Semantic Furniture Feature Extraction

By decorAID

Lim Sheng Xiang 1005005
Bryce Goh Ying Ming 1005016
Matthew Pohadi 1005379
Mah Yi Da 1005024
Ayu Permata Halim Mendoza 1006069

Table of Contents

1. Introduction

People often use mood boards to plan their home renovations, collecting images that represent their ideal living space. However, finding decor and furnishings to match these mood boards can be difficult, as the ideal items may not exist or be available.

This is where reverse image search technology comes in. Our project investigates how this technology can help homeowners find decor that matches their vision by searching for similar items on the market. We aim to improve how reverse image search meets users' needs, making it easier to turn their dream homes into reality.

2. Problem Scoping and Definition

To effectively address the challenge of bridging the gap between the conceptual and the achievable realms in home decor, the project unfolds into two phases:

- Phase 1: Detecting and extracting furniture in an image.
- Phase 2: Searching for similar furniture items based on extracted furniture

This process is illustrated in Figure 1, where the initial phase detects and extracts the sofa, while the subsequent phase recommends sofas that are available in the market.

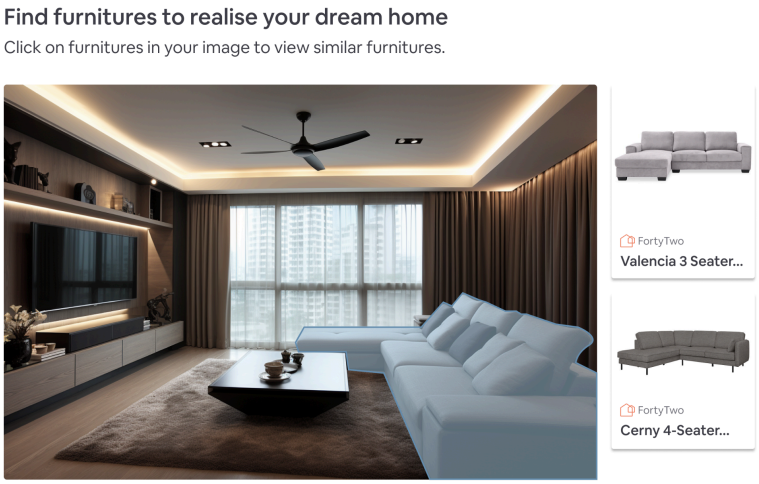


Figure 1. User interface of our solution.

The detection and extraction of furniture in phase 1 can be achieved through semantic segmentation. Since there are existing open-sourced models trained on the ADE20k dataset that are sufficient for segmenting furniture, the bulk of the project focuses on the recommendation of similar furniture. This phase can be broken down into the following steps: web scraping, data preprocessing, model research and finally model evaluation.

3. Dataset and Collection Methodology

3.1. Methodology

Fortytwo.sg, a prominent online furniture shopping platform in Singapore, was selected for the dataset collection due to its extensive and diverse array of furniture offerings. The process involves web scraping of the site, collecting data across several key furniture categories: sofas, beds, tables, chairs, and storage units.

Data collection was facilitated through the utilisation of Puppeteer, a Node.js library that provides high-level APIs for headless chrome control. To expedite the data collection process, worker threads in Node.js were leveraged to execute JavaScript in parallel.

The data collection process followed these steps:

1. Each worker thread is responsible for collecting data for a category.
2. With reference to a separate CSV that stores all URLs of products for that category, each worker thread systematically visits all listings.
3. On each listing's webpage, relevant information was extracted and stored in a CSV file containing the furniture details.

For each product listing, a comprehensive set of features essential for subsequent analysis was extracted, including Images, Category, Subcategory, Product id, Product url, Image Urls, Price, Colours, Description and Specifications.

3.2. Statistics of dataset

After we have collected the data for all products available, we analysed the distribution of our data. Figure 2 below shows the distribution of subcategories for all categories in the data we collected.

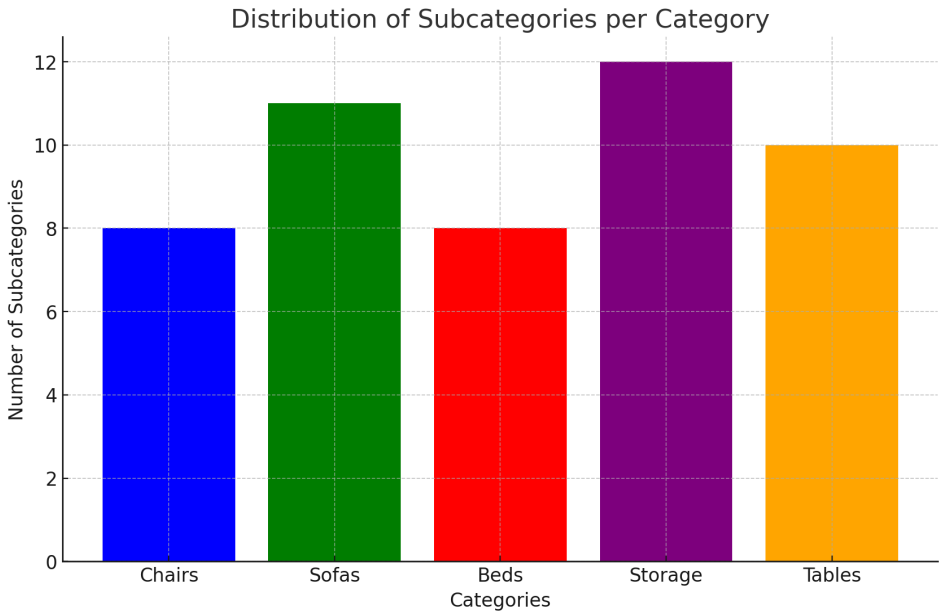


Figure 2. Distribution of subcategories per category.

As seen above, we have managed to capture a relatively diverse set of products, with each category containing an average of 9 to 10 subcategories. Next, we analysed the distribution of images for our data.

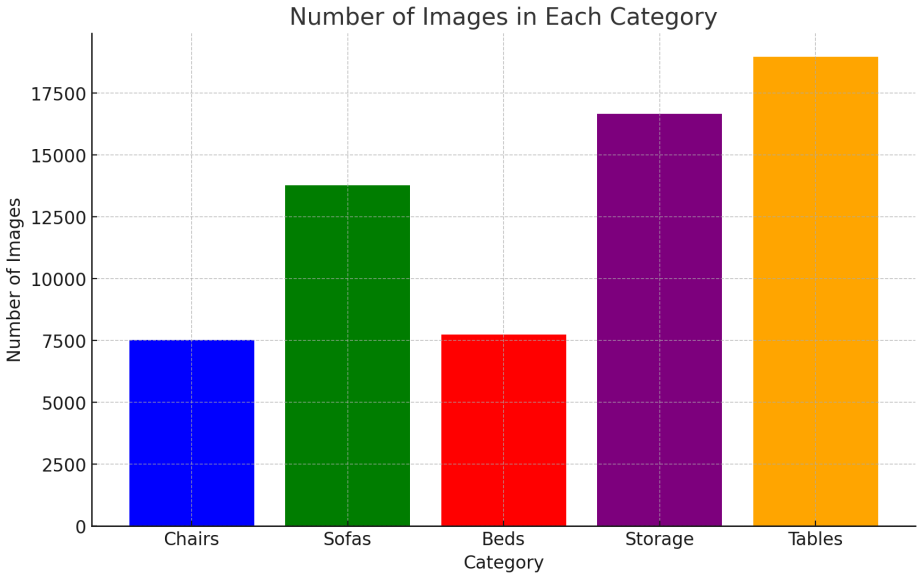


Figure 3. Distribution of images collected.

From Figure 3 above, it can be seen that each category does contain a huge amount of images, amounting to thousands. Thus, given the extensive number of images and time constraint imposed by the project, a decision was made to focus on two categories. In order to introduce a degree of complexity to the model, the products in these two should share some similarity in physical attributes. Thus, the chairs and sofas category were chosen.

3.3. Images Collected

For the images collected, they are categorised into two main categories as illustrated in Figure 4; Product Images and Lifestyle Images.

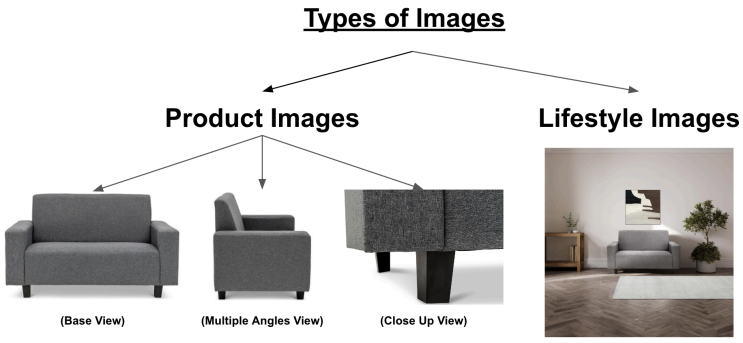


Figure 4. Different types of images.

Product Images, characterised by a white background, provide a clear view of the furniture from various perspectives including a base view, multiple angles, and a close up view. Lifestyle Images, in contrast, showcases the furniture within an actual living space, offering insights into styling and context.

