FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Artificial Intelligence in Home Interior Design: Neural Networks for Combinations of Patterns and Accessories
Student:	Viktoriia Sukha
Supervisor:	Ing. Mgr. Ladislava Smítková Janků, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

This work is focused on application of neural networks to the problem of a selection of combinations of patterns of decorative textiles and accessories in the design of selected parts of the home interior (living room, children's room).

1. State of the art in the areas of an application of NN's, particularly in the design.
2. In collaboration with the supervisor, design a methodology for generating combinations of patterns and accessories for interior design.
3. From publicly available resources, create datasets of suitable patterns and accessories and datasets.
4. Design a neural network and implement it using existing tools or libraries.
5. Perform experiments and evaluate them.

Electronically approved by Ing. Karel Klouda, Ph.D. on 7 February 2022 in Prague.

Bachelor's thesis

ARTIFICIAL INTELLIGENCE IN HOME INTERIOR DESIGN: NEURAL NETWORKS FOR COMBINATIONS OF PATTERNS AND ACCESSORIES

Viktoriia Sukha

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: doc. Ing. Mgr. Ladislava Smítková Janků, Ph.D.
May 11, 2023

Citation of this thesis: Sukha Viktoriia. *Artificial Intelligence in Home Interior Design: Neural Networks for Combinations of Patterns and Accessories* . Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

I would like to thank my boyfriend and friends who supported me even in the worst conditions.

Declaration

In Praze on May 11, 2023 ....

Abstrakt

Tato bakalářská práce se zaměřuje na využití neuronových sítí při navrhování vybraných prvků interiéru domu. Hlavním cílem práce je extrahovat vlastnosti jako barva, styl a materiál a na základě těchto vlastností vytvářet kombinace vzorů a doplňků pomocí grafových neuronových sítí. Metodika této práce zahrnuje použití VGG pro extrakci rysů, YOLOv7 pro detekci objektů a GCN pro predikci kompatibility mezi prvky interiéru.

Klíčová slova neuronové sítě, interiérový design, kombinace vzorů, extrakce funkcí, barva, styl, materiál, VGG, YOLOv7, GCN, GNN

Abstract

This bachelor's thesis delves into the utilization of neural networks in designing specific components of home interiors. The objective of this research is to extract features including color, style, and material, and employ these features to construct a graph neural network that generates combinations of patterns and accessories. The methodology adopted in this work entails the utilization of VGG for feature extraction, YOLOv7 for object detection, and GCN for compatibility prediction.

Keywords neural networks, interior design, pattern combinations, feature extraction, color, style, material, VGG, YOLOv7, GCN, GNN

Chapter 1
Introduction

In this chapter, we present an overview of the motivation behind this work, define its goals, outline the methodology implemented, and provide an overview of the structure of the entire study.

Artificial intelligence (AI) has shown remarkable success in image recognition, a field known as computer vision. This ability has found applications in various domains, providing solutions to many real-world problems. In this work, we aim to apply computer vision to the field of interior design, with the goal of significantly reducing the time required for selecting interior elements based on user preferences.

Artificial intelligence is used extensively in different areas, including interior design. Creating AI-assisted interior design faces many challenges, such as recognizing different styles, color combinations, recognizing shapes and objects, combining these elements with each other, and much more.

Using existing libraries and tools, we design a neural network that combines patterns and accessories based on user preferences, taking into account their preferred style, colors, and materials. The neural network generates a set of patterns and accessories that fit these requirements and are combined with each other in a coherent manner.

Our motivation for this work stems from our belief that the future lies in the widespread use of neural networks in various fields. Correctly applying these techniques has the potential to simplify and improve people's lives and work. Neural networks can efficiently analyze large datasets and organize them better than humans. Interior designers often spend considerable time searching for the right materials, patterns, and accessories. By automating this process, the productivity of designers can be significantly increased, while also saving time for people who choose interiors for their homes.

1.1 Goals

The primary aim of this study is to develop an algorithm capable of detecting objects in interior design, extracting specific features from patterns and accessories, and predicting the compatibility of these objects using the extracted features. Subsequently, the algorithm should be able to select the most suitable object based on the compatibility scores. To achieve these objectives, several steps must be taken:

1. Identify a suitable dataset and select an appropriate architecture for the object detector to be used in interior design contexts.

2. Develop or select a neural network architecture that can effectively extract features from interior design objects, focusing on characteristics such as style, color, and material.
3. Create a database of images and their corresponding features, which will serve as a foundation for generating various combinations.
4. Design an algorithm that can create meaningful combinations based on the available data, taking into account the compatibility of the objects in the design context.

1.2 Methods

The VGG neural network, a variation of CNN, was used in this work to extract features such as style, color, and materials from patterns and attributes. Tensor libraries were used to implement the VGG neural network and train the model.

To create the dataset for this work, web scrapers were developed to collect images from two different sources: Google Images and Houzz. The Selenium library was used to automate the scraping process and an HTML parser was used to extract image URLs from the webpages.

In addition to using the VGG neural network, the YOLOv7 object detection system was utilized to detect attributes of interior spaces and combine them in future models. YOLOv7 is a real-time object detection system that uses a deep neural network to detect objects in images. This system was trained using a dataset of annotated images and was used to detect specific attributes of interior spaces, such as furniture, decor, and lighting.

To manage the collected images and their associated features, the Pandas library was used to create a dataframe. The dataframe contained the image file path and features extracted by the VGG neural network, including the style, color, and material of the pattern or accessory.

To predict the compatibility of interior design objects, a Graph Convolutional Neural Network (GCN) was developed using the PyTorch-Geometric library. This library offers a variety of methods for deep learning on graphs, making it a suitable choice for this task.

Overall, these methods were used to create a dataset of patterns and accessories with their associated features, which were then used to train a neural network to generate combinations of patterns and accessories for interior design.

1.3 Structure of work

Research In this part, we will conduct a comprehensive review of existing research works that focus on problems related to interior design. We will explore the methods that have been used in these works and the ideas they have proposed. The goal of this part is to identify the strengths and weaknesses of the existing approaches and to use this information to inform the development of a new algorithm.

Theory In this part, we will study and describe the basics of neural networks, including the main algorithms and principles used in their development. We will explore the different types of neural networks, such as convolutional neural networks (CNNs) and Graph Convolutional Neural Network (GCN), which we will use in this work. The aim of this part is to provide a theoretical foundation for the algorithm we will propose in the implementation part.

Algorithm In this section, we present a overview of the implementation structure employed in this study. We delve into the key components of the algorithm, including the detector, models utilized for feature extraction, and the process of constructing a graph neural network to predict compatibility. Furthermore, we elucidate the interconnections between these components and provide insight into why it was necessary to implement each of them to achieve the objectives of this study.

Datasets In this section, we delve into the intricate process of dataset construction for all the neural networks implemented in this study. We address the challenges encountered during dataset creation, the techniques employed for data labeling, and the underlying principles guiding the selection of classes. Furthermore, we provide valuable insights into the utilization of pre-existing datasets, elucidating their relevance and applications within the context of this research.

