My House, My Rules: Learning Tidying Preferences with Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract: Robots that arrange household objects should do so according to the user's preferences, which are inherently subjective and difficult to model. We present NeatNet: a novel Variational Autoencoder architecture using Graph Neural Network layers, which can extract a low-dimensional latent preference vector from a user by observing how they arrange scenes. Given any set of objects, this vector can then be used to generate an arrangement which is tailored to that user's spatial preferences, with word embeddings used for generalization to new objects. We develop a tidying simulator to gather rearrangement examples from 75 users, and demonstrate empirically that our method consistently produces neat and personalized arrangements across a variety of rearrangement scenarios.

Keywords: graph neural networks, preference learning, rearrangement tasks

1 Introduction

Rearranging objects in unstructured environments is a fundamental task in robotics. For example, a domestic robot could be required to set a dinner table, tidy a messy desk, and find a home for a newly-bought object. Given goal states stipulating where each object should go, many approaches exist for planning a sequence of actions to manipulate objects into the desired arrangement [1].

But how does a robot know what these goal states should be, for a rearrangement to be considered successful, tidy, and aesthetically pleasing? The ability of a robot to understand human preferences, especially regarding the handling of personal objects, is a key priority for users [2]. Several factors involved in spatial preferences are universal: lining up pencils on a desk in parallel is more aesthetically pleasing than scattering them randomly, and stacking plates away in a cupboard rather than on the sofa is common sense. However, many of the factors involved are inherently personal, as is their relative prioritization. For example, is the person left or right-handed? Do they want their favorite book tidied away neatly on a shelf, or placed on the table nearby for convenience? How high is their risk tolerance — do they deliberately keep fragile glasses further from the edge of the shelf, even if it makes them harder to reach? Do they order food in the fridge with the tallest objects at the back and the shortest at the front? Or do they keep the most frequently used items at the front and easily reachable? In this case, how does the robot know which items are most frequently used and ought to be placed near the front? This in itself depends on personal preference.

It is clear that spatial preferences are a complex, multi-faceted, and subjective concept. In this paper, we propose an object rearrangement strategy which models these preferences with a latent user vector representation, inferred by directly observing how a user organizes their home environment. The network is trained on arrangements made by a set of training users. For a test user, the robot will observe how they have arranged their home. This can be achieved by a robot using existing techniques for localization, object identification and pose estimation [3, 4, 5]. By passing these example scenes through the Variational Autoencoder, the network will extract this test user's preference vector. This can then be used by a robot to tidy up these scenes, and also predict personalized arrangements for new objects and new scenes based on how training users with "similar" preferences to this test user arranged them. These predicted arrangements can then be used as target states for planning and manipulation algorithms [1].

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

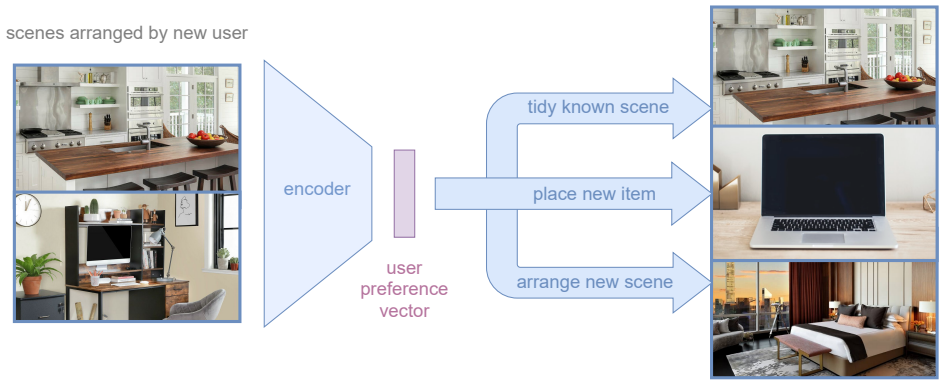

Figure 1: NeatNet infers a user preference vector, and can then predict personalized arrangements.