3.4. Ideal Dataset

The ideal dataset for this project would have the following criteria:

1. Product images with a clean white background. This decreases the noise being fed into the model during training.
2. Product images that do not contain any text. This is to prevent our model from learning incorrect features. For example, texts in an image do not belong to a furniture even though both the furniture and the text are in the same image.
3. All images in the dataset should have the same aspect ratio as our model expects a fixed input size.

3.5. Problems with Dataset

While the dataset from fortytwo.sg is comprehensive and diverse, it is not without its challenges. Key issues identified include the mislabeling of lifestyle images as product images, the presence of extraneous text within some images, and inconsistencies in aspect ratios across the different image types.

Figure 5. Lifestyle Image mislabelled as a product Image

Figure 6. Product images containing text

For instance, Figure 5 illustrates a scenario where an image, clearly depicting furniture within a living space, is erroneously categorised as a product image post-web scraping. Similarly, Figure 6 highlights a product image that includes text overlays indicating furniture dimensions, which could potentially distract the model during the training process by focusing on irrelevant details.

Such discrepancies can significantly hinder the model's ability to learn accurately, as mislabeling introduces incorrect data points, and text in images can lead to inaccurate learning of our product features. Furthermore, inconsistent aspect ratios may complicate image processing and feature extraction, necessitating additional steps to standardise image inputs.

To ensure the integrity of our model training, a meticulous preprocessing phase was undertaken. This includes employing specific strategies to correct mislabelings, remove extraneous text, and standardise aspect ratios across all images. Each step in our preprocessing strategy, detailed in the forthcoming ‘Dataset Preprocessing’ section, will refine our dataset, void of said challenges, making it ideal for training.

4. Dataset Preprocessing

To make the ideal dataset for training purposes, challenges identified in the earlier sections needed to be addressed. To clean the mislabeled data, the 'Percent Coverage' method followed by the 'White to Non-White Pixel Ratio' method was applied. Following the cleaning process, images containing texts were removed and the aspect ratio of all the data was standardised.

4.1. Percent Coverage

Upon analysis of the data, it was observed that all product images have white backgrounds with the respective furniture centred within the frame. In contrast, lifestyle images lack white backgrounds, and may exhibit off-centred furniture placement. This observation led to the differentiation of the images via percent coverage, first done by creating a bounding box to cover non-white pixelated areas. If the image identified is indeed a product image, then the bounding box will be created from the centre as illustrated in Figure 7 below.

Figure 7. Bounding box on product image

After identifying the bounding box area, percent coverage is calculated using the following formula.

$$Percent\ Coverage = \frac{Bounding\ Box\ Area}{Image\ Area}$$

The hypothesis posited is that percent coverage can effectively differentiate product images from lifestyle images. If this hypothesis proves valid, an image with percent

coverage approaching 1 would signify a lifestyle image, since the bounding box area for lifestyle images will align closely or match with the image area of one without a white background. Conversely, if an image has a percent coverage lesser than 1, then it will be identified as a product image.

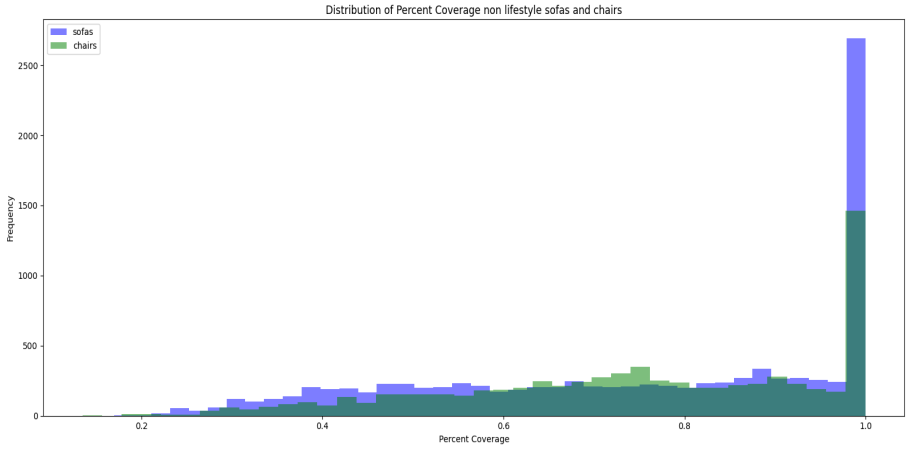


Figure 8. Distribution of percent coverage over entire dataset

The distribution shown in Figure 8 supported the hypothesis, as identified ‘lifestyle’ images are concentrated near the percent coverage value of 1.0, while the ‘product’ images are scattered with percent coverage values of less than 1.0.

Figure 9. Inaccurate representation of lifestyle images.

However, upon further analysis, the limitations of this method became apparent, as evident from Figure 9. Some of the images classified as lifestyle images using the percent coverage method are actually product images, as illustrated by the red boxes in Figure 9. Therefore, an alternative method was necessary to achieve a better differentiation.

4.2. White to Non-White Pixel Ratio

Given the limitations of the percent coverage method, a different method was needed to differentiate mislabelled and accurately labelled product images. It was observed that accurately labelled product images have a higher proportion of white pixels compared to mislabeled product images. Hence, the ratio of white to non-white pixels was employed as a subsequent differentiating measure.

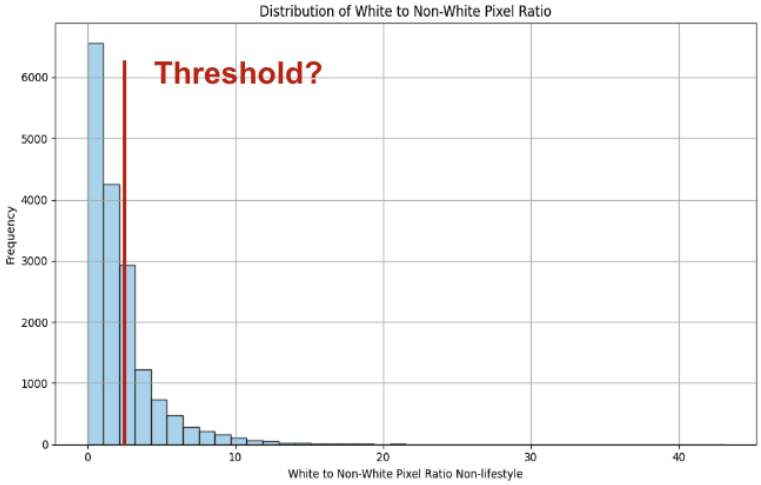


Figure 10. Ratio distribution of product images (Accurate and Mislabelled)

Figure 10 shows the distribution of the 'White to Non-white Pixel Ratio' for product images provided by fortytwo.sg. A 'White to Non-white Pixel Ratio' close to zero indicates that the product image is likely to be mislabeled. Beyond a certain threshold, images are correctly labelled as accurate product images. The task was then to determine an appropriate threshold for this differentiation.

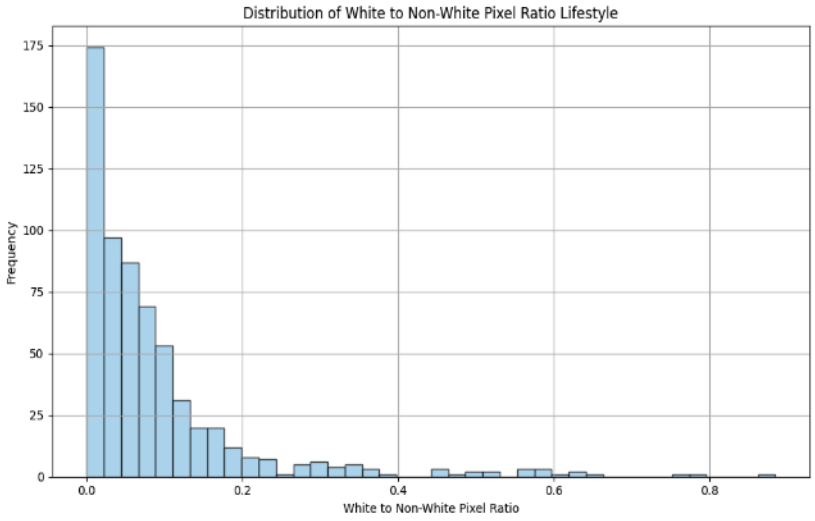


Figure 11. Ratio distribution of known lifestyle images

To calculate the threshold value, the ratio distribution of known lifestyle images, as shown in Figure 11, was utilised to support the analysis. Quartiles and the Interquartile Range (IQR) were identified from this distribution. Based on these values, an appropriate threshold of 0.27 was established. Images with ratios equal to or less than 0.27 are classified as lifestyle images, while those surpassing 0.27 are considered product images.