Used architectures and libraries In this section, we discuss the architectures and libraries used in this study, explaining their functionality and purposes. We provide insights into why these tools were chosen to accomplish the objectives of this work.

Implementation In this section, we will describe the algorithm developed for creating combinations of patterns and accessories in interior design. The implementation process consists of several stages, including data collection, feature recognition using the VGG neural network, and the development of a Graph Convolutional Neural Network (GCN) for predicting compatibility between interior design objects.

Results and experiments In this part, we will present the results of the experiments conducted using the proposed algorithm. We will discuss what the algorithm can and can not do, and we will analyze the strengths and weaknesses of the algorithm. The aim of this part is to evaluate the effectiveness of the proposed algorithm and to provide insights for future research in this area.

Chapter 2
Research

In this section, we are going to analyze the works of some authors in order to see what tools they use to solve these problems. We have also explored the pros and cons of using such tools to solve different problems. We will give some content of these works and what conclusions we have made from them below.

2.1 Computational Intelligence in architectural and interior design: a state-of-the-art and outlook on the field

Emil RACEC in [1] shows problems of design automation which faces particular interior design.

At the article deals with the evolution of building plan models from untrained models that were based on direct enumeration. Because of the exponential growth in complexity their use became impossible for complex projects.

In the next part they evolved to more flexible probabilistic approaches models that might not fully specify layout requirements. Learnable relations have success in dependencies of complex components including their location relative to each other.

And finally it goes to generative models, which solved the problem of creating a variety of models by studying existing models to create new ones, also being able to compose them, preserving the style and preference of the user. But this method works better in the virtual world than in the real one, since in our world there are many restrictions that must be observed.

Further, the author said that interior design and artificial intelligence are being developed in parallel. AI is helpful in automating some processes and facilitates the work of designers using expert knowledge. AI helps in interior design to select furniture depending on the type of a room and functionality. It is also used for arranging furniture, combining styles, in ornamental decoration.

This work leans on the fact that computerization plays a big role in interior design. The author describes how technology evolved along with design and what was used for each purpose. Problems which AI can solve in design and how it can help specialists.

2.2 Classifying in interior design

In the article [2], the author proposes utilizing CNNs to recognize various interior elements and categorize them based on material, size, shape, and style. To train these models, the author created a dataset by scraping Google Images using keywords and by extracting data from www.Houzz.com, which is one of the largest commercial websites for interiors and furniture.

After scraping, the author manually filtered the dataset based on different features. Using this dataset of chairs, the author trained four distinct CNN models to recognize material, capacity, functional features, and design style. Subsequently, these trained CNN models were applied to a dataset containing chairs, and as a result, each chair was assigned characteristics according to these parameters. The mean accuracy of training these models is 94.3%; parameters such as color and shape are recognized quite well, but challenges arise with features like style and material. This is because style is more subjective, and material can appear different in various images. These models can be employed for implementing recommendation systems based on specific requirements.

This work [2] demonstrates that CNN models are effective in recognizing objects and classifying them based on relevant parameters.

In this paper [3], the author asserts that shape recognition is a crucial aspect of interior design due to its inherent complexity. While attributes such as light or color can be specifically defined or calculated, there is no clear measure for form assessment. The author selected 343 images of built house interiors based on their historical context and style. The historical periods were categorized into five time periods, and features such as surface, lines, and points were extracted from 3D models.

Using the Graphical Clustering Toolkit (gCLuTO), the author divided the rooms, which were created in different styles, into 25 clusters based on form similarity. As a result, 16 clusters exhibited various rectangular shapes, five clusters had curved shapes, and four had inclined shapes. The differences between clusters were not solely based on curvature forms; slope angles, location, and the number of features also played a significant role in determining cluster distinctions. This work serves as a foundation for design studies on forms and can be applied to fields such as environmental psychology, environmental behavior studies, neuro-architecture, or any other domain that requires the measurement and evaluation of forms.

This study may be beneficial for measuring and evaluating various forms and grouping them according to different parameters.

In the paper [4], the author says that texture is a fundamental characteristic of objects. Therefore, in this work, using a two-stream neural network that uses the CNN model and handcrafted-based, the author separates images by texture types. The CNN model receives an input RGB image of $300 \times 300 \times 3$ and returns a 2048 feature vector to the output, in turn for handcrafred RGB is converted to black and white format and using GLCM features, LBG features and others is converted into its own feature vector, then these vectors are combined and a texture classifying model is obtained. This work shows that a CNN combined with a handcrafted-based model gives more accurate results than just a CNN.

2.3 Predicting human design decisions with deep recurrent neural network combining static and dynamic data

In this study [5], the author aims to predict human decisions in design by employing RNN and combining both static and dynamic data. Static data consists of designer-related attributes, which remain constant throughout the project. In contrast, dynamic data comprises the sequential actions taken by the designer, as they change at different time stages.

The author tests the RNN on two distinct design tasks: an energy-plus house and a solar parking lot. These tasks differ in complexity and thus effectively evaluate the versatility of this approach.

The author introduces two methods for integrating static and dynamic data. The first method directly combines the data into a one-hot vector, serving as input for the RNN. The second method, based on clustering, utilizes a hidden representation of both static and dynamic data to obtain a surrogate function. This function can enhance the integration process through supervised and unsupervised learning. Experimental results demonstrate that incorporating static data improves the performance of LSTM/GRU models for design prediction.

Overall, the study indicates that utilizing both static and dynamic data yields superior results compared to traditional models that rely solely on the sequence of actions in design without considering human attributes.

2.4 Stochastic Detection of Interior Design Styles Using a Deep-Learning Model for Reference Images

This study [6] aims to categorize reference images of interior styles using a deep learning model. To prepare the data for training, the author scrapes images from various sharing platforms such as ”Ohouse,” ”Ggumim,” and ”Houzz.” The collected data is then preprocessed through labeling, resizing, and augmenting, which includes rotation, shifting, and stretching. These steps increase the number of training samples and enable the model to learn from non-ideal images. The data is subsequently divided into training and validation datasets.

The author utilizes a pre-trained VGG model with a fully connected layer consisting of 256 outputs, a dropout layer, and a classification layer with softmax activation. This architecture predicts the probability of an image belonging to each class. The CNN model ultimately learns to recognize the styles of living room interiors (casual, classic, modern, and natural) with an average probability of 85 percent.

Subsequently, the author develops a recommendation system that can suggest similar interiors or find suitable options based on user requirements. Overall, this study demonstrates that the CNN model is highly effective in classifying and identifying interior designs from images.

Chapter 3

Theory

In this section, we will examine the underlying principles of neural networks, focusing on their architecture and functionality. We will also investigate different types of neural networks and their respective applications. The purpose of this discussion is to enhance the comprehension of the entire work. The theoretical concepts presented in this section are based on the book by Aggarwal [7] and are summarized in our own words.