We make the following contributions, adding a layer of personalisation to robotic rearrangement:

Novel Graph VAE Architecture. We present NeatNet: an architecture for learning spatial preferences. Users are encoded through representation learning with Variational Autoencoders. Graph Neural Network (GNN) layers are used to learn from scenes arranged by users.

Word Embeddings for Object Generalization. Our network extracts semantic features for an object from the word embedding of its name, enabling generalization to objects unseen during training.

User-Driven Evaluation. We develop TidySim: a 3D simulator web app for users to arrange scenes. With rearrangement data from 75 users, we demonstrate empirically that our method can learn spatial preferences, and predict personalized arrangements which users rate highly. Visualizations of predicted arrangements are available in our supplementary material (along with our code) and on our anonymous project page: sites.google.com/view/tidying-preferences.

2 Related Work

Grouping semantically related objects. Collaborative filtering [6] can be used to group objects by shelf. It predicts how a user would categorize a given object based on how similar users categorized it. Data is mined from online object taxonomies such as shopping websites to generalize to objects not seen during training: preferences about known objects nearby in the hierarchy are used to predict how to tidy a new object. Object taxonomies can also be used to predict the correct cupboard in which to place an object, viewed as a classification task [7]. While these approaches are effective in grouping objects into boxes or cupboards, our method addresses the general case of learning continuous spatial relations, like how to set a dining table or tidy an office desk.

Dirichlet Process models. Recent work [8] learns spatial concepts for tidying up: that is, the relationship between an object, its tidied location, and a name to describe that location, so that voice commands can be used. Gibbs Sampling is used to learn model parameters. Another approach with Dirichlet Process modeling uses context from sampled human poses [9] to place objects in convenient locations. Density function parameters are shared across objects of the same type. This approach is evaluated on a training dataset of arrangements made by 3-5 users, with a further 2 users scoring arrangements. While these methods predict generally tidy arrangements, it is not shown whether they can tailor them to individual preferences. We develop a tidying simulator to gather data on a larger scale, demonstrating that our neural encoder makes personalized predictions on a dataset of 75 users. Furthermore, our method uses word embeddings to predict positions for objects unseen during training, and learns transferable representations of tidying preferences which can be applied to arrange new scenes.

Example-driven tidiness loss function. Another approach is to ask each user to provide several tidy arrangements for a scene, to serve as positive examples [10]. To tidy a scene, the robot computes a target tidy layout using a cost function which encourages altering the object-object distances in the untidy scene to match those in the "closest" tidy positive example. Simulated re-annealing is used to optimize this cost function efficiently. We add value firstly because our method generalizes to new objects without requiring users to arrange them manually, and secondly because our method applies

prior knowledge from other users, meaning that it works well even when the positions of objects in an individual positive example are noisy, as in real-world use cases.

Neural recommender systems. Neural networks have been used for making recommendations based on user preferences [11]. The YouTube neural recommender system [12] predicts a user's expected watch time for a future video based on past videos watched and searches made. However, these methods do not address the challenges of learning spatial relations by observing scenes and predicting tidy arrangements for any set of objects. We apply GNN components to solve this.

3 Method

Here, we introduce NeatNet: a novel architecture for learning spatial preferences. First, we describe a Variational Autoencoder for learning latent representations of a user's preferences (Section 3.1). Then, we introduce the GNN layers, which allow NeatNet to learn from scenes which have been tidied by users, modeled as graphs of objects (Section 3.2). Next, we adapt the architecture to enable learning from multiple example scenes arranged by one user (Section 3.3). Finally, we provide details on how object semantic embeddings are generated from word embeddings, allowing generalization to unseen objects (Section 3.4).

3.1 Encoding User Preferences with VAEs

Our objective is to learn a user encoder function $\mathcal{E}_\phi$, represented by a neural network with learned parameters ϕ. Its output is a user's preference vector u, and its input is a representation of a scene arrangement made by that user. Each scene is defined by the terms s (the semantic identities of the objects in the scene, further explained in Section 3.4), and p (the position encodings for the objects in the scene, e.g. coordinates). These are not passed into the VAE directly: scenes are first encoded using GNN layers (Section 3.2), so that the network can operate on scenes with a variable number of objects. We also train a position predictor function $\mathcal{D}_\theta$. Given a set of objects identified by s, and a user preference vector u, this predicts a position for each object which is consistent with those spatial preferences.