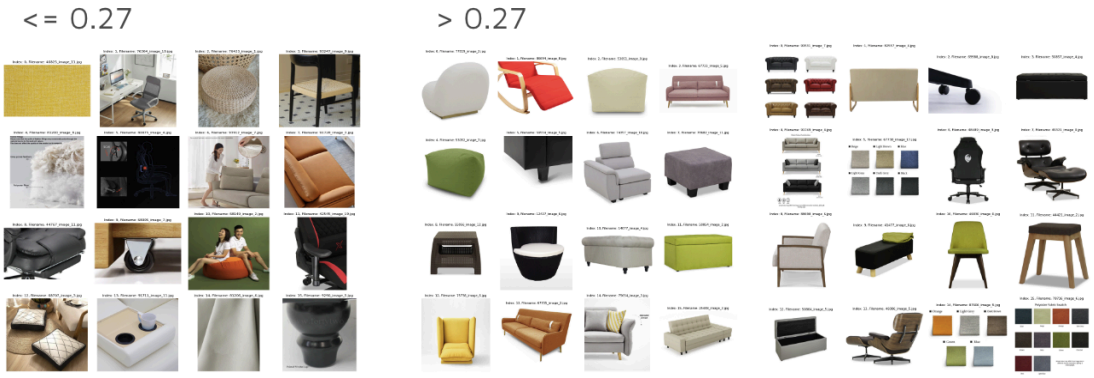


Figure 12. Accurate differentiation of lifestyle and product images

To validate the effectiveness of this method, a list of product and lifestyle images differentiated using the above threshold was compiled (Figure 12). This figure demonstrates the successful differentiation between lifestyle and product images, effectively addressing the initial problem of mislabeled data. This leads to the second problem concerning the presence of extraneous text in quality product images suitable for training.

4.3. Removing Text from Images

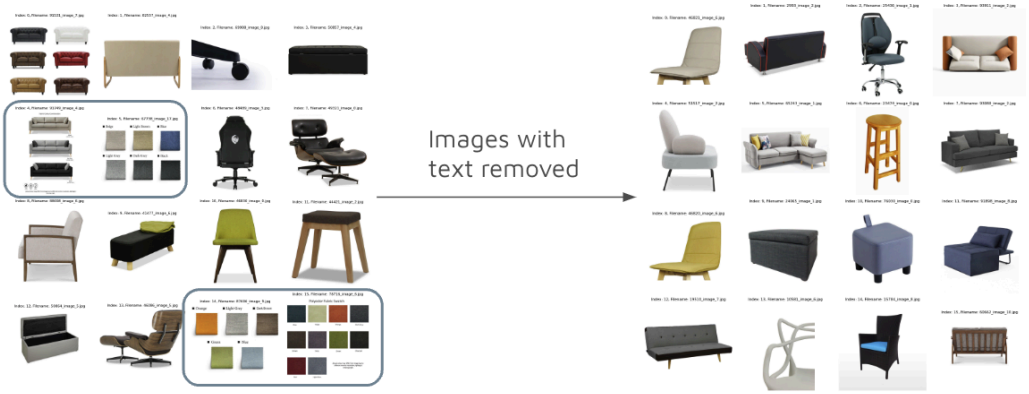


Figure 13. Text removal process

To remove extraneous text from good product images, Tesseract was utilised for text detection within these images and its removal. With that accomplished, attention shifted towards rectifying the final issue within the dataset: inconsistent aspect ratios.

4.4. Consistent Aspect Ratios

The resolution of the input images to the model has to be kept consistent, thus, it is necessary to determine the input shape and resolution of the images for the model.

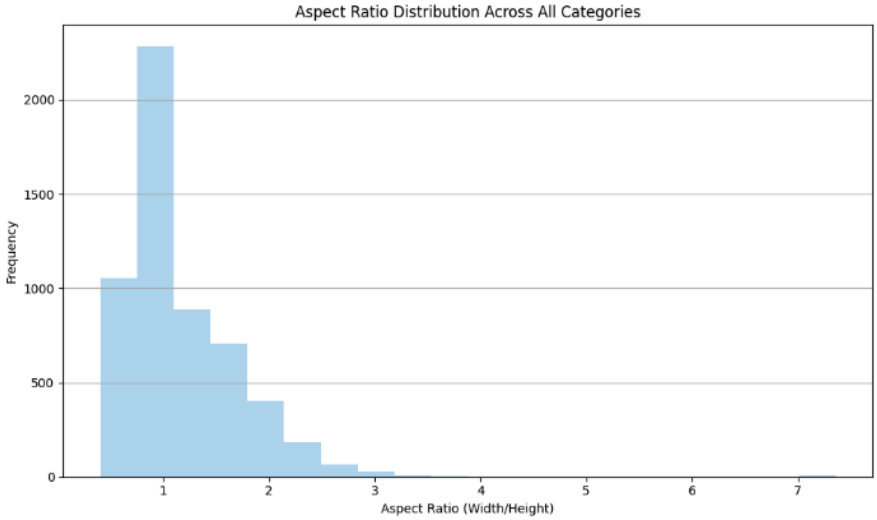


Figure 14. Aspect ratio distribution across all images

To determine the best aspect ratio to use, the distribution of aspect ratios for the entire image dataset was plotted. As illustrated in Figure 14, the majority of images have an aspect ratio of one, indicating a square shape. Hence, a fixed aspect ratio of 512 x 512 pixels was selected for the model's input.

Figure 15. Padded, resized and centred image to a 512 x 512 resolution.

All images are then resized, centre cropped and padded with white background to the desired aspect ratio.

5. Model

To recommend similar furniture items based on user selection, the approach consists of two key processes: (1) fingerprinting a furniture by representing it as a feature vector and (2) employing clustering techniques to group similar furniture while separating dissimilar ones.

5.1. Model Architecture Selection

Given the two requirements, the decision was made to utilise concepts from both transfer and contrastive learning. Firstly, employing a pre-trained model such as ResNet-50 to fingerprint an image is logical due to time and hardware constraints. This approach enables the utilisation of pre-existing knowledge. Secondly, to train customised fully connected layers for furniture clustering, contrastive learning using a siamese network is the most suitable choice.

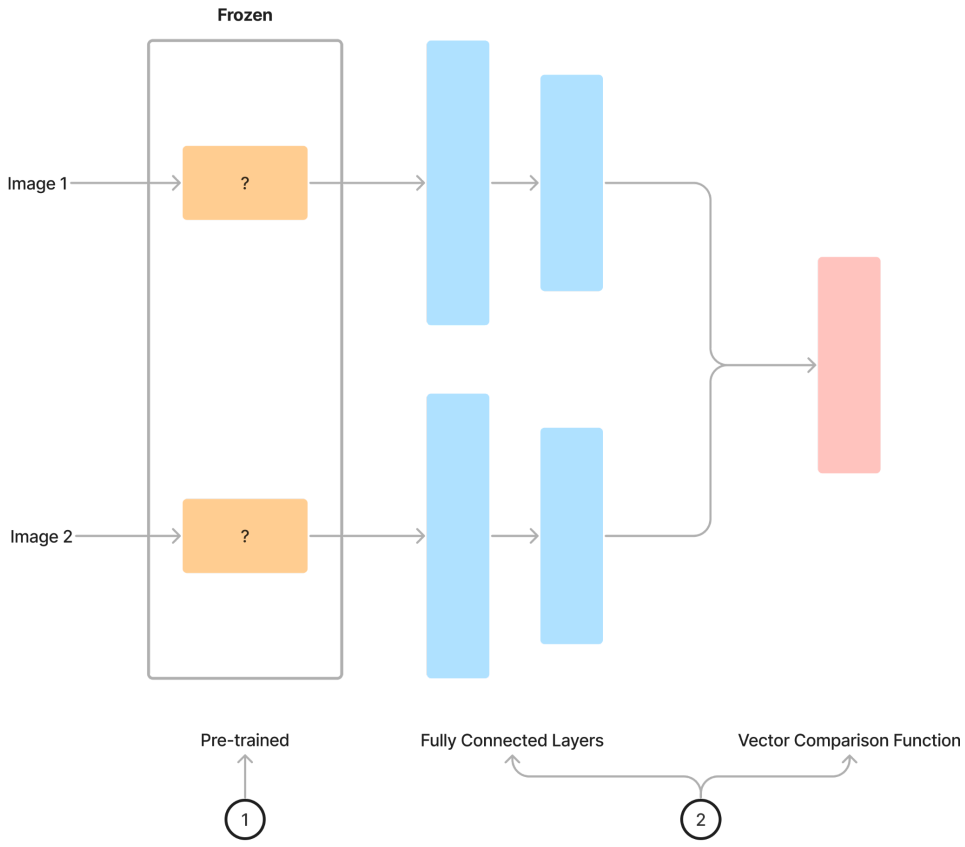


Figure 16. High level overview of our siamese network.

As shown in Figure 16, the Siamese network model operates by freezing the weights of a pre-trained model then inputting those embeddings into trainable fully connected layers.

The project underwent two major iterations of model architecture and pre-trained model selection. The first iteration utilised image pairs and the ResNet architecture, while the second iteration expanded to image triplets and incorporated the BLIP model for improved performances. Adjustments were made based on preliminary results to refine our approach and enhance model accuracy.

5.2. Iteration 1 (with ResNet and Pair of Images)

5.2.1 ResNet

ResNet, short for Residual Network, is a type of Convolutional Neural Network (CNN) known for its ability to handle deep architectures without running into the vanishing gradient problem. (He et al., 2015) In our Siamese network, it is used as a feature extractor in each branch of the network, effectively learning and highlighting the subtle differences and similarities among furniture images.