Neural networks are a type of computer model that mimic the way the human brain processes information. They are made up of multiple layers of interconnected "neurons," which receive input data, process it, and transmit the results to other neurons. Neural networks are used in artificial intelligence and machine learning to tackle various types of problems, including classification, regression, and clustering. They are employed in a wide range of fields, including natural language processing, pattern recognition, robotics, and more. These networks can also be utilized to create artificial agents that can learn from experience and independently solve problems.

There are many different types of neural networks, including feedforward neural networks, convolutional neural networks, and recurrent neural networks. The architecture of a neural network, including the number of layers and the number of neurons in each layer, can be adjusted to suit the needs of a particular problem.

3.1 Basic concepts of neural networks

In this chapter, we will describe the fundamental building blocks of neural networks and how they interact to process and analyze input data. These elements include neurons, weights, biases and activation functions. we will also discuss the overall structure of a neural network, which is designed to filter and process input data.

3.1.1 Neuron

A neuron is the fundamental unit of a neural network, which is a mathematical representation of the human nervous system. It analyzes and manipulates input data, performs calculations based on the data, and generates an output.

The purpose of a neuron is to analyze input data and pass the results on to other neurons in the network. To do this, it multiplies the input data by corresponding weights, adds up the results, and applies an activation function. The activation function determines the strength with which the neuron is ”turned on” and ready to transmit its output to other neurons in the network.

3.1.2 Weights

Weights are factors that are taken into account when neurons in neural networks analyze input data. They determine the significance of each feature in the final output.

The role of weights in neural networks is to consider the importance of different features in the processing of input data by neurons. For instance, if a neural network is used for image recognition, weights can determine the relative importance of different colors and shapes for identifying objects. In general, weights assist the neural network in identifying important characteristics and making more precise classifications.

3.1.3 Activation function

The activation function is a specific function that is used in neural networks to control the intensity with which a neuron is activated. It takes as input the sum of the weighted inputs and produces an output value, which is then transmitted through the network.

There are many different activation functions that can be used in neural networks. Some of the most common activation functions include:

Linear The formula is: f(x) = x.

This function is used when a simple linear model is needed, such as for regression.

Sigmoid The formula is: $f(x) = f(x) = \frac{1}{1+e^{-x}}$ This function is often used in neural networks for binary classification because it produces an output value between 0 and 1, which can be interpreted as a probability.

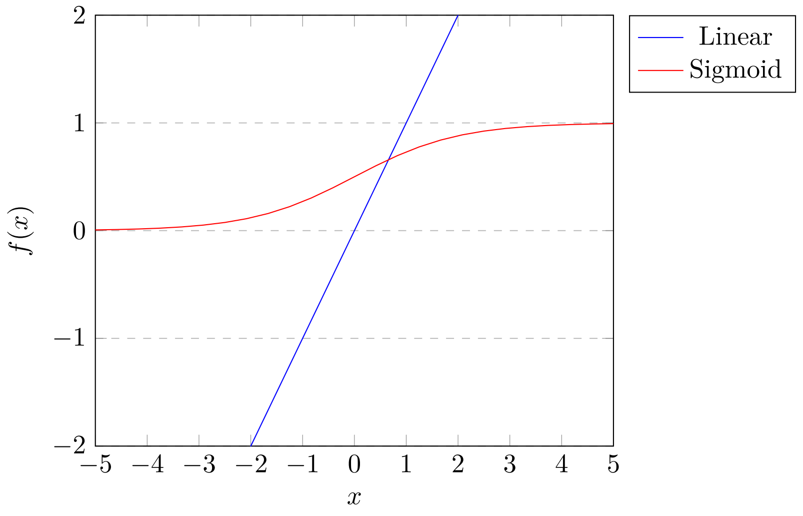
Hyperbolic Tangent (tanh) The formula is: f(x) = tanh(x).

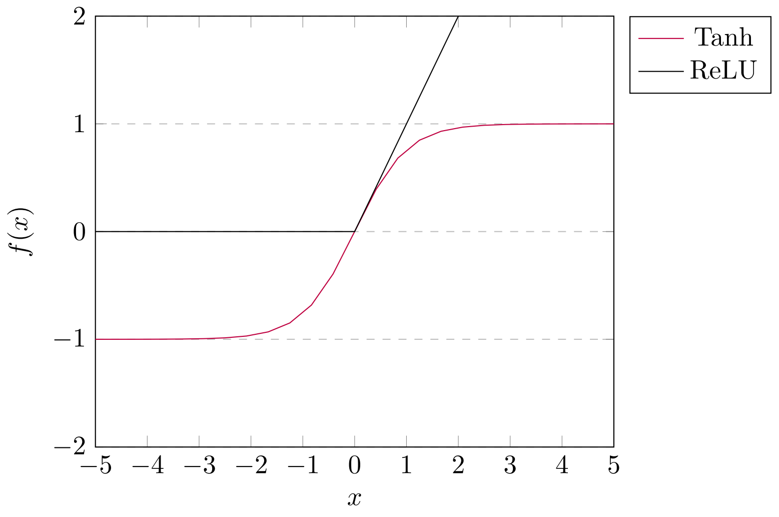
This activation function is often used in neural networks because it smooths the output values, converting them to a range of -1 to 1. This can be useful for tasks such as classification, where it is important to consider negative values. However, it is worth noting that the tanh function is sensitive to outliers, which can lead to degradation of the model quality.

ReLU (Rectified Linear Unit) The formula is: f(x) = max(0, x).

This function is often used in neural networks for image and audio processing. It works so that all negative values are replaced by 0, and all positive values remain unchanged. This can lead to faster learning speed and better accuracy of the model.

Below are graphs of these functions:





3.1.4 Bias

A bias is a value that is added to the weighted sum of a neuron's inputs before the activation function is applied. The role of biases in a neural network is to help the network learn more complex patterns and relationships in the data, and to help prevent the network from becoming stuck in a local minimum during the training process. To use biases in a neural network, they are typically initialized to a small, non-zero value and updated during the training process along with the weights. The values of the biases are typically updated using techniques such as gradient descent and backpropagation, which adjust the biases to optimize the network's performance on a given task.

3.2 Types of neural networks

3.2.1 Perceptron

The simplest NN is perceptron, it has only one layer of input nodes and one output node. Input layer contains d nodes with features X = { x_1, x_2 ... x_d } and edges to output node with weight

W = { w_1, w_2 ... w_d }. Using these values, the linear function computes the output node.

$$\hat{y} = sign\{\sum_{i=1}^{d} w_i * x_i\}$$

After that, we can predict output value $\hat{y}$ using the sign function that maps our value of output node to -1 or +1. The error of prediction we can count using E(X) = y - $\hat{y}$, where y is a real value, $\hat{y}$ is a predicted value. If the error is not 0, we need to update the weight of NN. Instead of a sign function we can use other activation functions for different types of goals, like a support vector machine, least-squares regression, regression classifier.