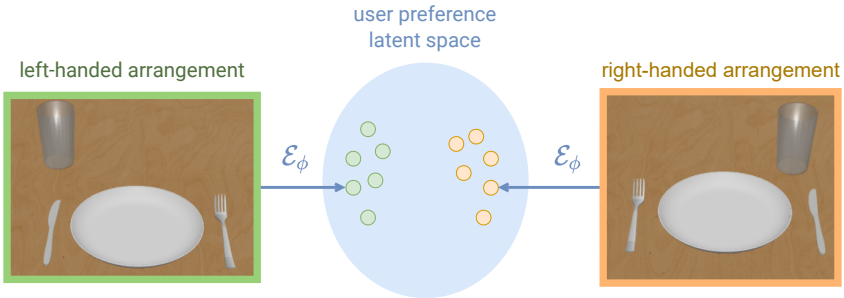

Figure 2: Two users being mapped to points in the user latent space based on how they arrange a dining scene. Users close together in this latent vector space share similar spatial preferences.

These networks are trained end-to-end as a Variational Autoencoder [13]. A scene represented by s and p is passed into the encoder $\mathcal{E}_\phi$. The output user preference vector u, along with the semantic embeddings of the objects in the scene s, is passed into the position predictor $\mathcal{D}_\theta$, which acts as a decoder by reconstructing the original scene. Since the user preference vector is a low-dimensional bottleneck, this encourages the network to learn important spatial preferences like left/right-handedness which are useful for predicting the positions of many objects at once. For variational inference, the encoder $\mathcal{E}_\phi$ predicts a Gaussian distribution over the possible user vectors u which could explain the current input scene. At training time, we sample from this distribution to obtain u, and we take the mean at inference time. The VAE objective is thus $\phi^*, \theta^* \triangleq \text{argmin}_{\phi,\theta} L(\phi, \theta)$.

$$L(\phi, \theta) \triangleq \mathbb{E}_{p_{data}(s,p)} \left[\|\mathcal{D}_\theta(\underbrace{\mathcal{E}_\phi(s,p)}_{=u}, s) - p\|_2^2 + \beta KL\left[q_\phi(u|s,p) \| p(u)\right] \right] \tag{1}$$

Here, p_{data} represents the training data distribution. $p(u)$ is a prior over the user preference vectors: a standard Gaussian, with zero mean and unit variance. KL represents the Kullback–Leibler divergence (standard in VAEs [13]) which encourages the estimated u distributions to stay close to the $p(u)$ Gaussian prior over the user space, and the hyperparameter β scales this prior term to make the latent space less sparse and encourage disentanglement [14].

3.2 Encoding Scenes with GNN Layers

We will now encode scenes as vectors so that they can be passed through the VAE. To do this, we use Graph Neural Network (GNN) layers. GNNs can operate on scenes with a variable number of arbitrary nodes, which is critical since the robot cannot foresee all the objects it will ever encounter. Each node represents an object in the scene. Node i has feature vector x_i formed by concatenating the semantic embedding for that object s_i with its position encoding p_i. The scene graph is fully connected, to avoid making assumptions about which object-object relations are relevant or not.