5.2.2. Model Architecture

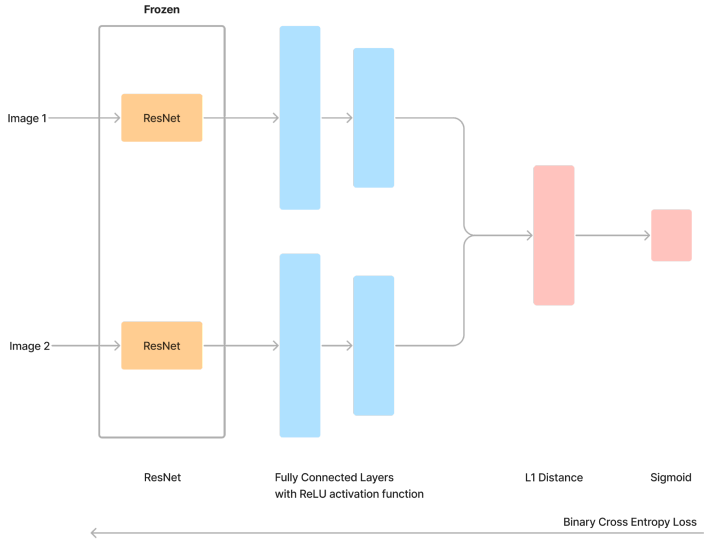


Figure 17. High level overview of our model training architecture

In the initial iteration, a pre-trained ResNet model was incorporated into a custom architecture designed to process pairs of furniture images. To prepare the data, image pairs were labelled as similar or dissimilar based on whether they belonged in the same sub-category, assigning similarity scores of 1 for similar pairs and 0 for dissimilar pairs. The ReLU activation function was selected since it has an unbounded positive upside which would lead to more complex features representations. The feature representations were then fed through an L1 distance function combined with a Sigmoid function to generate bounded

similarity scores between 0 and 1. Finally binary cross entropy was used as the loss function for calculating the loss.

5.2.3. Learnings

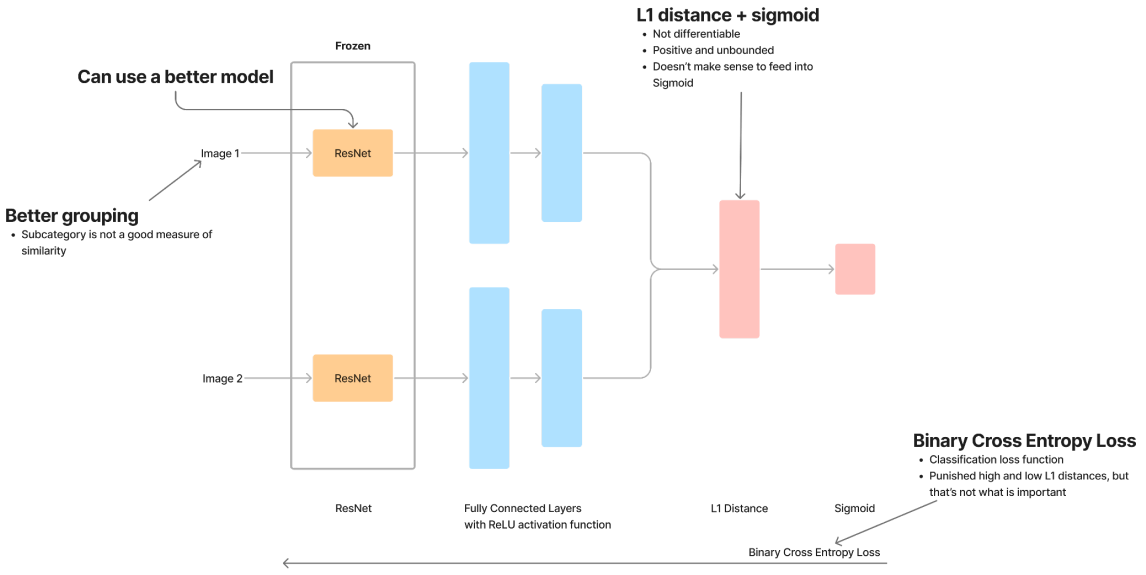


Figure 18. Summary of our learnings.

Inherently there are a few major issues with the first iteration:

1. Using L1 distance with Sigmoid
    a. The range of a L1 distance function does not fit the domain of a Sigmoid function.
    b. L1 distance is not differentiable everywhere and is unbounded. This would make training a generalised model harder.
2. Using Binary Cross Entropy Loss function
    a. Binary Cross Entropy Loss function punishes the model purely for a high or low L1 distance but does not care about relative distances between images which is our main goal.
3. Using sub-category as a metric for similarity
    a. Does not reflect the true style similarity.

Among these major issues, we also realised that we could use a better pre-trained model such as BLIP.

5.3. Iteration 2 (with BLIP and Triplets)

5.3.1 BLIP

BLIP (Bootstrapped Language-Image Pre-training) is a deep learning model that is designed to understand and generate captions for images by learning how to interpret images alongside with texts. BLIP is able to integrate the understanding

of what is depicted in an image to how it is typically described in text. Thus, BLIP is able to capture contextual and semantic elements within an image, potentially offering rich feature representations of our furniture, leading to more accurate similarity comparisons and ultimately better furniture recommendations (Li et al., 2022).

BLIP's architecture consists of multiple encoders to encode visual and text features. For the scope of this project, only the output of the Image encoder will be utilised as feature vectors of the furniture image. Leveraging insights from the previous iteration, the pre-trained ResNet model was substituted with the pre-trained BLIP model for this iteration.

5.3.2. Model Architecture

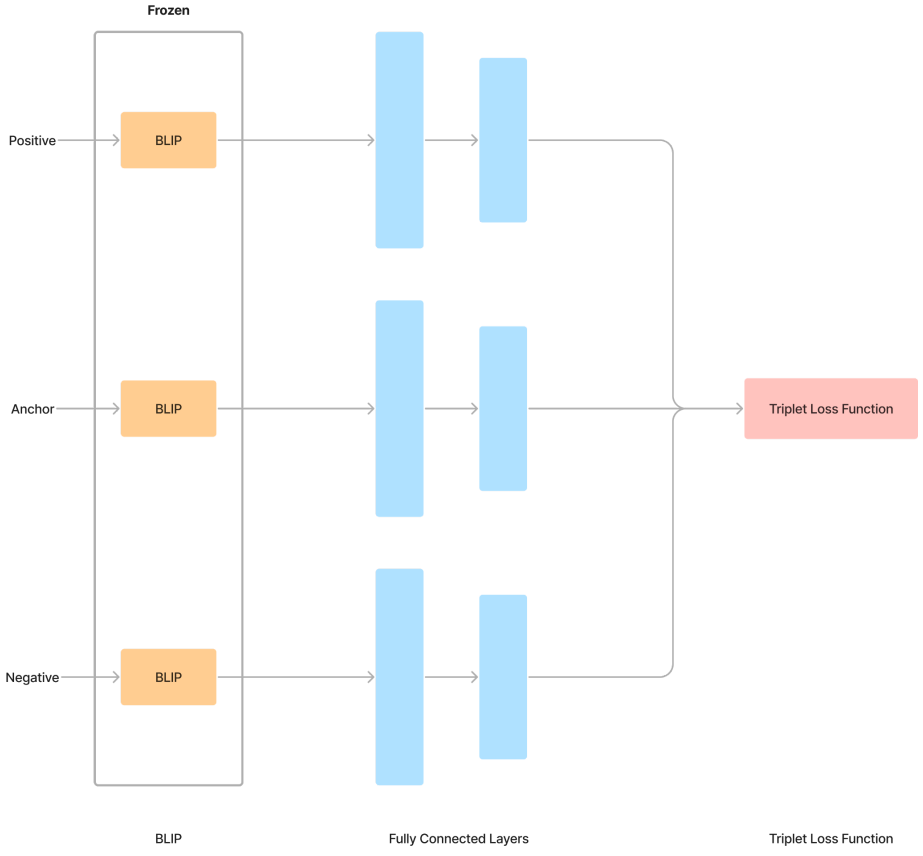


Figure 19. Model architecture of siamese network with BLIP.

In this iteration, the strategy involves freezing the weights of the BLIP model, focusing our training efforts on the fully connected layers, accompanied by a triplet loss function at the end.

6. Data Preparation

To effectively utilise a triplet loss function, the dataset was adjusted to support triplets, each comprising an anchor, a positive (similar), and a negative (dissimilar) furniture image. The steps involved include: (1) Improving grouping of similar furniture through manual intervention, (2) Splitting the dataset into training, validation and test datasets with an 80:10:10 split, (3) Generating triplets from the training dataset and (4) Preprocessing and caching of all images with BLIP for quicker training velocity. This setup aims to refine the model's ability to discern nuanced differences and similarities between furniture items.

6.1. Manual Grouping of Similar Furniture

To improve the accuracy of similar furniture grouping, a team member performed manual classification on Figma, based on criteria such as form, pattern and material, ensuring a consistent grouping standard across the dataset. A single-person execution was maintained to uphold consistency in groupings and style interpretation. This method of grouping was selected for its efficiency and simplicity in establishing a reliable source of truth.