3.2.2 Feedforward neural network

A feedforward network is a type of ANN that does not have any feedback or loop connection, so the information flows only in one direction (from input layer to output).

The basic structure of a feedforward neural network can be represented by the following formula:

$y = f(\sum_{i=1}^{n} w_i x_i + b)$

where:

y is the output of the network $f(x)$ is the activation function w_i is the weight of the ith input x_i is the ith input b is the bias The activation function is used to introduce non-linearity to the network. Some common activation functions include sigmoid, tanh, and ReLU.

Feedforward neural networks are used in a variety of applications, including image and speech recognition, natural language processing, and even playing video games. They are trained using an optimization algorithm, such as stochastic gradient descent, to minimize the error between the predicted output and the desired output.

3.2.3 Perceptron

The simplest NN is perceptron, it has only one layer of input nodes and one output node. Input layer contains d nodes with features X = $\{x_1, x_2$ ... $x_d\}$ and edges to output node with weight W = $\{w_1, w_2 \ldots w_d\}$. Using these values, the linear function computes the output node.

$$\hat{y} = sign\{\sum_{i=1}^{d} w_i * x_i\}$$

So at the 3.1 you can see scheme of perceptron:

After that, we can predict output value $\hat{y}$ using the sign function that maps our value of output node to -1 or +1. The error of prediction we can count using E(X) = y - $\hat{y}$, where y is a real value, $\hat{y}$ is a predicted value. If the error is not 0, we need to update the weight of NN. Instead of a sign function we can use other activation functions for different types of goals, like a support vector machine, least-squares regression, regression classifier.

The goal of the perceptron algorithm is to minimize the prediction error, so we can write the goal of the algorithm in least squares form that also names as a loss function:

$$Minimize_{\bar{W}} L = \sum_{(\bar{X},y) \in D} (y - \hat{y})^2$$

The training algorithm of ANN gives a random element of input and after each iteration updates the weight vector based on the error E(X) as follows:

$$W = W + \alpha * E(X) * X$$

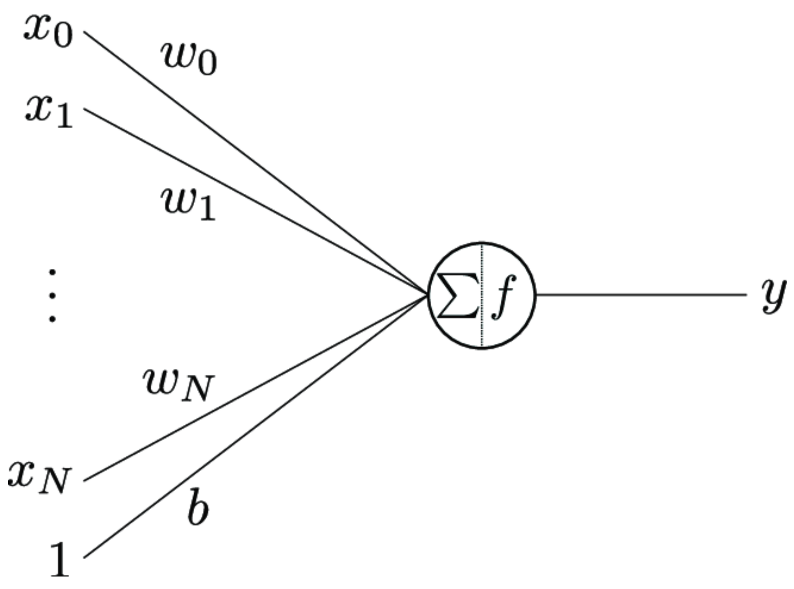


Figure 3.1 CNN model [8]

Where the parameter α is a learning rate of the NN. Training algorithm reiterates over all the training data and balances the weight until our prediction is really close to the actual value, this process called feeding.

Sometimes, the output of a prediction model includes a constant value that cannot be explained by the input features. This constant value is known as the bias. Bias is usually used for balance distribution, this is especially important when the two binary classes are not evenly distributed. The bias variable b helps to account for this constant value and improve the accuracy of the prediction:

$$\hat{y} = sign\{\sum_{i=1}^{d} w_i * x_i + b\}$$

3.2.4 Recurrent neural networks

Recurrent neural networks (RNNs) are a type of artificial neural network that are specifically designed to process sequential data. They are particularly useful for tasks such as language translation, speech recognition, and time series forecasting.

RNNs have a unique architecture that includes feedback connections, which allow the network to retain information from previous time steps and use it to inform the current prediction. This is accomplished through the use of hidden states, which are updated at each time step and passed on to the next time step.

The 3.1 shows a scheme of RNN:

RNN consists of:

- Input layer: The input layer receives the input data, x, at each time step t.
- Hidden layer: The hidden layer computes the hidden state h_t at each time step t, using the previous hidden state h_{t-1}, the current input x_t, and the weights and biases for the hidden layer. The computation can be represented by the following formula:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

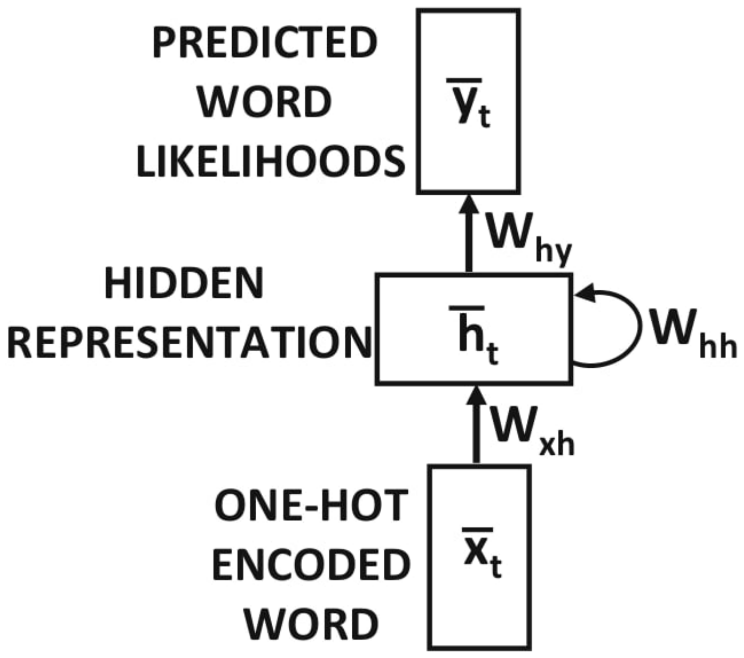


Figure 3.2 CNN model [**rnn-image**]

- Output layer: The output layer computes the output y_t at each time step t, using the current hidden state h_t and the weights and biases for the output layer. The computation can be represented by the following formula:

$$y_t = g(W_y h h_t + b_y)$$

- Weights and biases:
  - $W_h x$: Weights matrix connecting the input layer to the hidden layer
  - $W_h h$: Weights matrix connecting the previous hidden state to the current hidden state
  - b_h: Bias term for the hidden layer
  - $W_y h$: Weights matrix connecting the hidden layer to the output layer
  - b_y: Bias term for the output layer
- Activation functions:
  - f: Activation function for the hidden layer
  - g: Activation function for the output layer

A recurrent neural network (RNNs) is universal because it can compute any function that can be computed by a Turing machine, meaning that it can perform any computation that is possible by a computer. The output of the RNN can be obtained after a number of time steps that is proportional to the amount of time needed by the Turing machine to complete the computation and the length of the input. These results apply to precise implementations of functions, not approximations, and the input to the RNN must be a binary sequence, with the output being discretized as a binary output.

