei-Fei, 2009:

- Creation of a large-scale, annotated image database, showcasing a methodological approach to organizing and annotating images. The dataset containing more than 14 million training images across 1000 object classes. It is one of the top models from the ILSVRC-2014 competition.

- The principles of ImageNet guide us thorugh the creation and enhancement of BIM object image (IFCNetCore) datasets for our CNN component, ensuring diversity and comprehensiveness for improved model generalization.

Were using the VGG model as a base model for Single-View CNN (SVCNN). It is a convolution neural network (CNN) model supporting 16 layers. K. Simonyan and A. Zisserman from Oxford University proposed this model. The VGG model can achieve a test accuracy of 92.7% in ImageNet.

8.**Illumination for computer generated pictures** by Bui Tuong Phong, 1975:

- Provides foundational knowledge on rendering techniques, particularly shading and illumination models for computer-generated images.

- Understanding these rendering techniques aids in generating and augmenting training datasets with varied lighting conditions, enhancing the CNN's recognition capabilities under different visual scenarios, these techniques were used for creating the dataset for CNN.

9.**A geometric deep learning approach for checking element-to-entity mappings in infrastructure building information models** by Koo, B., Jung, R., Yu, Y. & Kim, I., 2020:

- The paper explores the use of geometric deep learning for BIM-to-IFC mappings, employing MVCNN and PointNet to classify infrastructure elements.

- Insights from this study informed the CNN component of our model, particularly the adaptation of multi-view learning approaches for the effective geometric data handling.

10.**Building Model Object Classification for Semantic Enrichment Using Geometric Features and Pairwise Spatial Relationships** by Ma, L., Sacks, R. & Kattel, U., 2017:

- Proposes a systematic approach to classify objects in BIM models using geometric features and spatial relationships for semantic enrichment.

- This methodology informed the development of our GNN component, especially in constructing and utilizing graph representations of BIM models for enhanced classification through spatial and relational context understanding altogether.

Methodology

(The end of this document concludes the whole latex for every file included so far in the project, generated with CHATGPT and monitored by us)

1. Introduction to Methodology

The methodology of our project centers on integrating Convolutional Neural Networks (CNNs) with Graph Neural Networks (GNNs) to enhance the classification of BIM (Building Information Modeling) objects. This hybrid approach is chosen for its ability to capture both the geometric features of BIM objects through CNNs and the contextual relationships within building structures through GNNs, thereby addressing the multifaceted nature of BIM object classification.

2. Data

2.1 Data Collection

Our dataset comprises 3D models of BIM objects collected from publicly available BIM repositories and industry collaborations, the full IFCNet dataset currently consists of 19,613 confirmed objects distributed over 65 classes, most of which are highly imbalanced with respect to the number of objects they contain. Therefore, a subset of 20 classes is selected for the experiments. The first version of this sub-dataset, called IFCNetCore, contains a total of 7,930 objects and that is processed data were using. The initial selection criteria, and the dataset chosen IFCNetCore included the diversity of object types, representation accuracy, and relevance to construction and architectural design, ensuring a comprehensive dataset that reflects the real-world complexity of building environments, and specifically BIM elements. The data jsons contained in ‘RevitBuildings’ directory we got from prof. austern and are real plans.

2.2 Data Preprocessing

For 3D objects, the code initially rendered these into 2D images from various viewpoints to capture geometric features effectively, using scripts like `make_mvcnn_data.py` for multi-view image generation. Graph representations were prepared by defining nodes as BIM objects and edges as adjacency relationships, employing `graph_splitter.ipynb` and `helpers.py` to encode spatial and relational context into graph structures.

2.3 Data Usage

Weve used the preprocessed data as data points, each in the relevant data type to fit the future model it is going to be ingested to, we have done this all in the ‘datasets.py’ file. For the Hybrid model dataset creation we have sampled visuals (from multiview dataset) for every element that war relevant and can be found in RevitBuildings data jsons. Data loaders are tailored for each model and are created in ‘mvcnn.py’, ‘arcgnn.py’, ‘hybrid.py’.

3. Model Development

3.1 Single and Multi view Convolutional Neural Network (CNN)

3.1.1 Architecture

The MVCNN (+ SVCNN) architectures detailed in ‘models.py’ leverages multiple convolutional layers, pooling layers, and fully connected layers, designed specifically to recognize geometric patterns in BIM object shapes. Activation functions such as ReLU were utilized to introduce non-linearity, enhancing the model's learning capability. The models also utilizes the pretrained VGG model.