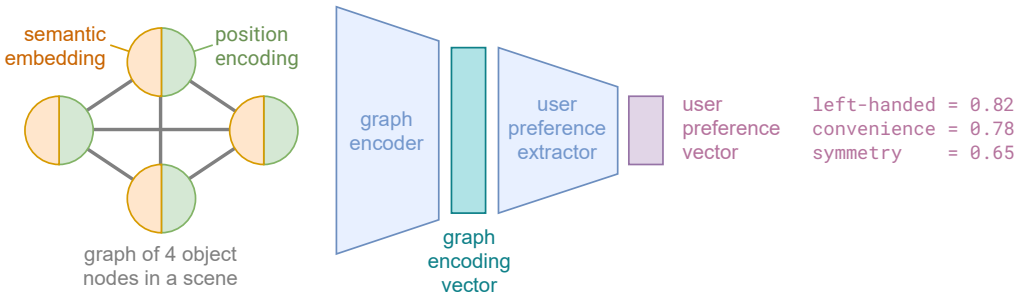

Figure 3: The encoder network, all together representing the function $\mathcal{E}_\phi$ defined in Section 3.1. The input is a node feature matrix with a row for each node's feature vector $x_i \triangleq s_i \,||\, p_i$. The graph encoder consists of GAT layers (described below) separated by ELU non-linearities [15]. Afterwards, we pass the final hidden feature vector of each node through a network of fully-connected layers, which has the same output size as the graph encoding vector. To aggregate the output node vectors into a single graph encoding vector, we apply a global add-pooling layer. This graph encoding vector is then passed through a user preference extractor, which is a series of fully-connected layers separated by LeakyReLU activations [16]. The output is a mean and variance for the user preference vector distribution, from which we sample to obtain u. We illustrate some examples of user preferences that might be captured in u (e.g. left-handedness), though of course this is a latent vector learned with an unsupervised method.

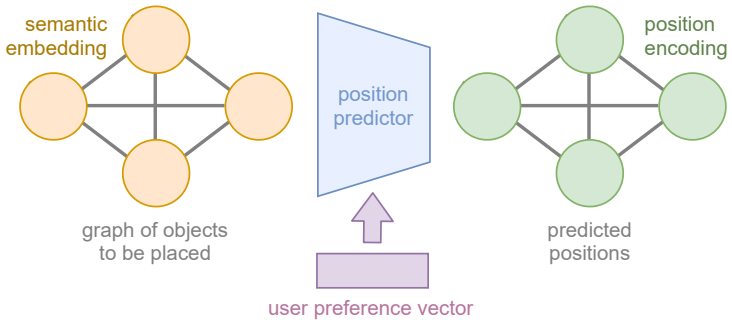

Figure 4: The position predictor, which acts as the decoder $\mathcal{D}_\phi$. The input now consists of a set of objects for which we wish to predict positions, according to a user preference vector u. We set the node feature vectors to be $x_i \triangleq p_i \| u$. This time, the output is a position vector for each node. Our experiments focus on positions, but the position vector can be extended to include orientation information. Similarly to the graph encoder, the position predictor consists of a sequence of GAT layers, after which each node is passed through a series of fully-connected layers, the final output of which is the predicted position vector for that node.

Both the graph encoder (Figure 3) and position predictor (Figure 4) use GNN layers. As is standard for GNN layers, we need to compute the hidden feature vector h_i for each node i, based on its current feature vector x_i as well as the node feature vectors x_j of its neighbours. To compute h_i, we choose the Graph Attention (GAT) layer [17]. h_i is computed as a weighted sum of the node feature vectors x_j in the current node's neighbourhood, where the weights α_{ij} are generated by the soft attention mechanism. Since our graph is fully connected, we calculate attention weights for all the other objects in the scene: therefore the attention mechanism allows the network to choose for itself which object-to-object relations are the most important for predicting correct arrangements. In order to further increase the power of the graph network, we can add a second GAT layer which takes as input the h_i features produced by the first layer. The computation of h_i by the GAT layers proceeds as follows, where $\|$ represents concatenation, and the learned parameters are the weights $\mathbf{W}$ and a:

$$h_i \triangleq \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} x_j \qquad \alpha_{ij} \triangleq \frac{\exp\left(\text{LeakyReLU}\left(a^T \left[\mathbf{W}x_i \| \mathbf{W}x_j\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(a^T \left[\mathbf{W}x_i \| \mathbf{W}x_k\right]\right)\right)} \tag{2}$$