3.2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of artificial neural network that are specifically designed to process data with a grid-like topology, such as images. They are commonly used in tasks that involve image and video analysis, such as image classification, object detection, and video action recognition.

To use a CNN, a large dataset of labeled examples is required for training. The CNN is then trained using an optimization algorithm, such as stochastic gradient descent. Once trained, the CNN can make accurate predictions on new, unseen data.

A Convolutional Neural Network (CNN) is composed of multiple layers, including convolutional layers, pooling layers, and fully-connected layers. Each layer has a specific role in the model and contributes to the model's ability to extract and abstract features from the input data. The architecture of CNN is shown on 3.3:

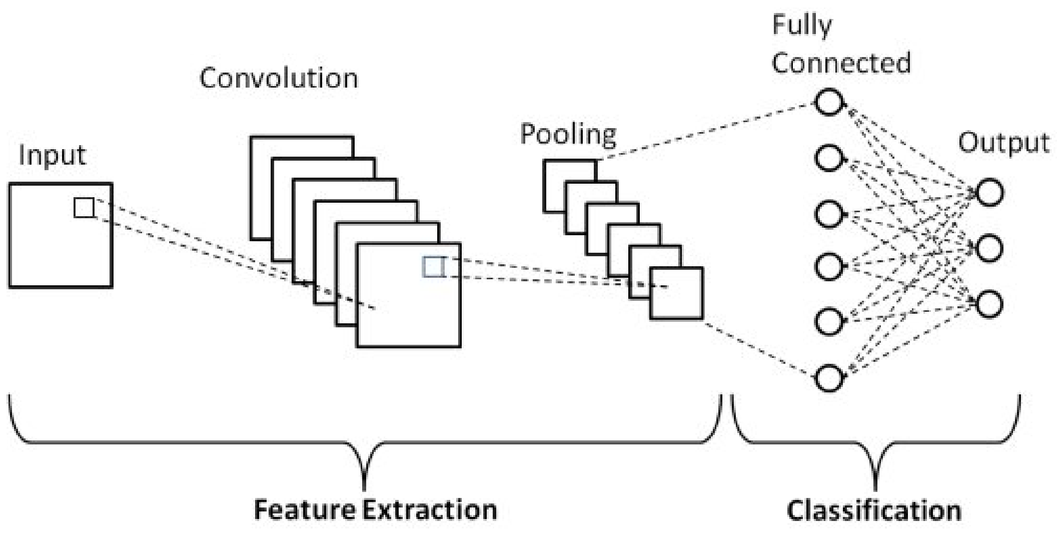


Figure 3.3 CNN model [9]

The input layer receives the input data, typically in the form of an image or a sequence of images. The input data is passed through the convolutional layers, which apply a convolution operation to extract local features. The convolution operation can be represented by the following formula:

$$f(x, y) = (x * y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x(i)y(j)$$

where x and y are the input functions and $*$ is the convolution operator.

The convolutional layers are followed by pooling layers, which reduce the spatial dimensions of the feature maps by applying a pooling operation, such as max pooling or average pooling. The pooling operation can be represented by the following formula for max pooling:

$$f(x) = \max_{i=1}^{n} x_i$$

where x is the input data and n is the size of the pooling window.

The final layer is the fully-connected layer, which performs the final classification or regression tasks. The fully-connected layer takes the output of the pooling layer as input and applies a linear operation, followed by a non-linear activation function, such as the rectified linear unit (ReLU). The linear operation can be represented by the following formula:

$$f(x) = Wx + b$$

where W is the weight matrix and b is the bias term.

In summary, the structure of a CNN consists of an input layer, followed by multiple convolutional and pooling layers, and ending with a fully-connected layer. Each layer is connected to the next through a linear operation and possibly a non-linear activation function, and the weights and biases of the layers are learned through training.

3.2.6 Graph neural network

The theoretical concepts presented in this section are based on the paper by Thomas N. Kipf and Max Welling [10] and are summarized in our own words.

Graph Convolutional Networks (GCNs) are a type of neural network specifically designed to operate on graph-structured data. In GCNs, the input data is represented as a set of nodes and edges that connect these nodes. GCNs have been used extensively in computer vision, bioinformatics, recommendation systems, and other fields that deal with graph-structured data.

In a GCN, each node in the graph represents a vector of features that capture information about that node. The goal of the GCN is to learn a function that maps each node to a low-dimensional vector representation that captures both the node's features and its neighborhood structure. This is achieved by using a localized linear filter that aggregates information from a node's neighbors, defined as:

$$H_i^{(l+1)} = \sigma(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} H_j^{(l)} W^{(l)})$$

where $H_i^{(l)}$ is the representation of node i at layer l, $\mathcal{N}(i)$ is the set of neighbors of node i, $W^{(l)}$ is the learnable weight matrix for layer l, and c_{ij} is a normalization constant that depends on the degree of nodes i and j. The activation function σ is typically a non-linear function such as ReLU.

The above equation can be written in matrix form as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

where $H^{(l)}$ is the matrix of node representations at layer l, $\tilde{A} = A + I$ is the adjacency matrix of the graph with added self-loops, $\tilde{D}$ is the diagonal matrix of node degrees of $\tilde{A}$, and I is the identity matrix. The normalization of $\tilde{A}$ by $\tilde{D}^{-\frac{1}{2}}$ and $\tilde{D}^{-\frac{1}{2}}$ ensures that the filter is localized and takes into account the degree of the nodes.

Overall, GCNs are a powerful tool for working with graph-structured data. They are able to capture both the features and the structure of the graph, making them suitable for a variety of tasks such as node classification, link prediction, and graph classification.

Chapter 4

Algorithm

In this chapter, we discuss the structure of the algorithm implemented in this work. We discuss the detector employed for furniture recognition, the neural network utilized for feature extraction from cropped images, and the process of constructing a graph neural network based on the generated data to predict compatibility among interior design elements.

In this study, we present a method that predicts the compatibility between interior design objects and subsequently generates various combinations based on this compatibility. The algorithm operates on features extracted from furniture items. To implement this algorithm, several steps are required: create a detector that can recognize and crop images of furniture, extract features from the cropped furniture images, and develop a graph neural network (GNN) to predict compatibility between accessories based on the extracted features. Below we will elaborate on each of these steps.