3.1.2 Training Process

Training coded in ‘mvcnn.py’ involved the use of `Trainer.py` framework with cross-entropy loss and Adam optimizer, with metrics such as accuracy and loss tracked to evaluate geometric feature extraction performance. Data procesing and augmentation techniques were also applied to increase model robustness.

3.2 Graph Neural Network (GNN)

3.2.1 Architecture

Our GNN architecture, detailed in `models.py’ uses graph convolutional layers to process the graph representations, focusing on encoding the spatial and relational context between BIM objects.

3.2.2 Training Process

The GNN training coded in ‘arcgnn.py`, facilitated by the `Trainer.py` framework, utilized node features and edge relationships to enhance classification accuracy, employing a similar loss function and optimizer as the CNN component.

4. Integration of MVCNN and ArcGNN to the HybridModel

4.1 Hybrid model architecture

HybridModel architecture, detailed in `models.py’ uses the pretrained integrated models and a fusion layer and a simple (for now) linear layer to create the final classification.

4.2 Fusion Layer by concatination

A fusion layer was implemented to combine the outputs of the CNN and GNN, enabling the integrated model to leverage both visual features and contextual information for classification.

4.3 Joint Training Strategy

The training strategy involved sequential phases where the CNN and GNN components were first trained independently (or not, depends o user prefrence) on their respective data types before their outputs were fused and fine-tuned jointly to ensure effective integration.

5. Evaluation and Metrics

Performance evaluation utilized accuracy, precision, recall, and F1-score, measured against a train, validation and finally test set to assess the hybrid model's effectiveness (it is also eveluated when pretraining the separate models themselves). Hoping for a notable improvement over standalone models, highlighting the synergy between CNN (visual) and GNN (contextual) components.

6. Implementation Details

The project was implemented using PyTorch for model development and training, with experiments run on CPU hardware that makes it hard to manage computational demands. Libraries such as torchvision and PyTorch Geometric were instrumental in facilitating model construction and training processes.

7. Challenges and Solutions

We encountered challenges in data heterogeneity and model integration complexity (so complex!), and also on computational demends (CPU). These were maximly addressed through intended and focused preprocessing and by iteratively refining and checking how the models perform alone before the fusion mechanism between CNN and GNN outputs, ensuring seamless model integration.

8. Conclusions

This project intends to demonstrate the potential of hybrid CNN-GNN models in enhancing BIM object classification (or even 3D objects classification when they have a dependency between them). By capturing both geometric features and contextual relationships, our approach offers a comprehensive solution to a complex problem, potentially revolutionizing how we interact with BIM data (and more).

Results

MVCNN (1 epoch, 1000 batches):

```
/Users/avishagnevo/Desktop/archi_project/ifcnet-models-master/ifcnet-env/lib
/python3.9/site-packages/sklearn/metrics/_classification.py:1509: UndefinedM
etricWarning: Precision is ill-defined and being set to 0.0 in labels with n
o predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
{'train_loss': 0.004047002683185514, 'train_accuracy_score': 0.6633663366336
634, 'train_balanced_accuracy_score': 0.15952380952380954, 'train_precision_
score': 0.4738330975954738, 'train_recall_score': 0.6633663366336634, 'train
_f1_score': 0.5528052805280528}
```

ArcGNN (30 epoch, 1000 batches):

```
/Users/avishagnevo/Desktop/archi_project/ifcnet-models-master/ifcnet-env/lib/python3.9/site-packages/sklearn/metrics/_cla
cation.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples
  `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
{'train_loss': 0.030279628786360886, 'train_accuracy_score': 0.607, 'train_balanced_accuracy_score': 0.2331246307960532,
n_precision_score': 0.5402352535224065, 'train_recall_score': 0.607, 'train_f1_score': 0.5037732978482655}
```

Hybrid Model (1 epoch, 1000 batches):