3.3 Generalizing Across Scenes

Many spatial preferences, such as left/right-handedness, will be exhibited by the user consistently across scenes. We now extend NeatNet to learn from multiple scenes arranged by the same user, as shown in Figure 5. Every example scene arranged by the current user is passed through the single-scene encoder network $\mathcal{E}_\phi$ in Figure 3. Each scene may produce a subtly different estimate of the user's preferences, so they are aggregated through a global pooling layer to produce an average user preference vector u. On the decoding side, we use the same u vector as the input to the position predictor to reconstruct *all* the scenes created by this user. Since u is the architecture's bottleneck, this encourages the network to learn important preference features which generalize across scenes, such as left/right handedness.

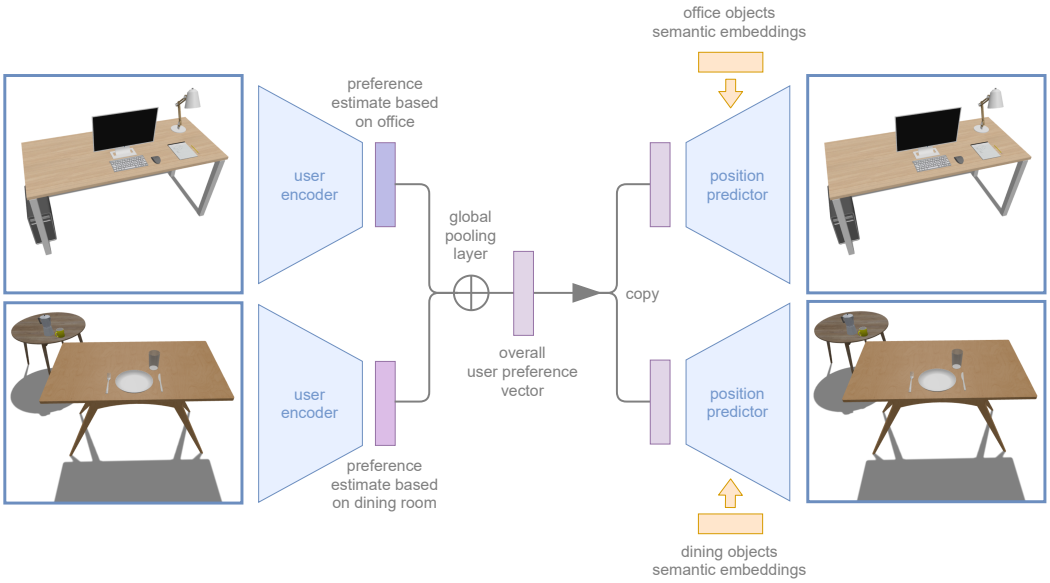

Figure 5: Learning one generalized user preference vector across multiple example scenes per user.

3.4 Semantic Embeddings

Each object's identity is represented by its semantic embedding vector. A straightforward approach is to use one-hot encoding. A limitation of this method is that it does not generalize readily to new objects. An alternative is feature vectors: example entries include the height of an object, or RGB values describing its color palette. However, this may not capture all features necessary for tidying.

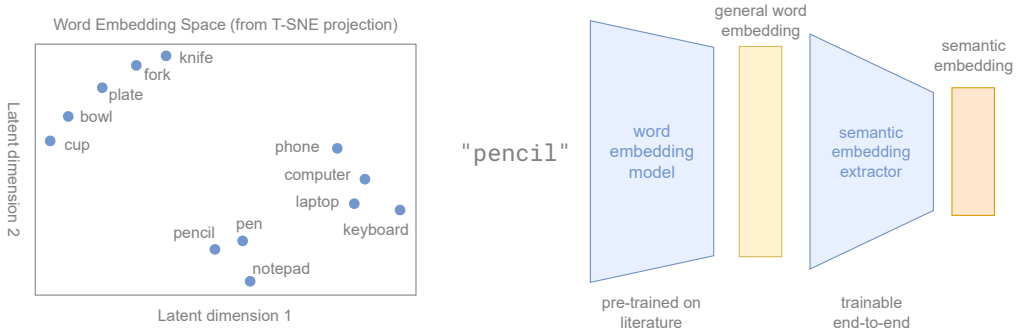

Figure 6: **Left:** Word embedding space. Object names which appear in similar linguistic contexts will be close together in this latent space. **Right:** the semantic embedding extractor outputs an object's semantic embedding vector, using fully-connected layers. This is trained end-to-end with NeatNet, to extract the object features which are most relevant for our spatial arrangement task.