Initially, we trained a YOLOv7 detector to recognize furniture within various interior settings with the goal of generating a dataset consisting of cropped images. These cropped images were then utilized for feature extraction, ultimately creating a dataset suitable for training a graph convolution neural network (GCN).

Subsequently, we aimed to develop a neural network for extracting features from furniture items, which would then be used to create a graph for predicting compatibility. The challenge in combining images of patterns and accessories based on features is the potential for differing features that do not always intersect. As a solution, we propose focusing on a set of parameters common to all interior elements: color, material, style, and room type. We believe these features are sufficient for effectively predicting compatibility.

Initially, our intention was to create a single neural network for predicting all these features. However, we encountered several challenges, the primary one being the labeling of data from different manufacturers. Consequently, we were unable to create a dataset with labels containing all of these features. We decided to train four separate neural networks to predict each feature independently.

For feature extraction, we employed the VGG architecture, which consists of several layers of convolutional neural networks. We created four datasets for each feature by dividing images into classes and preprocessing the data through resizing and normalization. These datasets were then fed into pretrained VGG networks, resulting in four VGG neural networks that extract features that can be used for creating combinations.

Finally, we assembled all the necessary tools to train a graph neural network (GNN) that would predict compatibility between elements of interior design. Utilizing YOLOv7, we cropped the images, and with the VGG neural networks, we extracted features from the cropped furniture images obtained from interior scenes. These features were then used to create a graph.

For objects of interior design that belong to the same room type and share a similar style, we established an edge between them. If they also have the same color or material, the weight of the edge is increased. To each node, we added the image that was transformed into a vector along with its corresponding features.

Subsequently, we constructed a graph convolution network, with the input being the built graph and the output being a compatibility matrix. Based on this output, we are able to generate combinations of patterns and accessories for interior design.

Chapter 5
Datasets

In this chapter, we elucidate the process of constructing datasets for all neural networks implemented in this study. We discuss the challenges faced during dataset creation, the methods employed for data labeling, and the principles guiding class selection. Additionally, we provide insights into the pre-existing datasets utilized in this work and explain their respective applications.

5.1 Dataset for VGG

In this work, we collect a dataset of images of patterns and accessories for use in training our models. To obtain a diverse and representative sample, we source our data from two sources: www.Houzz.com, a website that offers a large collection of interiors and interior attributes, and Google image search, a widely used image search engine.

In our initial approach, we aimed to collect images of patterns and accessories with labels that corresponded to the parameters we desired to use, such as style, color, and material. To obtain a large and diverse dataset, we used www.Houzz.com as a source and developed a scraper to collect the images and their associated labels.

However, we encountered a significant challenge in the annotation process, as different manufacturers used different formats and terminology to describe the attributes of their products. This resulted in a highly fragmented dataset, with a large number of variations in the naming and labeling of materials, for example. As a result, we were unable to use the labels as they were.

To illustrate this issue, we present an example of patterns for covering walls and the materials from which they are made. We collected a total of 5490 images and divided them by the name of the material, resulting in 173 folders with the names of different materials (Figure 5.1 shows a portion of this distribution). Despite the large size of the dataset, the variations in the labeling made it difficult to use the labels directly for training our models.

Despite our attempts to preprocess and clean the collected dataset of images of patterns and accessories in order to make it suitable for training our models, we were unable to find a satisfactory solution to address the inconsistency in the labeling of the attributes by different manufacturers. As a result, we determined that the dataset was unreliable for training and decided to use it only for generating combinations and testing our model.

We also investigated other sources for labeled data, but encountered similar challenges with inconsistency in the labeling. Therefore, we concluded that it was infeasible to collect a dataset in which each image contained all the desired characteristics, and we had to modify our approach accordingly. Rather than attempting to predict all the desired parameters at once, we decided to build separate models for each parameter and generate the labels ourselves.

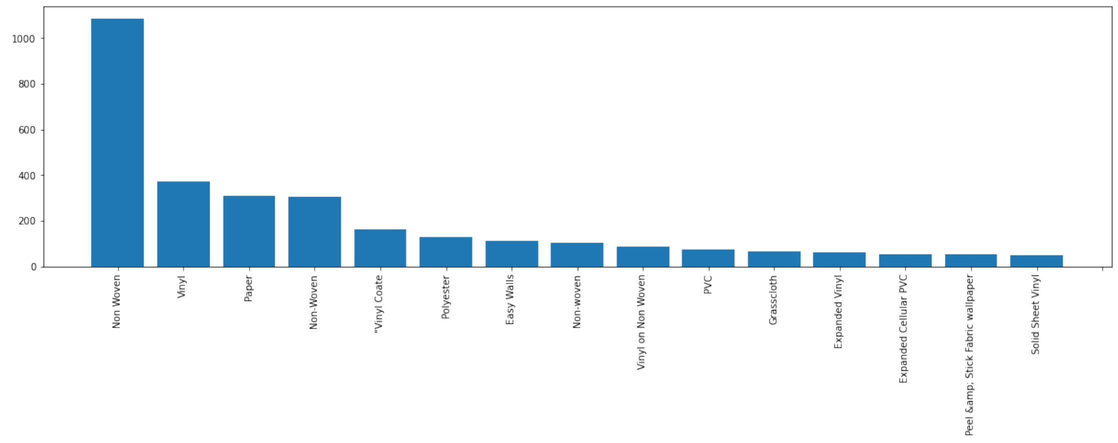


Figure 5.1 Distribution of materials from houzz.com

To this end, we used Google image search as a source for the images and manually labeled them according to the parameters of interest, such as style, color, and material. This approach had its own set of challenges, as Google image search does not always return images that match the query, leading to a need for more thorough cleaning and preprocessing of the data.

5.1.1 Materials

One of the parameters we consider is the material of the objects, as it is an important factor that influences the appearance and properties of the interior elements.

To collect data for this parameter, we selected seven main categories of materials from which interior objects are commonly made:

- ceramic
- glass
- paper
- plastic
- stone
- textile
- wood

We used Google image search to collect a large dataset of images that represent these materials, both in the form of simple images of the materials themselves and images of interior objects made from these materials. In total, we obtained 20215 images, which we divided into the respective categories based on their material.

Figure 5.2 shows the distribution of the images by material group. As can be seen, the dataset is balanced and covers a wide range of materials, enabling us to train models that are capable of extracting meaningful features from a diverse set of images.