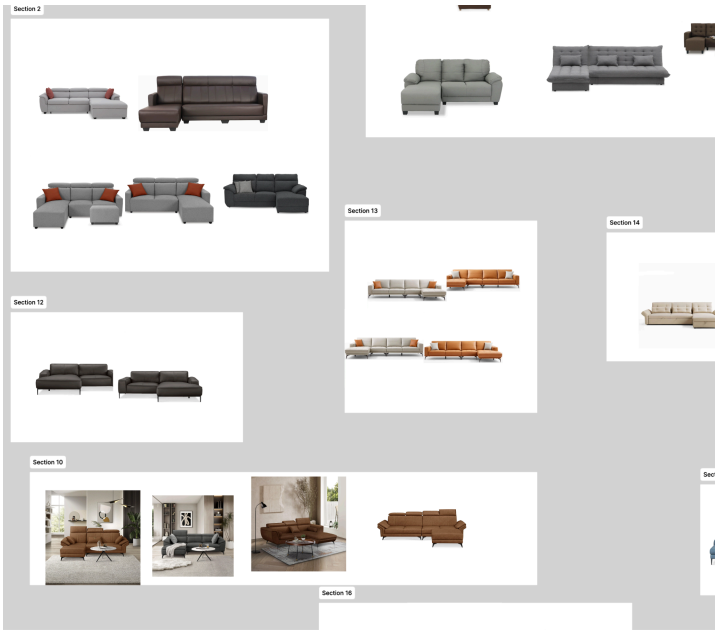


Figure 20. Example of manual grouping on Figma.

As seen in Figure 20, form, pattern and material are important factors when grouping similar furniture together within a subcategory. For example, a leather L-shaped sofa with recessed backrest is different from a fabric L-shaped sofa with flat backrest. This process ends in the generation of a comprehensive JSON file, detailing the group labels for each item, facilitating accurate triplet formation.

6.2. Splitting the Dataset

To ensure the generalizability of the model, evaluation against unseen products was a priority, necessitating a careful splitting of our dataset into training, validation, and testing sets while maintaining an 80:10:10 distribution. To streamline the training process within project time limits, the dataset curation involved including only a single, primarily front-facing image per furniture item, ensuring no overlap across datasets.

It was also crucial for the training dataset to cover all furniture groups comprehensively. The test and validation dataset also needs to have products that are not in the training set. It is important to note that the dataset split occurs at the product level rather than the image level.

Each group was represented by at least two furniture items to facilitate triplet formation. Within each triplet, the positive and anchor images originated from the same group, while the negative image was selected from a different group. Groups containing only one item were omitted to uphold this requirement. The dataset splitting process is outlined in Figure 21 below.

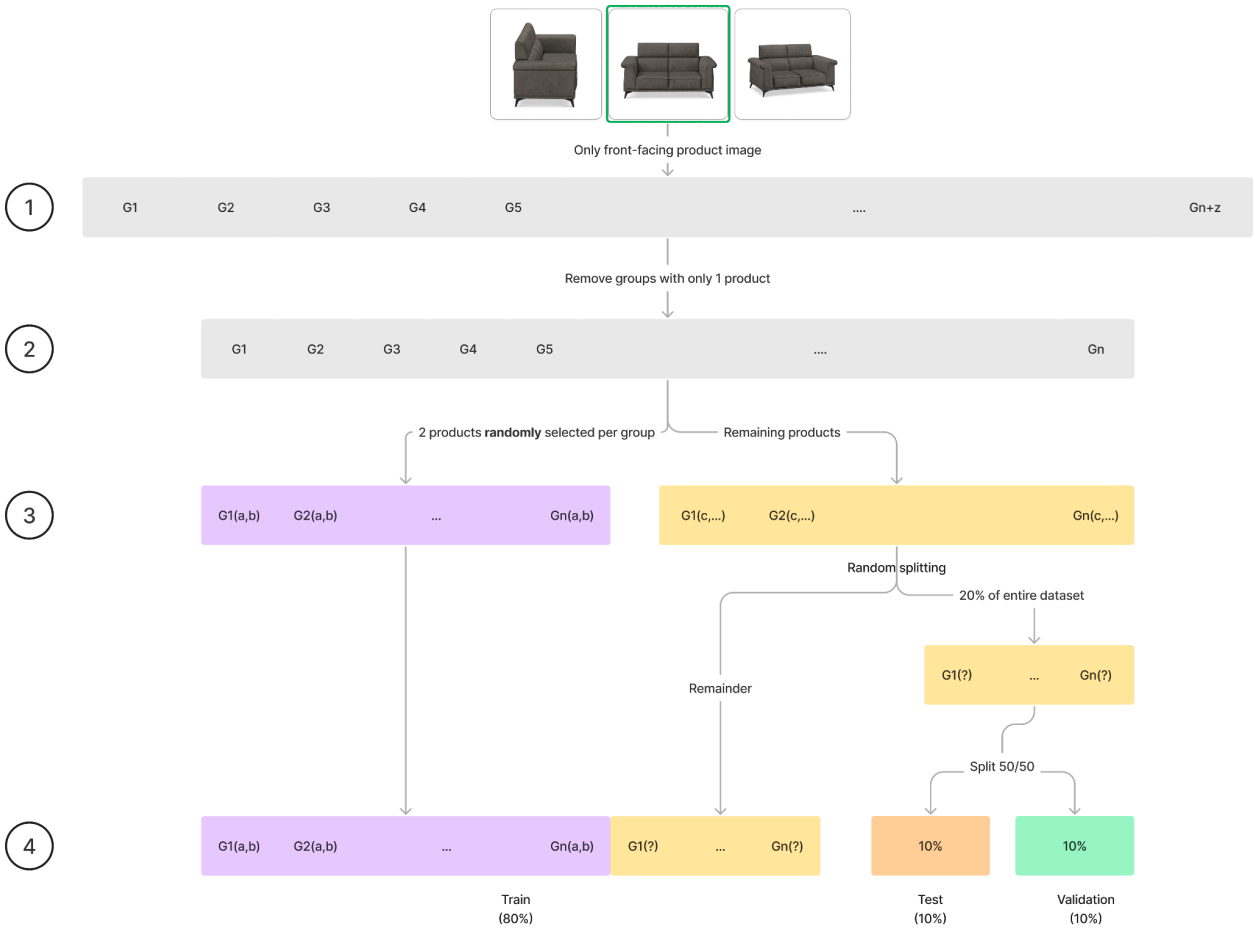


Figure 21: Illustration of how we split the dataset.

The systematic splitting process is illustrated in Figure 21. At the first step, only the front-facing product image of each product across all groups is retained. Each group is represented as “G1” in the diagram. Subsequently, groups represented by only a single product were excluded. Lastly, to ensure that all groups are represented in the training dataset, two items from each group are randomly selected to be part of the initial training dataset, represented by G1(a,b) and Gn(a,b) highlighted in purple. For the remaining products, 20% of products are reserved to be evenly split between testing and validation sets. The residual products are then added back to the initial training dataset, achieving the desired 80:10:10 ratio.