We also develop a method using word embeddings. Objects which appear in similar linguistic contexts are used by people for similar activities, and so are often arranged in a related way: for example "pen" and "pencil", or "salt" and "pepper". We load word embeddings from a FastText model [18], pre-trained on the Common Crawl dataset of over 600 billion tokens. Names of objects are converted to word embeddings and passed through the semantic embedding extractor (Figure 6), yielding the semantic embeddings referred to in Figure 3. This approach allows our method to predict positions for new objects which the robot has never seen arranged during training.

4 Experiments

4.1 Collecting Rearrangement Data from Users

Gathering data from human users at scale is a significant challenge for robotics experiments. We developed TidySim: a 3D object rearrangement simulator deployed as a web app, allowing users to arrange a series of scenes through an intuitive interface. Features include: moving the camera, clicking and dragging objects to move or rotate them, and an inventory for unplaced objects.

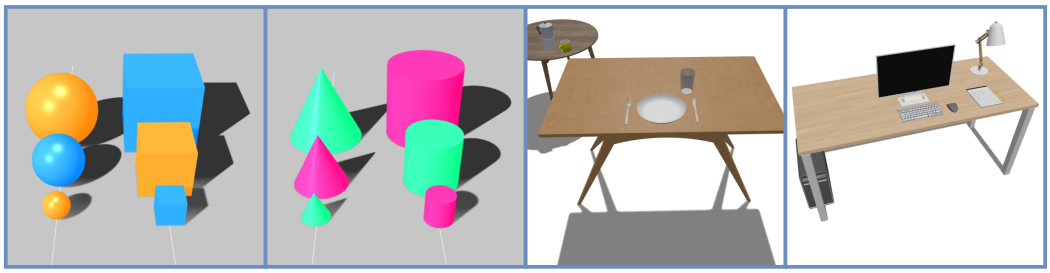

Figure 7: Scenes arranged by users. From left: (1-2) are abstract scenes, where users must split the objects into two lines, either by shape or by color, (3) is a dining room, and (4) is an office desk.

4.2 Experimental Procedure

We designed the scenes in Figure 7 to maximise experimental value while keeping the user workload low ($\sim$ 5 minutes total per user). Rearrangement data from 75 users was gathered by distributing the experiment link on social media. Each user submits a tidy arrangement for a number of scenes. We set aside a group of 8 test users, male and female, and majority non-roboticists. The training dataset was formed from the 67 remaining users. NeatNet was trained by passing training users through the Graph VAE and updating parameters with a neural network optimizer according to the loss function in Equation 3.1. For each test user, we passed their example scenes through NeatNet to extract their spatial preferences, and predict a personalized arrangement for the test scene (varies by experiment).

Arrangements were also produced by the baseline methods in Section 4.3. The test user was then asked to score the arrangements produced by each method based on how tidy they were according to their preferences, on a scale from 0 (completely untidy) to 10 (perfectly tidy).