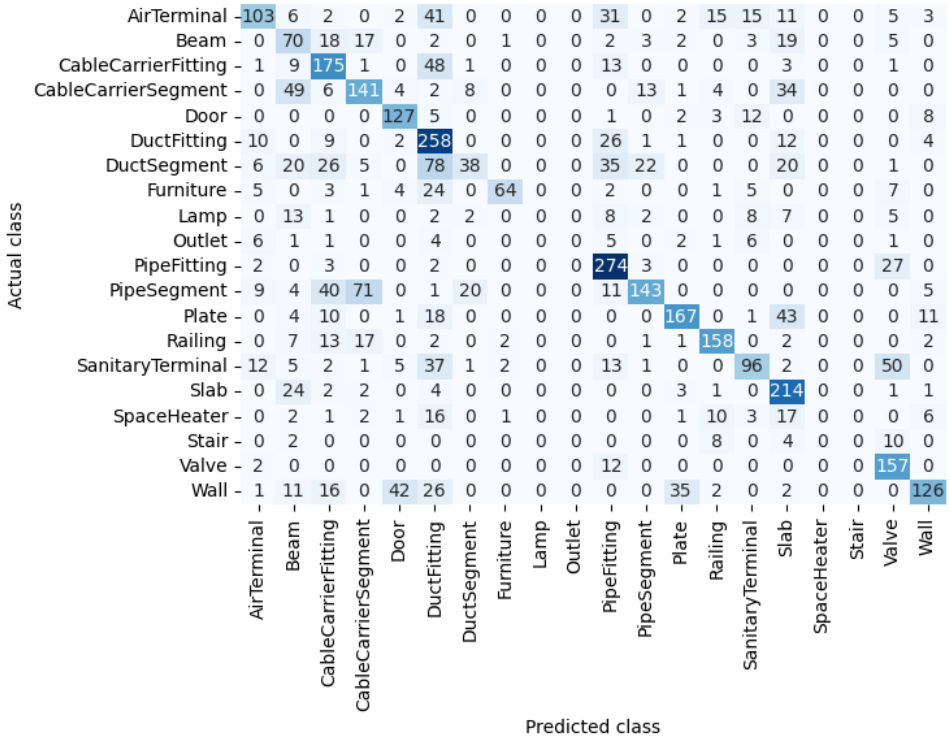
```
/Users/avishagnevo/Desktop/archi_project/ifcnet-models-master/ifcnet-env/lib/python3.9/site-packages/skle
arn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.
0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
{'train_loss': 0.035651575820722536, 'train_accuracy_score': 0.584, 'train_balanced_accuracy_score': 0.18
031906223203772, 'train_precision_score': 0.4182612088833019, 'train_recall_score': 0.584, 'train_f1_scor
e': 0.4671452154402974}
```

The results we've seen so far are essentially a proof of concept for our project. While we faced some limitations, notably in the data categories available—our IFC objects dataset wasn't as comprehensive as we would have liked, and we even lacked any BIM objects 3D data—, the Hybrid Model still managed to perform close to the individual models. This performance is particularly noteworthy considering the Hybrid Model's complexity (twice the parameters to twick) and the limited training sessions. The Hybrid Model integrates more parameters than the individual models and naturally requires more extensive training to truly excel. Given these factors, the results we achieved are not surprising but are indeed very encouraging. They confirm that the foundational design of our model is sound and that the integration of different modeling techniques is a viable approach. We are genuinely satisfied with how the Hybrid Model has leveraged the strengths of the underlying models despite the (much) shorter training time. This success gives us confidence that with the right data the Hybrid Model's performance could improve significantly.