6.3. Generation of Triplets

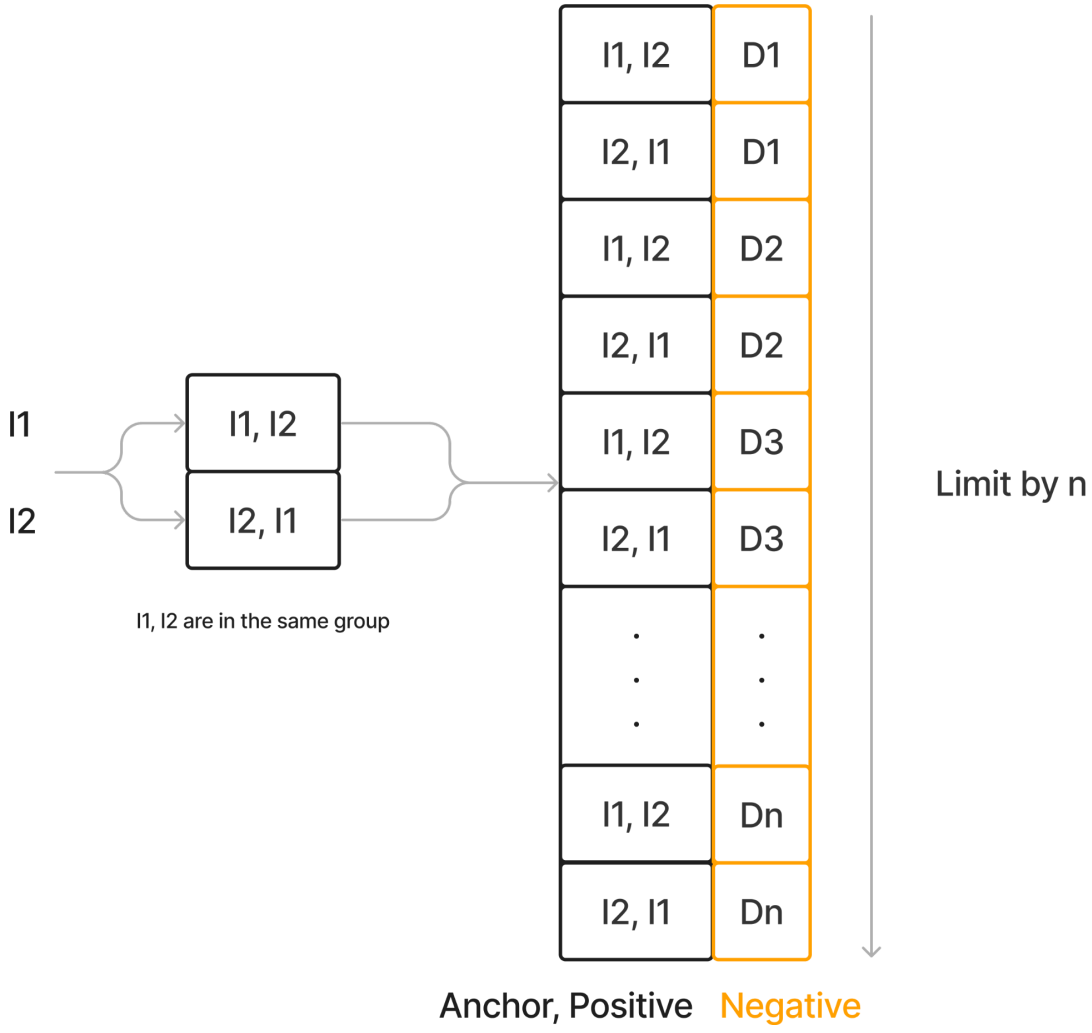


Figure 22: Illustration of how triplets are generated.

Next, triplets need to be generated for the training dataset. Within a triplet, an anchor image will be accompanied by a positive image from the same group and a negative image from a different group. Due to the nature of the dataset, every anchor-positive pair will be accompanied by many permutations of negative images to form triplets. To streamline the development process, the decision was made to reduce the number of triplets that can be formed for each anchor-positive pair. As shown in Figure 22, each anchor-positive pair is limited to *n* number of randomly selected negative images.

6.4. Caching BLIP Embeddings

To expedite training, all images in the dataset are processed with BLIP. The output embeddings would be stored in a CSV file, acting as cache during training. With our data preparation pipeline finalised, we will discuss our evaluation methodology in the next section.

7. Evaluation Methodology

To evaluate the efficacy of our modified Siamese network incorporating BLIP's image encoding capabilities, we conducted a comparative analysis against the baseline BLIP model. This comparison aims to assess the improvements or regressions in performance resulting from our architectural changes. Our evaluation methodology involves: (1) Predicting the nearest group for each validation or test image, (2) With the predicted group and true group label of each image, we calculate the precision score, a metric of success for our model.

7.1. Predicting the Nearest Group

Given an image of a furniture, we wanted to evaluate if it is clustered among other products of the same group. Our trained Siamese model has a baseline understanding of the training data and has clustered the products to their respective groups. We can use that understanding to evaluate how well the model predicts the groups of new products. In order to do this, we use the training data to find centroids of each group as shown in Figure 23. Then, given an unseen product image, we find the closest centroid whose group is the predicted group.

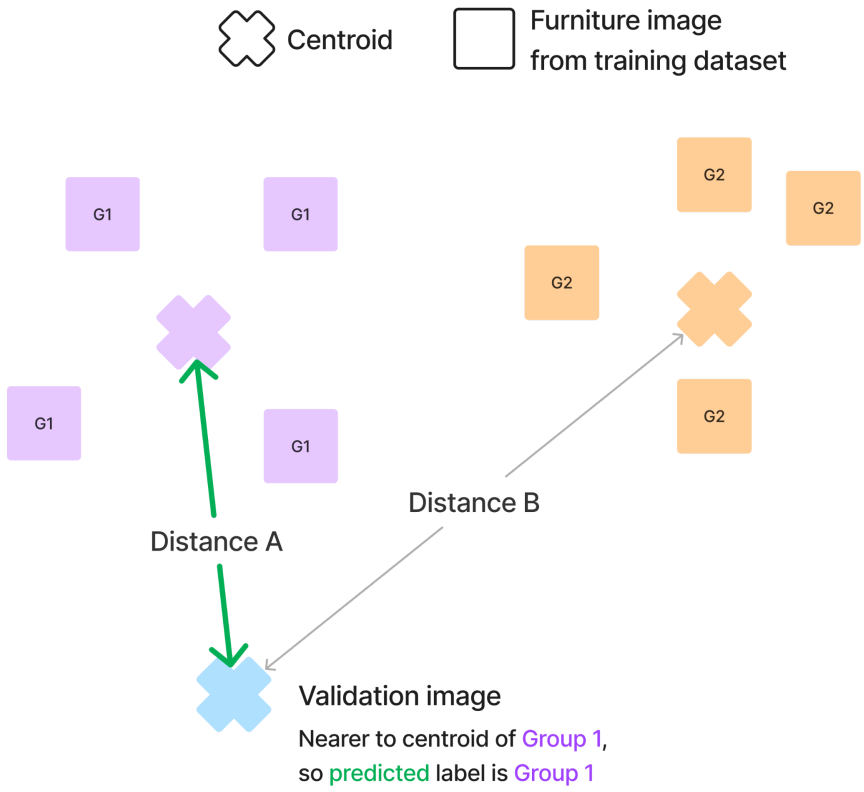


Figure 23: Finding nearest centroid for group prediction.

The nearest predicted group for the image is found by calculating the euclidean distance of its feature vectors against the centroids of the training dataset as shown in Figure 23. The group with the shortest euclidean distance represents the predicted group for the image.

7.2. Precision Score

With the predicted and true group labels of the furniture images, we calculated the F1, precision and recall score.

Given the importance of minimising irrelevant recommendations to ensure the utility and user satisfaction of our recommendation system, we focused on the precision score as our primary success metric. This metric effectively measures our model's accuracy in recommending only relevant furniture items, thereby reducing false positive rates where incorrect recommendations could significantly impact the model's perceived efficacy.

With our evaluation method established, we proceeded to optimise the model's hyperparameters and architecture to enhance its performance.

8. Architectural and Hyperparameter Tuning

Given the time constraint of our project, we needed to iterate quickly and explore multiple combinations of hyperparameters and architecture for our model. Therefore, we will perform hyperparameter tuning using a reduced version of the training data and evaluation will be done against the validation set.

To get the best performance and results from our model, we will need to tune its architecture and hyperparameters iteratively in a sequential manner.

To achieve that, we undertook these steps:

1. Reduce the number of triplets generated from the training dataset for the tuning process.
2. Choose the optimal batch size based on the shortest time taken to train an epoch.
3. Find the optimal architecture and its learning rate.
4. Find the optimal triplet loss margin.

8.1. Generation of Reduced Triplets

Using the same method as described in the ‘Generation of Triplets’ section, we reduced the number of training triplets formed to approximately 10% of the original size, amounting to 450,000 triplets. This reduction was predicated on a distribution analysis ensuring that the smaller subset mirrors the diversity and distribution of the full training set.

8.2. Choosing Optimal Batch Size

Batch size plays a key role in determining the training time and computing resource consumption (Godbole et al., n.d.). To minimise training time and conserve computational resources, we systematically evaluated the time taken for 1 epoch of training across various batch sizes - 32, 64, 128, 256, 512, 1024.

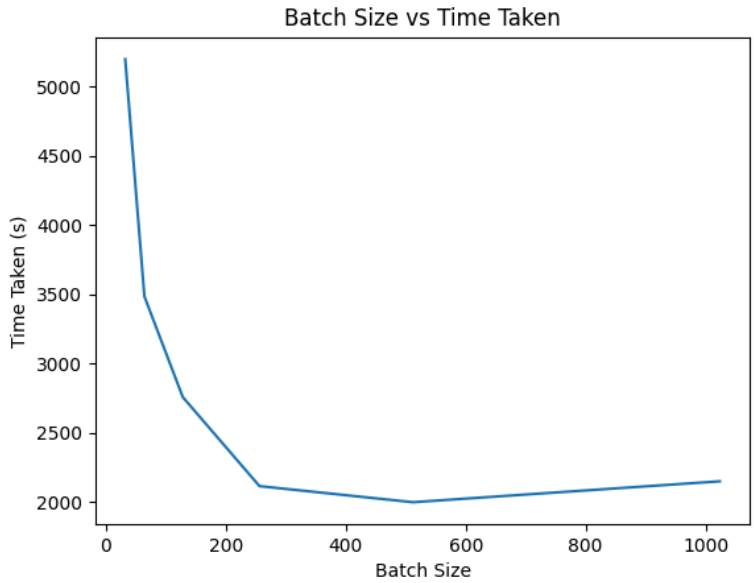


Figure 24. Time taken for 1 epoch for each batch size.

Our analysis, illustrated in Figure 24, indicated a batch size of 512 offered the most efficient training duration. Hence this batch size was thus selected for subsequent tuning processes.

8.3. Tuning Architecture and Learning Rate

After the batch size has been determined, we proceed to find the optimal architecture of our fully connected layers. Getting the best architecture before other hyperparameters is important as the architecture affects all downstream hyperparameters.