4.3 Baselines

In Section 2, we qualitatively compare with existing methods to show where we add new capabilities. For a fair quantitative evaluation of our method on this dataset, we test against a variety of baseline methods depending on the specific rearrangement experiment. `Positive-Example`: to tidy a known scene, this baseline copies the example arrangement which was supplied by the test user (which the `NeatNet` method uses to extract this user's tidying preferences). This baseline ignores priors from other users and is unavailable when generalizing to new objects or scenes. `Random-Position` sets the missing object's predicted position randomly, while `Mean-Position` uses the object's mean position across all the example arrangements from training users. `NeatNet-No-Prefs` is an ablation baseline: the test user's preference vector is replaced with the zero vector (representing a neutral preference, which test users might not find tidy). The `Pose-Graph` method produces an arrangement which some users will find tidy, but is still not personalized to the current user's preferences. It learns a probabilistic pose graph of relative object positions from the training example arrangements, and solves it using Monte Carlo sampling and scoring to find the most likely arrangement. When placing an unseen object, `Mean-With-Offset` uses the mean position of the other objects in the scene, and adds a small random offset to minimise collisions with other objects. `Nearest-Neighbor` finds the object already placed by the user which has the closest word embedding vector to the unseen object, and places the new object next to it, i.e. it would place the salt next to the pepper. For the task of arranging a new scene, the `Random-User` baseline copies how a random training user arranged that scene, without personalisation.

4.4 Experiment Results

We designed a series of experiments to test our method's capabilities. Visualizations of predicted arrangements are available in our supplementary material and on our anonymous project page: sites.google.com/view/tidying-preferences.

Can NeatNet Tidy a Known Scene? (Table 1) Suppose that the robot has seen the user tidy a room. After the room becomes untidy, the robot must tidy up each object. This tests how well the network can reconstruct the scene based on the user preference vector and its prior knowledge. For each scene separately, we trained NeatNet as described in Section 4.2. Then for each test user we pass their scene through the VAE and ask them to rate the reconstructed scene.

Tidying Method	Abstract 1	Abstract 2	Dining Table	Office Desk	Average
`Positive-Example`	8.75	7.50	8.60	9.18	8.51
`Mean-Position`	3.25	3.00	2.34	2.22	2.70
`NeatNet-No-Prefs`	4.50	3.75	2.58	3.70	3.63
`Pose-Graph`	—	—	6.98	7.82	7.40
`NeatNet`	**8.75**	**8.25**	**9.12**	**8.90**	**8.80**

Table 1: Tidying a known scene. Each row is a tidying method, and each column a scene. Each number represents the mean tidiness score given by test users to that method for that scene.

`NeatNet` can reconstruct personalized tidy arrangements from a user preference vector, outperforming baselines and approaching the user's handmade positive examples. Surprisingly, users sometimes prefer the reconstructed tidy scene *over their own example*. This is because in each individual user-created arrangement, there is noise in how they placed each object (realistically this would also include error from localization/pose estimation). Our method uses prior knowledge from other users to correct for noise in individual examples (analogous to VAE image denoising [19]).

Can NeatNet Find a Home for a New Object? (Table 2) If a user brings a new object (e.g. a cup) into the scene, the robot should predict a tidy position for it. NeatNet is trained as before, but with the positions of random nodes masked out from input examples to encourage generalization. At test time, we mask out the new object's position from the test user's example arrangement, but

the position predictor must still place it. This is analogous to the use of generative models for "inpainting" missing or obstructed regions of input images [20]. This experiment is repeated for each object below. `NeatNet` outperforms the baselines by successfully combining what it knows about the current user's preferences with prior knowledge of how similar users placed the new object.

Prediction Method	Cup	Fork	Knife	Plate	Average
`Random-Position`	26.70	15.76	15.96	7.10	16.38
`Mean-Position`	11.42	15.06	13.48	**1.64**	10.40
`NeatNet-No-Prefs`	13.20	14.98	13.38	2.88	11.12
`NeatNet`	**5.72**	**1.18**	**1.44**	1.96	**2.58**

Table 2: Placing a new object into the scene. Figures show the absolute error between the predicted and true, unmasked positions of each object (in centimeters).

Can NeatNet Generalize to Objects Unseen During Training? (Table 3) The network must place a new blue box into an abstract scene, and a laptop into the office scene, though it has never seen a laptop during training. The training process is similar to the previous experiment. `NeatNet`'s placement of the new objects (box and laptop) is satisfactory to users. To place the laptop, `NeatNet` considers how the user placed several semantically similar objects: the computer, laptop and keyboard. By contrast, the `Nearest-Neighbor` baseline places the laptop based solely on how the user placed their desktop computer, thus misplacing it under the desk.