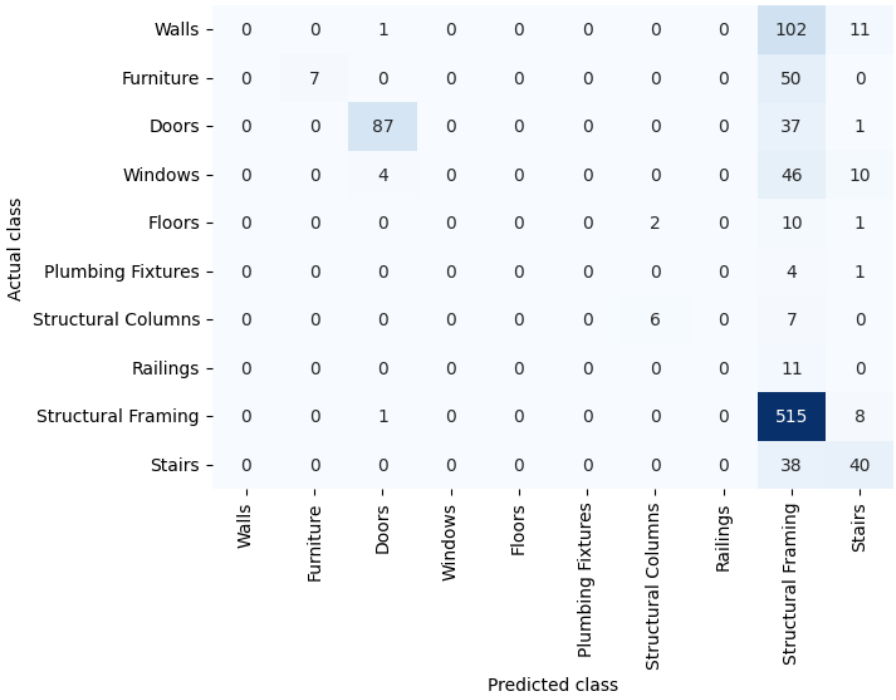
In sum, while this project is submitted in its early stages, the positive outcomes we've observed are a clear indication that we're on the right track. With this POC, we can clearly see that the potential for improved results with enhanced data and more training is promising.

Here are some confusion matrics weve created to visualize a bit better the POCs:

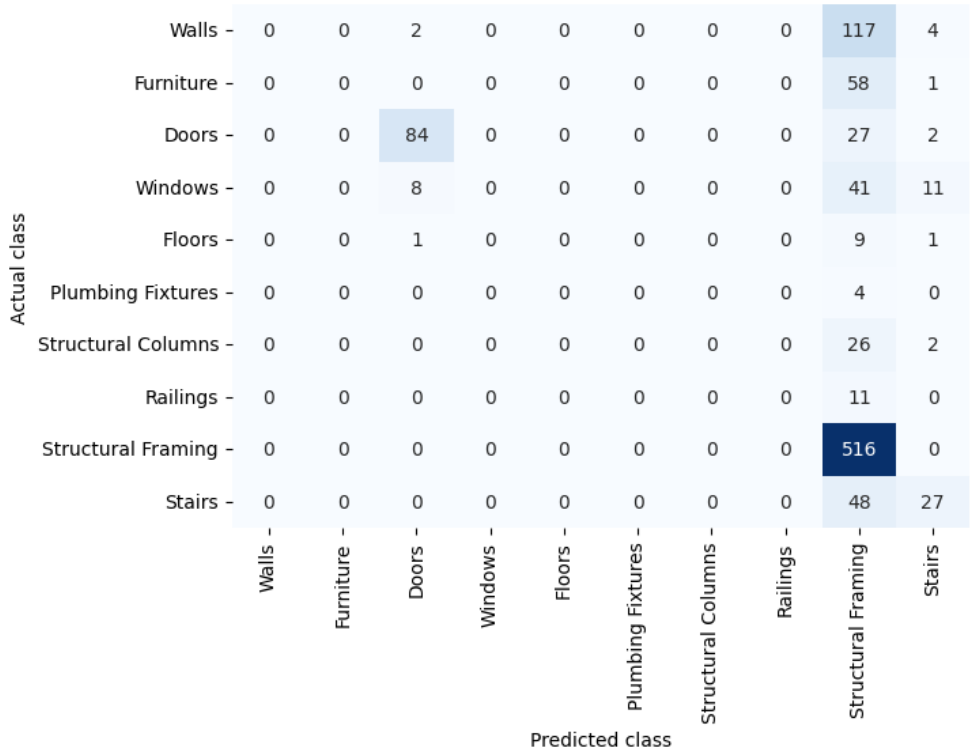
Initial POC of MVCNN (only 1 epoch trained) confusion matrix on train set including all the original categorues can be found in IFCNETCore dataset:



Initial POC of ArcGNN (only 30 epoch trained) confusion matrix on train set (for only 1000 data points) including only categories can be found in the Revit Building dataset:



Initial POC of HybridModel (only 1 epoch trained) confusion matrix (for only 1000 data points) on train set including only categories can be found in the Revit Building dataset, and sampled random data from the IFCNET dataset to create the hybrid dataset for training:



Note on Dataset Imbalance - It's important to mention that, despite our efforts to create balanced datasets, both the RevitBuildings and IFCNetCore datasets we used were somewhat imbalanced. This situation reflects the real-world architectural data, especially when focusing on specific categories like the ones we targeted in our project, with more time we would asses this imblance with statistical weightning methods like IPW or undersampling the majority classes.