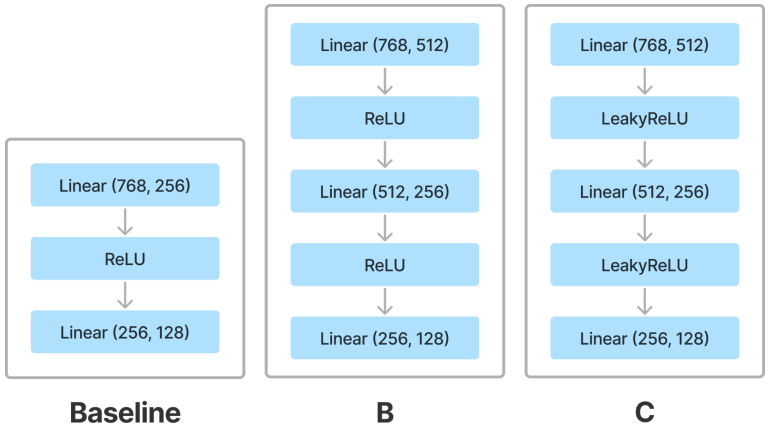


Figure 25. The 3 different architectures of our fully connected layers we tested on.

We evaluated three distinct architectures, as detailed in Figure 25, varying in layer complexity and activation function used. By introducing more layers to our model, we

hoped that it can better capture the nuanced differences between furniture to improve precision score. Furthermore, the leaky ReLU activation function is able to represent negative values unlike the ReLU activation function, thus, this allows for neurons in our model to contribute more to the overall learning process, potentially improving the precision score of our model.

In addition, hyperparameters such as learning rate have a huge impact on the performance of a model as this affects the rate of gradient descent of the model. An optimally tuned learning rate ensures that our model is able to converge to an optimal performance efficiently. Thus, we wanted to find the combination of learning rate and model architecture that gives us the highest precision score. The below Figure 26 shows the values of the learning rate we used.

Learning Rate	0.00001	0.000001	0.0000001

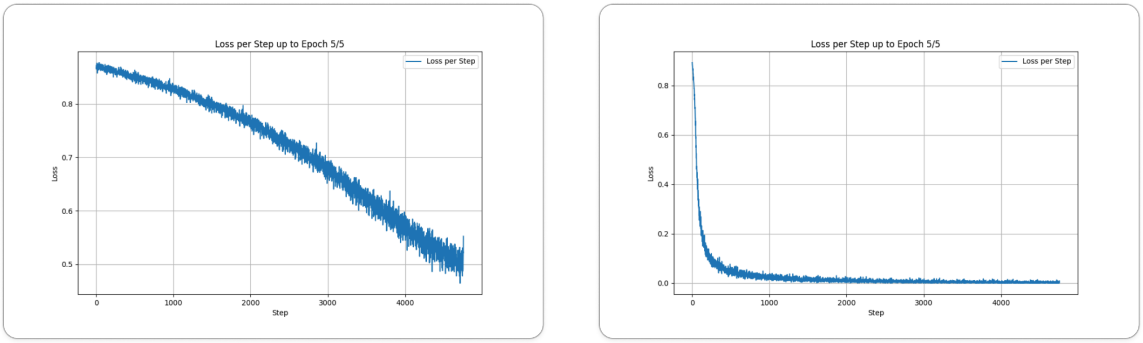
Figure 26. Learning rate used.

Next, each iteration of the architecture was run with all values of learning rate with these hyperparameters: 5 epochs, triplet loss margin of 1.0 and batch size of 512. From the results, there was a tie between two combinations of learning rate and model architecture for the highest precision score obtained as shown in Figure 28 (All results are shown in Appendix 2).

Architecture	Learning rate	Average loss	Precision
Baseline	0.000001	0.089	0.52
B	0.0000001	0.714	0.55
C	0.00001	0.030	0.55

Figure 27. Precision scores of the top performing learning rate for each model architecture.

GIven the big difference in average loss for both best performing results, we investigated further and looked at the loss graph for the results as shown in Figure 28 below.



B **C**

Figure 28. Loss graph for results of architecture B and C.

In the comparative analysis of both graphs, we observed that the loss trajectory for model C has already reached convergence while the loss for model B has not approached convergence. This divergence in loss trajectories signals a potential for further improvement in performance for model B through extended training over additional epochs. The parity in precision scores, coupled with model B's potential for improvement, led us to choose model B as the optimal architecture, as shown in Figure 29 below.

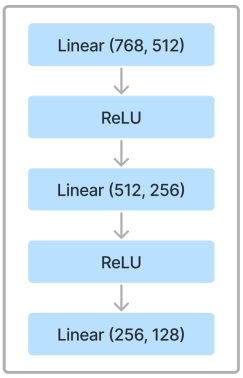


Figure 29. Optimal architecture and learning rate chosen.

8.4. Tuning Triplet Loss Margin

Finally, due to the time constraints of the project, we focused on optimising the triplet loss margin within a manageable range. Testing an infinite number of values was not feasible, and setting the margin too high might complicate training and even prevent the model from converging. Therefore, we selected a fixed set of values to test: 0.5, 1.0, 1.5, 2.0, 2.5 and 3.0. Our aim was to identify which setting would maximise the precision score, as shown in Figure 30 below.

Triplet loss margin	Precision
0.5	0.40
1.0	0.55
1.5	0.43
2.0	0.34
2.5	0.42
3.0	0.43

Figure 30. Results with different triplet loss margin tested.

The analysis indicated that a margin of 1.0, in conjunction with our previously determined optimal learning rate and model architecture, yields the highest precision score of 0.55.

8.5. Final Architecture and Hyperparameter

Thus, the final hyperparameters and architecture we have selected are shown below in Figure 31.

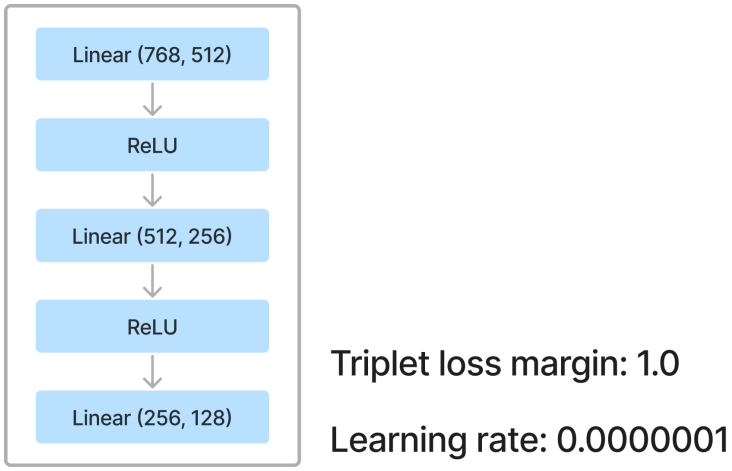


Figure 31. Final hyperparameters and architecture chosen.

9. Training

With the finalised hyperparameters and model architecture obtained, we wanted to understand how our model would fare against the pre-trained BLIP model, thus, we trained with the triplets generated for full training for 5 and 10 epochs, then, we evaluated our model against unseen images that were represented in the test dataset. The results from the training were as such:

Epochs	Average Loss	Precision
5	0.194	0.39
10	0.118	0.43

Figure 32. Results from training with 5 and 10 epochs.

	Precision
BLIP	0.46

Figure 33. Baseline performance of the pre-trained BLIP model.

Comparing the results for both 5 and 10 epochs against the results of the pre-trained BLIP model, it can be seen that our model is achieving a lower precision score. However for our model, there seems to be an improvement in precision score with more epochs, indicating that our model might be able to improve further with more epochs. To verify our hypothesis, we ran our training again with 15 and 20 epochs respectively, shown in the Figure 34 below.

Epochs	Average Loss	Precision
15	0.087	0.41
20	0.069	0.42

Figure 34. Results from training with 15 and 20 epochs.

The results from further training showed a drop in performance after training our model for 15 epochs and above, thus, this is an indication that our model might be overfitting and we will need to explore other ways we could improve our model.

10. Improving the Final Model

To improve our model, we tried to mitigate overfitting on the training dataset by introducing random dropout to our model as seen in Figure 35 below. In total, we made two iterations of improvements to our model. First, we introduced random dropout, however, this made our model perform worse with more epochs. Secondly, we introduced random dropout but decreased the learning rate, this approach gave us the best performance for our model.