Tidying Method	Abstract Scene	Office Desk	Average
`Nearest-Neighbor`	4.82	1.32	3.07
`Mean-With-Offset`	2.66	5.86	4.26
`NeatNet`	**8.44**	**9.14**	**8.79**

Table 3: Placing an object unseen during training, with user scores for tidiness shown.

Can NeatNet Predict a Personalized Arrangement for a New Scene? (Table 4) At training time, the network is shown how the user has arranged both abstract scenes, with random scene masking applied to encourage generalization. At test time, the network is shown one abstract scene and asked to predict how the test user would arrange the other. We found that the `Random-User` baseline performs moderately well, especially when by luck the random training user which was chosen has similar preferences to the test user. However, when those preferences differ, the neural network method out-performs it. This shows the importance of accounting for preferences, and demonstrates that the network can learn preferences which enable generalization across scenes.

Tidying Method	Abstract 1	Abstract 2	Average
`Mean-Position`	2.48	2.36	2.42
`Random-User`	6.18	5.28	5.73
`NeatNet`	**9.16**	**9.52**	**9.34**

Table 4: User scores for predicted arrangements of a scene never seen before by the user.

5 Conclusions

Findings. Our NeatNet architecture learned latent representations of user spatial preferences by training as a Variational Autoencoder. Graph Neural Network components allowed it to learn directly from user-arranged scenes, represented as graphs, and word embeddings enabled predictions for unseen objects. NeatNet's ability to generalize and make personalized predictions was demonstrated through experiments with real users, adding new capabilities to existing methods. **Future work.** Rather than using the word embedding of the object's name, convolutional layers can be used to generate a semantic embedding for an object based on its visual appearance (e.g. teacups look similar to mugs and are placed similarly). Furthermore, we can combine our approach with model-based priors from existing methods, e.g. using human pose context [9] as part of our loss function to ensure objects are placed conveniently, while also tailoring to individual preferences.

References

[1] D. Batra, A. X. Chang, S. Chernova, A. J. Davison, J. Deng, V. Koltun, S. Levine, J. Malik, I. Mordatch, R. Mottaghi, M. Savva, and H. Su. Rearrangement: A challenge for embodied ai, 2020.

[2] C. Pantofaru, L. Takayama, T. Foote, and B. Soto. Exploring the role of robots in home organization. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, page 327–334, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450310635. doi:10.1145/2157689.2157805.

[3] J. McCormac, R. Clark, M. Bloesch, A. J. Davison, and S. Leutenegger. Fusion++: Volumetric object-level slam, 2018.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2016.

[5] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese. Densefusion: 6d object pose estimation by iterative dense fusion, 2019.

[6] N. Abdo, C. Stachniss, L. Spinello, and W. Burgard. Robot, organize my shelves! tidying up objects by predicting user preferences. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1564, 2015. doi:10.1109/ICRA.2015.7139396.

[7] M. J. Schuster, D. Jain, M. Tenorth, and M. Beetz. Learning organizational principles in human environments. In *2012 IEEE International Conference on Robotics and Automation*, pages 3867–3874, 2012. doi:10.1109/ICRA.2012.6224553.

[8] A. Taniguchi, S. Isobe, L. E. Hafi, Y. Hagiwara, and T. Taniguchi. Autonomous planning based on spatial concepts to tidy up home environments with service robots, 2020.

[9] Y. Jiang, M. Lim, and A. Saxena. Learning object arrangements in 3d scenes using human context. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, 2, 06 2012.

[10] M. Kang, Y. Kwon, and S.-E. Yoon. Automated task planning using object arrangement optimization. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 334–341, 2018. doi:10.1109/URAI.2018.8442210.

[11] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering, 2017.

[12] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[13] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2014.

[14] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[15] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[16] A. L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information, 2017.

[19] D. J. Im, S. Ahn, R. Memisevic, and Y. Bengio. Denoising criterion for variational auto-encoding framework, 2016.

[20] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with deep generative models, 2017.