Discussion:

What We Have Learned - Throughout this project, we've learned a great deal about the process of running a data science project from start to finish. It was a hands-on experience that taught us not just about coding and data analysis, but also how to keep a complex project moving forward on schedule. One of the key skills we developed was how to set up a project pipeline. This included everything from preparing data sets to training models and making sense of the results. We also learned how important it is to save our work regularly. This way, we didn't lose progress and could pick up where we left off without any setbacks. Another thing we have learned is how sticking to our project timeline was crucial. It helped us make steady progress by

breaking the work into manageable chunks (thanks to the Milestones planning requirments) and making sure we completed each part on time. This approach not only kept us focused but also highlighted the value of good planning in managing big projects.

Implications of Our Findings - Our project showed us that understanding the context in which data exists can be just as important as the data itself. By combining visual data with contextual information, we were able to improve our model's accuracy. Also, we found that converting 3D objects into multiple 2D images was a really effective technique that we want to keep using in future projects. Another practical lesson was about managing limited time and computational resources. We learned that sometimes simpler models are better not just because they're easier to handle but because you can run them more often and tweak them, which is often not possible with more complex models when you're on a tight schedule or don't have powerful computing resources.

If We Had More Time - If we weren't limited by time and our computers' capabilities, we would have liked to train our HybridModel for many more epochs to see just how effective it could really be. We would also spend more time on the other models to really nail down the advantages of using the HybridModel over using any one approach by itself. More powerful computers would have cut down our training times significantly, allowing us to experiment more and refine our models further. This would likely lead to even better results and give us a clearer understanding of which techniques work best and why.

Few words to add - This project was both a challenge and a joy. It pushed us to apply complex technical skills while also managing the project efficiently. While we sometimes got (very) frustrated by the limitations we faced (we faced many, in every step), these challenges also taught us to be resourceful and innovative with what we have. Moving forward, we're excited to take what we've learned and apply it to new projects.

Appendix:

Link to Github repository : https://github.com/avishagnevo/HybridClassification/tree/main

We would truly appreciate it if you could take the time to thoroughly explore our GitHub repository. We've poured our hearts into developing this project, working tirelessly to construct the pipeline that we believe showcases our hard work and dedication. Our GitHub repository is the core of our project, containing everything from the detailed README file to the comprehensive codebase that drives our hybrid classification model. It's not just about the final results, but the journey we took to get there, the challenges we faced, and the solutions we crafted. We believe that a deep dive into our repository will provide a clear view of the innovative approaches and methodologies we employed.

BIM elements classification Project

Intelligent Systems
[1] Avishag Nevo 324079763 Yogev Namir 318880754

April 1st 2024

The report contains pseudo-codes for all the files at the project (till this point)

1 Methodology

1.1 Data Preprocessing for MVCNN

The preprocessing pipeline for the Multi-View Convolutional Neural Network (MVCNN) begins with the extraction of 3D BIM objects from the IFC (Industry Foundation Classes) format. These objects are rendered into 2D images from various angles to capture comprehensive geometric features. The steps are as follows:

1. **IFC Extraction:** 3D models are extracted from the IFC files, which are standard data models used in the building and construction industry to describe building and construction industry data.
2. **Rendering 2D Images:** Each 3D model is rendered into 2D images from predefined viewpoints. This step is crucial for capturing the geometry of BIM objects from multiple perspectives.
3. **Dataset Creation:** The rendered images are then organized into a dataset, named *IFCNetCore*, suitable for training the CNN. Each image is labeled according to the object it represents, facilitating the supervised learning process.

1.2 Subgraph Creation for GNN

To leverage the contextual information within the building structure, we create subgraphs where nodes represent BIM objects, and edges denote the adjacency relationships. This process, designed to train the GNN, involves the following steps:

1. **Graph Construction:** A graph representation of the building is constructed. Each node in the graph represents a BIM object, while edges are formed based on the spatial and relational adjacency between these objects.
2. **Neighborhood Extraction:** For each node, a neighborhood is defined based on the adjacency relationships. This step uses the *helpers.py* functions to determine the connectivity and spatial relations among BIM objects.
3. **Subgraph Generation:** Subgraphs are generated for each BIM object by extracting it along with its neighborhood. This allows the GNN to learn from both the properties of individual objects and their context within the building structure.