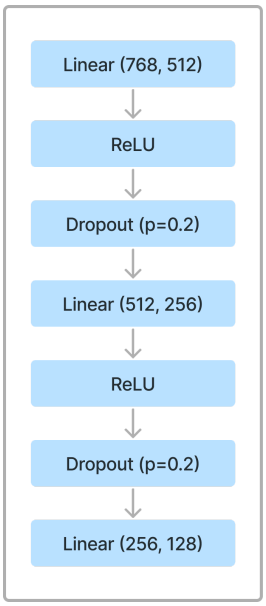
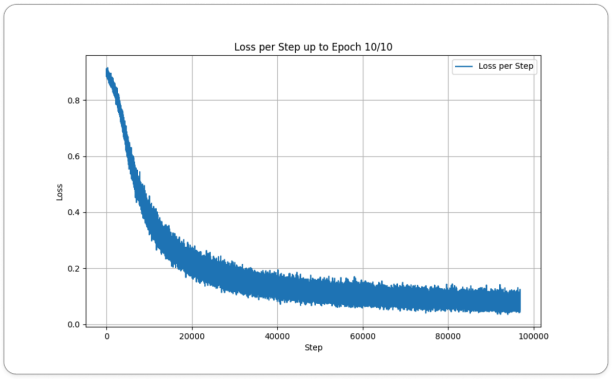


Figure 35. Model architecture B with random dropout.

10.1. First Iteration

With the below hyperparameters, we ran the training on 5 and 10 epochs and compared the results as seen in Figure 36 below:

- Triplet loss margin of 1.0
- Learning rate of 0.0000001
- Batch size of 512
- Dropout rate of 0.2



Epochs	Average Loss	Precision
5	0.276	0.43
10	0.187	0.36

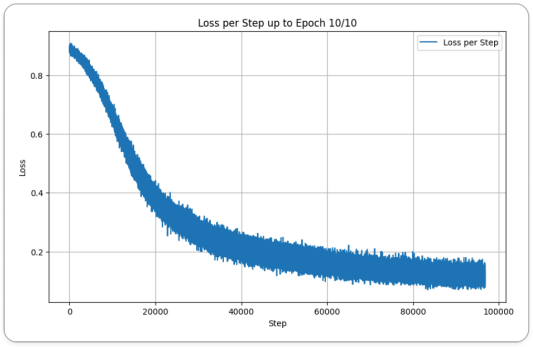
Figure 36. Results and loss graph from running with 5, 10 epochs and random dropout.

As expected, adding dropout to our model introduced noise during training, which is reflected in the higher average loss. However, the precision score did not improve but in fact worsened with more epochs, and the optimal precision score here is not better than our initial model in the previous section. As adding random dropouts adds noise to the training, this makes the training process for the model more challenging. Thus, we might need to reduce the learning rate of the model.

10.2. Second Iteration (with lower learning rate)

With the learnings from the previous section, we experimented with lowering the learning rate while maintaining the same dropout rate as before, thus, we came up with the below parameters, ran the training on 5 and 10 epochs and compared the results as seen in Figure 37 below:

- Triplet loss margin of 1.0
- Learning rate of 0.00000005
- Batch size of 512
- Dropout rate of 0.2



Epochs	Average Loss	Precision
5	0.409	0.45
10	0.280	0.37

Figure 37. Results and loss graph from running with 5, 10 epochs and random dropout and lowered learning rate.

This time, there is an improved initial precision score. Adding dropout helped mitigate overfitting at early epochs, achieving a higher precision score with just 5 epochs as compared to the original model architecture without dropout. However, the precision scores declined with additional epochs, suggesting that the current dropout rate might not be optimal, or that our model has reached its optimal performance with lesser epochs. Nonetheless, it is clear that the inclusion of dropout has helped enhance our model's generalisability with fewer training iterations, showcasing dropout as a viable method for improving a model's performance. Thus, we will be using the weights obtained with 5 epochs as our final model configuration.

10.3. Learning

Despite our efforts in hyperparameter tuning and incorporating random dropout into our model, it continues to converge and overfit as early as 5 epochs. This persistent early convergence, even after reducing the learning rate and enhancing dropout, suggests that additional tuning may not substantially improve the performance of our model. The rapid achievement of optimal precision scores indicates that our training dataset might not be challenging enough, allowing the model to quickly learn and exhaust the range of features available. This scenario implies that the dataset might be too simplistic for the model, which manages to capture the essential data characteristics within a very short training period.

11. Potential improvements

With the time constraint of the project, there were multiple optimisations and techniques which our team did not explore to improve the performance of our model. These improvements can be split into several pipelines: data preparation and preprocessing pipeline, architecture and hyperparameter tuning, and evaluation.

11.1. Data Preparation and Preprocessing

We could increase the diversity of the images by including different types of images already available for each furniture piece, such as close-ups and multiple angles, into our training dataset. This would help our model to recognise the furniture piece from various perspectives.

Furthermore, implementing image augmentations such as noise addition, rotation, blur and random erasing could further challenge our model during training, potentially improving its generalisability and performance on unseen images.

11.2. Tuning Architecture and Hyperparameters

We could explore more combinations of model architecture and hyperparameters, thus, utilising techniques such as grid search or random search to systematically explore a wider range of hyperparameters to find the optimal values. Next, adding validation loss metrics will provide us with deeper insights into the model's learning progress and help us to understand when our model starts to overfit. Last, other than adding random dropouts, we could introduce an early stopping mechanism. This will allow us to halt training as soon as the model starts to overfit or once it has reached its optimal performance on the validation dataset.

11.3. Evaluation

We could increase the diversity of our test dataset as well. The test dataset can include a wide variety of image types, conditions that our model may encounter in practical applications. This can involve images which have different lighting conditions and backgrounds. This will allow us to test the robustness of our model too.

12. User interface

To demonstrate the capabilities of our improved model, we decided to showcase it through a Gradio app.

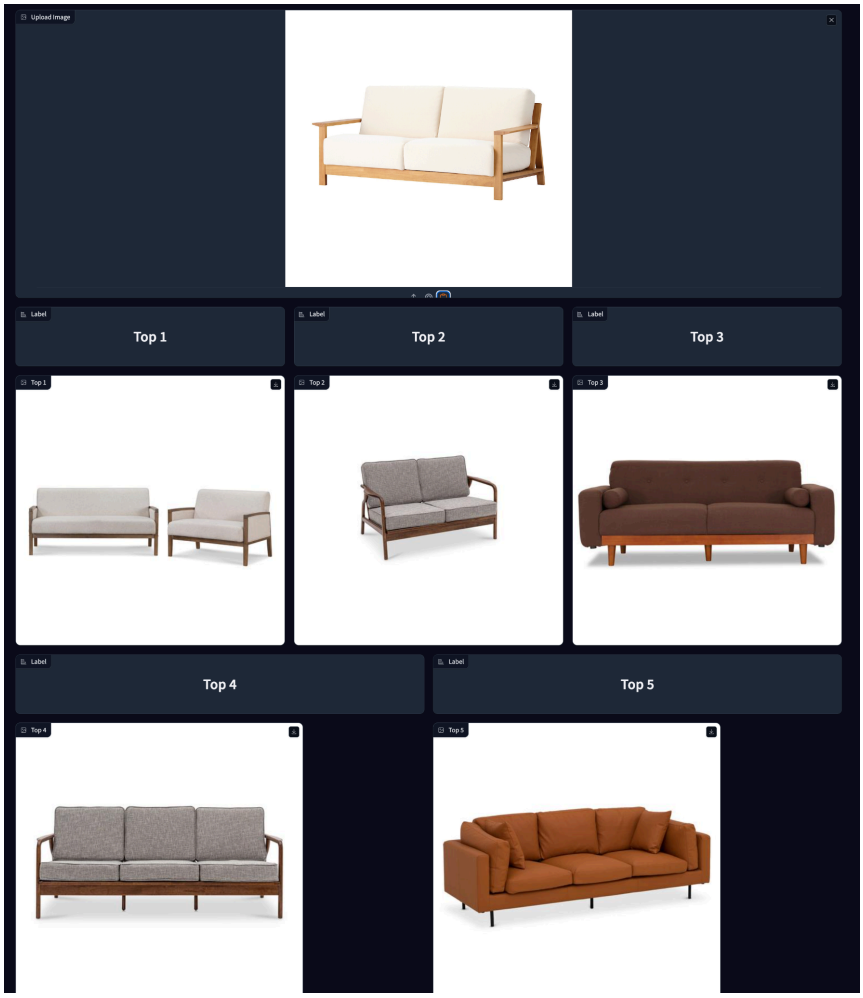


Figure 38. Top 5 similar furniture recommended.

With our Gradio application, users will be able to upload an image of a piece of furniture, then, the top 5 similar furniture will be recommended as shown in Figure 38 above.

References

Godbole, V., Dahl, G, E., Gilmer, J. Shallue, C, J., Nado, Z. (n.d.). Deep Learning Tuning Playbook.
https://github.com/google-research/tuning_playbook#choosing-the-batch-size

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition.
https://doi.org/10.1109/cvpr.2016.90

Li, J., Li, D., Xiong, C., & Xiong, S. (2022). BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation.
https://doi.org/10.48550/arXiv.2201.12086

Appendix

Appendix 1: Link to our Github repository
https://github.com/Fruittips/cds-furniture

Appendix 2: Results from finding optimal architecture and learning rate

Baseline

Learning rate	Average Loss	Precision
0.00001	0.197	0.44
0.000001	0.089	0.52
0.0000001	0.205	0.46

B

Learning rate	Average Loss	Precision
0.00001	0.031	0.49
0.000001	0.189	0.37
0.0000001	0.714	0.55

C

Learning rate	Average Loss	Precision
0.00001	0.030	0.55
0.000001	0.181	0.47
0.0000001	0.722	0.36