1.3 Multiview Image Dataset for MVCNN

The `MultiviewImgDataset` class is designed to handle the preprocessing and organization of BIM object images for the MVCNN component. It operates by loading and transforming 2D images of BIM objects, which are rendered from various viewpoints to capture their geometric features effectively. The main steps involved in this dataset preparation include:

1. Loading images from a specified directory, which are organized according to their class names and partitioned into training, validation, or test sets.
2. Each BIM object is represented by a fixed number of views (`num_views`), and images are grouped accordingly to maintain consistency across the dataset.
3. Upon accessing an item in the dataset, the corresponding images are loaded, optionally transformed (e.g., normalization, resizing), and stacked into a single tensor, alongside their associated label derived from the directory structure.

1.4 Single Image Dataset

The `SingleImgDataset` class facilitates the handling of datasets where each BIM object is represented by a single image. This simpler dataset structure is

useful for tasks that require only a singular perspective of each object. The workflow is similar to the `MultiviewImgDataset`, but simplified to accommodate only a single image per object instance.

1.5 GNN Architectural Dataset

The `GNNArchitecturalDataset` class prepares graph-based data for the GNN component, focusing on the structural and relational aspects of BIM objects within a building. It involves:

1. Parsing JSON files containing BIM object features and their relationships, extracted from the building's digital representation.
2. Each BIM object is represented as a node in the graph, with edges denoting the adjacency or relational ties between objects.
3. Node features include geometric dimensions and other relevant attributes, while edge connections encapsulate the spatial relationships.
4. The processed graph data, including node features, edge indices, and labels, are saved for efficient loading during model training.

1.6 Hybrid Dataset Integration

The `HybridDataset` class integrates the MVCNN and GNN datasets, enabling the simultaneous training and inference of the hybrid model. This approach leverages the geometric detail captured by the MVCNN with the contextual relationships encoded by the GNN. The integration process involves:

1. For each instance in the GNN dataset, the corresponding multiview images are retrieved from the MVCNN dataset based on a predefined category mapping.
2. This mapping ensures that the GNN's structural data is complemented with the appropriate visual representations from the MVCNN dataset.
3. The combined dataset thus provides a holistic view of each BIM object, incorporating both its visual geometry and its contextual placement within the building architecture.

1.7 Convolutional Neural Network (CNN) Architecture

The Convolutional Neural Network (CNN) architecture is designed for extracting high-level features from 2D images of BIM objects. While the exact layers and configurations may vary, a typical CNN architecture for this application includes:

1. **Convolutional Layers:** Multiple convolutional layers with varying kernel sizes to capture different aspects of the input images. These layers are equipped with Rectified Linear Unit (ReLU) activation functions to introduce non-linearity.
2. **Pooling Layers:** Pooling layers follow some of the convolutional layers to reduce the spatial dimensions of the feature maps, thus reducing computational load and overfitting.
3. **Fully Connected Layers:** After several convolutional and pooling layers, the feature maps are flattened and fed into fully connected layers, culminating in a final layer that matches the number of classes in the dataset for classification.
4. **Dropout and Normalization:** Dropout layers are included to prevent overfitting, and batch normalization layers are used to stabilize training by normalizing the inputs to each layer.

This architecture allows the CNN to learn complex patterns in the geometric shapes of BIM objects from their rendered 2D images.

1.8 Graph Neural Network (GNN) Architecture

The Graph Neural Network (GNN) component is tailored to capture the contextual relationships between BIM objects within a building's structure. A standard GNN architecture for this task could involve:

1. **Graph Convolutional Layers:** These layers, such as GCNConv, process the nodes' features along with the graph's edge information to capture the dependencies between nodes (BIM objects) based on their connections.
2. **Pooling Layers:** To reduce the dimensionality and aggregate information across the graph, global pooling layers like global mean pool are utilized to summarize the features of the entire graph into a single vector.
3. **Fully Connected Layers:** The aggregated graph features are then processed through one or more fully connected layers to facilitate the final classification decision.
4. **Activation Functions:** Non-linear activation functions like ReLU are used after each layer to introduce non-linearities into the learning process.

1.9 Single View CNN (SVCNN) Architecture

The Single View Convolutional Neural Network (SVCNN) is designed for processing individual 2D images of BIM objects. This model architecture typically features:

1. **Convolutional Layers:** A sequence of convolutional layers, each followed by a ReLU activation function, to extract features from the input images.
2. **Pooling Layers:** Pooling layers interspersed among the convolutional layers to reduce the dimensionality of the feature maps, enhancing computational efficiency and feature robustness.
3. **Fully Connected Layers:** One or more fully connected layers following the convolutional layers to integrate the learned features into a form suitable for classification.
4. **Output Layer:** A final fully connected layer with an output size matching the number of classes, using a softmax activation function to provide class probabilities.

1.10 Multi-View CNN (MVCNN) Architecture

The Multi-View Convolutional Neural Network (MVCNN) extends the concept of SVCNN to handle multiple 2D images of a single BIM object from different viewpoints. Its architecture is characterized by:

1. **Shared Convolutional Layers:** A series of convolutional layers shared across all views to extract view-specific features, ensuring efficient learning of geometric patterns from multiple perspectives.
2. **View Pooling Layer:** A layer that aggregates the features from all views into a unified representation, often implemented as a max-pooling operation across the view dimension.
3. **Classification Layers:** Following the view pooling, fully connected layers culminate in an output layer for classification, similar to the SVCNN.

1.11 Architecture GNN (ArcGNN) Architecture

The Architecture GNN (ArcGNN) focuses on the relationships and spatial context between BIM objects within a structure. Its key components include:

1. **Graph Convolutional Layers:** These layers process node features (BIM object attributes) and their connections (edges) to encode both local and global structural information.
2. **Pooling and Aggregation:** To summarize information across the graph, pooling layers aggregate node features into a global graph descriptor, facilitating the consideration of overall building topology.
3. **Fully Connected Layers:** The aggregated graph features are then processed by fully connected layers, leading to a classification or regression layer depending on the task.

1.12 HybridModel Architecture

The HybridModel combines the strengths of SVCNN, MVCNN, and ArcGNN into a cohesive framework, designed to provide a comprehensive understanding of BIM objects by considering both their visual features and architectural context. The integration strategy involves:

1. **Feature Extraction:** Parallel processing pathways for SVCNN and MVCNN to extract visual features, and ArcGNN to encode structural relationships.
2. **Feature Fusion:** A fusion layer or mechanism that intelligently combines the features from the individual models, taking into account the unique contributions of each.
3. **Classification or Regression:** The fused features are then fed into a series of fully connected layers that culminate in the final decision layer, optimized for the specific tasks of BIM object classification or attribute prediction.

1.13 MVCNN (and SVCNN) Training Pipeline

The training pipeline for the Multi-View Convolutional Neural Network (MVCNN), which also applies to the Single View CNN (SVCNN) due to its inclusion as a component, consists of the following steps:

1. **Data Preparation:** Loading and preprocessing of the dataset, including normalization and augmentation techniques, to prepare the multi-view images for efficient training.
2. **Model Initialization:** The MVCNN model is instantiated with predefined configurations. For SVCNN, a subset of the MVCNN pipeline is used focusing on single images.
3. **Loss Function and Optimizer:** Selection of a suitable loss function (e.g., cross-entropy loss) and an optimizer (e.g., Adam or SGD) for adjusting model weights during training.

4. **Training Loop:** Iterating over the dataset in batches, where each iteration involves forwarding the input through the model, calculating the loss, and updating the model parameters based on the gradient of the loss.

5. **Evaluation:** Periodic evaluation of the model on a validation set to monitor performance improvements and make adjustments to hyperparameters or the training process as necessary.

6. **Checkpointing:** Saving the model's state at certain intervals or when performance milestones are reached, ensuring that training can be resumed or the model can be deployed.

1.14 ArcGNN Training Pipeline

The Architecture GNN (ArcGNN) training pipeline is tailored to process graph-structured data, with key steps as follows:

1. **Graph Data Preparation:** Loading the architectural dataset, which includes processing the building's structural information into graph format (nodes and edges) suitable for GNN processing.

2. **Model Setup:** Initialization of the ArcGNN model, configuring the graph convolutional layers and any other components specific to the architectural understanding.

3. **Optimization Criteria:** Selection of a loss function appropriate for graph data and an optimizer to facilitate effective learning.

4. **Batch Training:** Unlike conventional image data, graph data might be trained in a different manner, possibly involving entire graphs or subgraphs depending on the model's capability to process varying graph sizes.

5. **Performance Monitoring:** Evaluation of model performance using metrics suitable for the classification or prediction tasks at hand, including adjustments based on validation feedback.

6. **Model Saving:** Regularly saving the model state to allow for training interruptions and later deployment of the model.

1.15 Hybrid Model Training Pipeline

The Hybrid model combines the MVCNN and ArcGNN models to leverage both image-based and graph-based features. Its training pipeline integrates aspects from both models:

1. **Integrated Data Handling:** Simultaneous loading and preprocessing of image and graph data, ensuring compatibility and synchronization between the two types of inputs.

2. **Hybrid Model Configuration:** Initialization of the Hybrid model, which includes setting up the fusion mechanism for combining features from MVCNN and ArcGNN outputs.

3. **Unified Optimization:** Adoption of a training strategy that encompasses loss functions and optimizers capable of handling the diverse nature of inputs and learning requirements.

4. **Coordinated Training Steps:** Execution of training steps that account for both image and graph data, possibly including separate but concurrent training phases for each model component before fusion.

5. **Evaluation and Adjustment:** Comprehensive evaluation using datasets that provide both types of data, and refinement of the training process based on performance metrics.

6. **Checkpointing and Saving:** Implementation of a robust system for saving the state of the hybrid model, facilitating both iterative improvement and deployment readiness.

1.16 Trainer Class and Training Function

The `Trainer` class in `Trainer.py` plays a central role in orchestrating the training process for the models. It abstracts and encapsulates the common tasks associated with training deep learning models, such as batch processing, loss calculation, model evaluation, and checkpoint management. The methodology facilitated by the `Trainer` class includes:

1. **Initialization:** The `Trainer` class is initialized with model-specific parameters, including the model instance, dataset, optimizer, loss function, and any relevant configuration options like batch size and number of epochs.

2. **Training Loop:** The core of the `Trainer` is its ability to efficiently manage the training loop. This involves iterating over the dataset in batches, forwarding each batch through the model, computing the loss, and updating the model's weights using backpropagation.

3. **Evaluation:** Alongside training, the `Trainer` periodically evaluates the model's performance on a validation set to monitor its generalization ability. This evaluation step helps in identifying overfitting and guiding hyperparameter tuning.

4. **Checkpointing:** To ensure that training progress is not lost and the best model state can be retrieved, the `Trainer` implements checkpointing functionality. This involves saving the model's parameters at specified intervals or when a new best performance is observed.

5. **Logging and Metrics:** Throughout the training process, the `Trainer` maintains logs of key metrics such as loss and accuracy. This data is crucial for understanding the model's learning dynamics and making informed decisions about adjustments to the training regime.

6. **Utility Methods:** Additional utility methods within the `Trainer` may include functions for data augmentation, learning rate scheduling, and handling of device placement (CPU/GPU) for training and inference tasks.

1.17 Model Training Configuration in `train_model.py`

The `train_model.py` script serves as the entry point for configuring and initiating the training process for the MVCNN, ArcGNN, and HybridModel. This script outlines the steps for setting up each model, preparing the dataset, and launching the training sessions. The key components of this process include:

1. **Model Configuration:** Each model (MVCNN, ArcGNN, and HybridModel) is instantiated with predefined architecture parameters. This step may involve setting the number of layers, activation functions, and any model-specific configurations that influence its structure and behavior.

2. **Dataset Preparation:** For each model, the corresponding dataset is prepared, which includes loading the data, applying necessary transformations (e.g., normalization, augmentation), and splitting it into training, validation, and test sets.

3. **Training Environment Setup:** This involves configuring the training environment, including setting up the device (CPU/GPU) for training, initializing the optimizer, and defining the loss function to be used during the training process.

4. **Trainer Initialization:** The `Trainer` class is then instantiated with the model, dataset, optimizer, and loss function, along with any additional parameters such as the number of training epochs, batch size, and evaluation intervals.

5. **Training Execution:** With the `Trainer` configured, the training process is initiated. This includes iterating over the dataset for the specified number of epochs, performing forward and backward passes, updating model weights, and periodically evaluating the model's performance on the validation set.

6. **Evaluation and Testing:** After training, the model's performance is further assessed using the test set to gauge its generalization ability and readiness for deployment.

7. **Model Saving:** Finally, the trained model, along with its configuration and state, is saved for future use, allowing for inference on new data or further refinement.