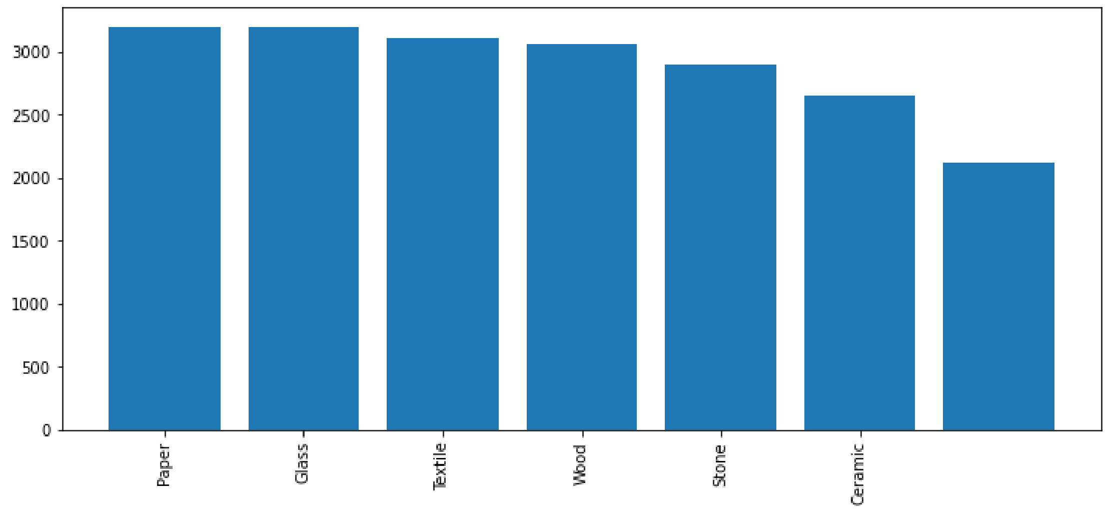


Figure 5.2 Distribution of materials

5.1.2 Styles

In the next phase of our project, we aim to train a model to recognize the style of interior attributes as an important parameter for combining images of patterns and accessories. Style is a significant factor that influences the aesthetic and visual appeal of the interior, and it is essential to consider it when selecting and combining elements.

To train our model, we collected a dataset of images of interior designs from Google image search, focusing on six of the most common interior styles:

- bohemian
- contemporary
- farmhouse
- industrial
- modern
- traditional

We selected a diverse set of images that represent these styles, both in terms of the overall design and the specific elements that define the style.

As a result, we obtained a dataset of 8910 images in different styles (the distribution of images is shown at Figure 5.3), which we used to train our model. By learning from this dataset, our model is able to extract features that are relevant to the style of the interior attributes and use them to make accurate predictions.

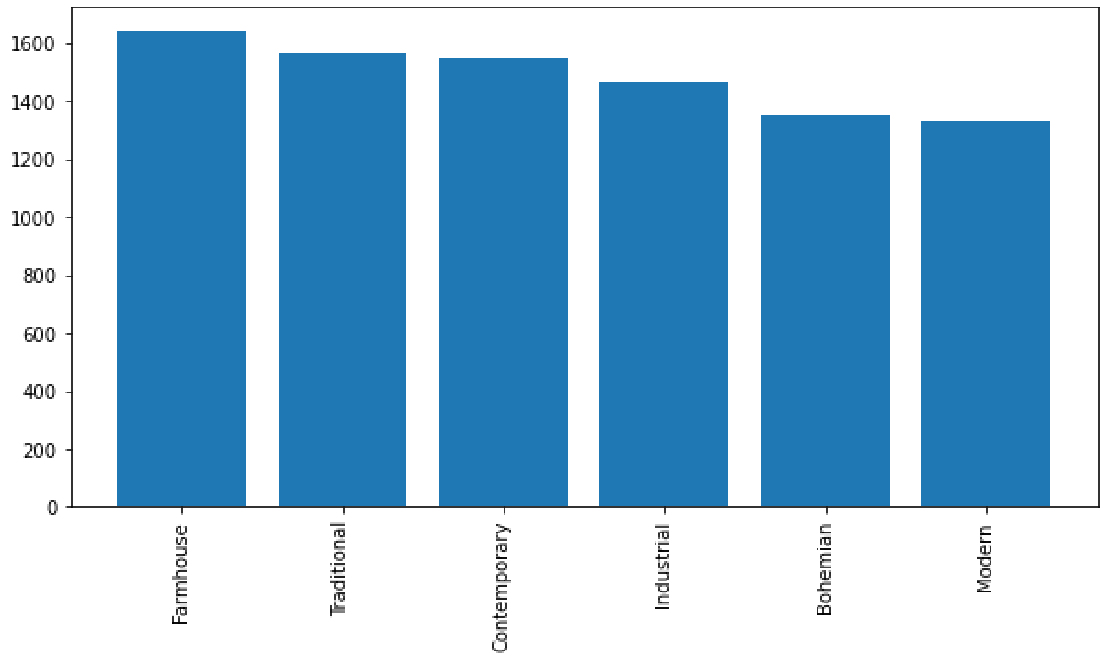


Figure 5.3 Distribution of styles

5.1.3 Colors

Also we scrapped 13,689 color images were scraped to aid in the training of a model for extracting colors from any given object. The main colors that were chosen for this task are listed below in a bulleted list:

- green
- orange
- red
- grey
- blue
- white
- yellow
- brown
- violet
- black

These colors were selected as they represent a wide range of hues and shades that are commonly used in interior design, and the scraped images can be used to train a model to recognize and categorize colors in the design process. The distribution of colors is shown at Figure 5.4).

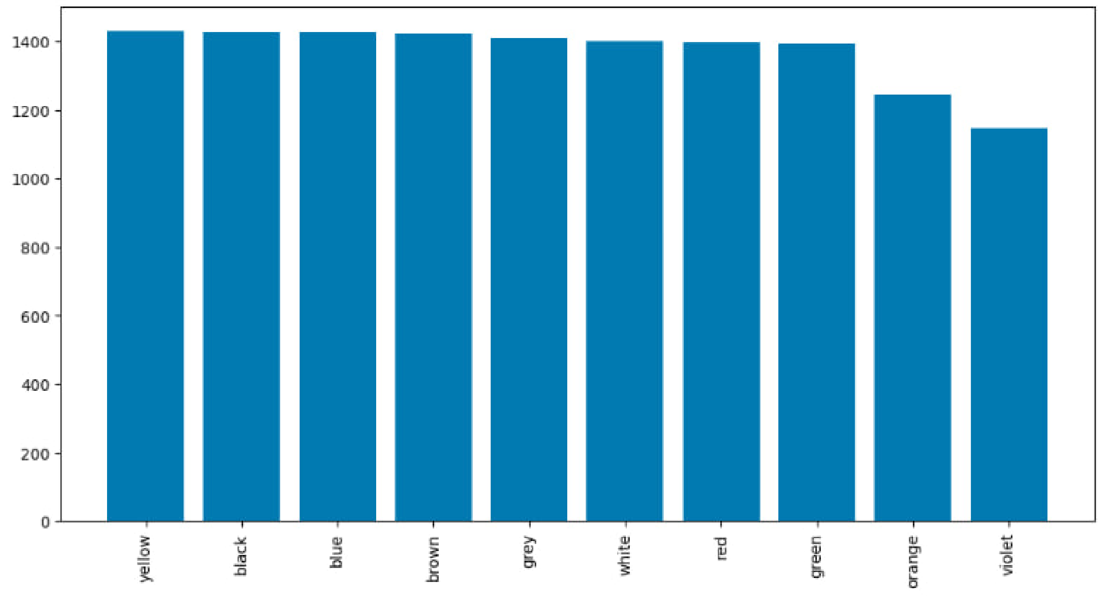


Figure 5.4 Distribution of colors

5.1.4 Type room

In addition to extracting style, material and color, we also sought to identify the type of room depicted in the image. For this task, we used the ADE20k dataset and selected three room categories:

- living room
- child's room
- bedroom

The dataset provided object coordinates for each image, allowing us to crop the relevant portion of the image for analysis. With this dataset, we were able to train a model to predict the type of room depicted in an image. The final dataset contained 16,744 cropped images labeled with their corresponding room type. The distribution of colors is shown at 5.5)

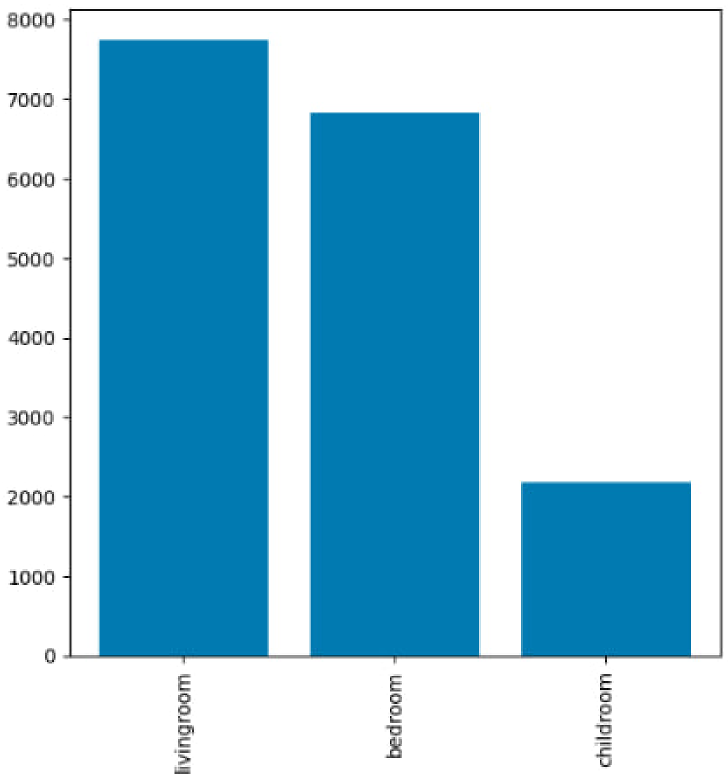


Figure 5.5 Distribution of colors

5.2 Dataset for GCN

In developing a combination of accessories, we decided to implement a graph convolutional neural network (GCN). To achieve this, it was necessary to create a dataset that could be represented as a graph. Consequently, we utilized the ADE20k dataset and performed object detection on interior elements. Following this, we cropped the images and generated features for each object. The resulting dataset was stored in a CSV file containing the image names, room types, attributes, styles, colors, and materials. The structure of the dataset is illustrated below:

0	ADE_train_00000386_15.jpg	bedroom	lamp	Industrial	red	ceramic
1	ADE_train_00000386_8.jpg	bedroom	wall	Modern	brown	paper
2	ADE_train_00003821_5.jpg	bedroom	bed	Industrial	orange	textile
3	ADE_train_00004132_11.jpg	bedroom	bed	Bohemian	yellow	textile
4	ADE_train_00004132_17.jpg	bedroom	wall	Bohemian	orange	ceramic

Each row in the CSV file represents an object in the scene, along with its associated metadata. The dataset was designed to facilitate the training of a GCN model that can effectively predict the compatibility between different interior design objects.

The ADE20k [11] dataset is advantageous for our purposes because it allows us to determine which accessories belong to a particular interior, enabling the creation of graphs based not only on features but also on the relationships between elements within the same interior. However, the dataset has some limitations concerning image quality, as some objects are very small, and cropped images may contain parts of other objects, which negatively impacts the efficiency of feature extraction.

To address these issues, we decided to create another dataset based on the Bonn Furniture Styles Dataset [12], which comprises images of interior objects. We also predicted features for each image and generated a dataset suitable for training and testing a GCN model. Although this dataset does not provide explicit information about the relationships between objects within the same interior, it still allows for the construction of graphs based on features.

Chapter 6

Used architectures and libraries

In this chapter, we discuss the architectures and libraries utilized in this study, explaining their functionality and purposes. We will elucidate the underlying principles of these tools and elaborate on the reasons for their selection in order to accomplish the objectives of this work.

In this study, we employ the VGG architecture to train models that extract features from images of interior objects, including the style, material, and type of room. Additionally, we utilize the YOLO detector to recognize furniture present in the images. The present work outlines the technical underpinnings of these two methods and elucidates the rationale for our selection of these techniques, with the aim of providing a more comprehensive understanding of the subsequent steps in our research.

6.1 YOLOv7

YOLO [13] is a real-time object detection system that uses a deep neural network to detect objects in images. The basic idea behind YOLO is to divide the input image into a grid of cells and predict the class probabilities and bounding boxes for each cell. The class probabilities represent the probability of an object belonging to a particular class, and the bounding boxes represent the coordinates of the box that tightly encloses the object.

YOLOv7 uses a backbone network, typically a pre-trained convolutional neural network (CNN), to extract features from the input image. The features are then passed through a series of convolutional layers to reduce the spatial dimensions of the features while increasing the depth of the feature map. This process allows the network to detect increasingly complex patterns and features within the image.

The output of the convolutional layers is then fed into a set of detection heads, which are responsible for predicting the bounding boxes and class probabilities for the objects in each cell of the grid. YOLOv7 uses a single detection head to predict all objects in the image, which allows it to detect multiple objects in real-time. Each prediction consists of a set of bounding boxes and the class probabilities for those boxes. The boxes are parameterized relative to the grid cell they are in, so the network predicts the center coordinates, width, and height of each box as offsets from the coordinates of the cell.

To train the YOLOv7 network, a loss function is used to measure the error between the predicted object detections and the ground-truth object annotations. The most commonly used loss function for object detection is the mean average precision (mAP) loss, which is a combination of classification accuracy and localization accuracy.

6.2 VGG

VGG, or the Visual Geometry Group network [14] is a deep convolutional neural network (CNN) architecture that comprises multiple layers. In this study, we utilize the VGG-16 architecture, which is composed of 16 convolutional layers. VGG has demonstrated excellent performance in recognition and classification tasks. Specifically, the VGG-16 model achieves 92.7% test accuracy on ImageNet, which consists of 10 million images distributed among 1000 classes.

The VGG architecture is characterized by convolution layers with rectified linear unit (ReLU) activation, max pooling layers, fully connected layers, and softmax normalization, as depicted in Image 6.1.

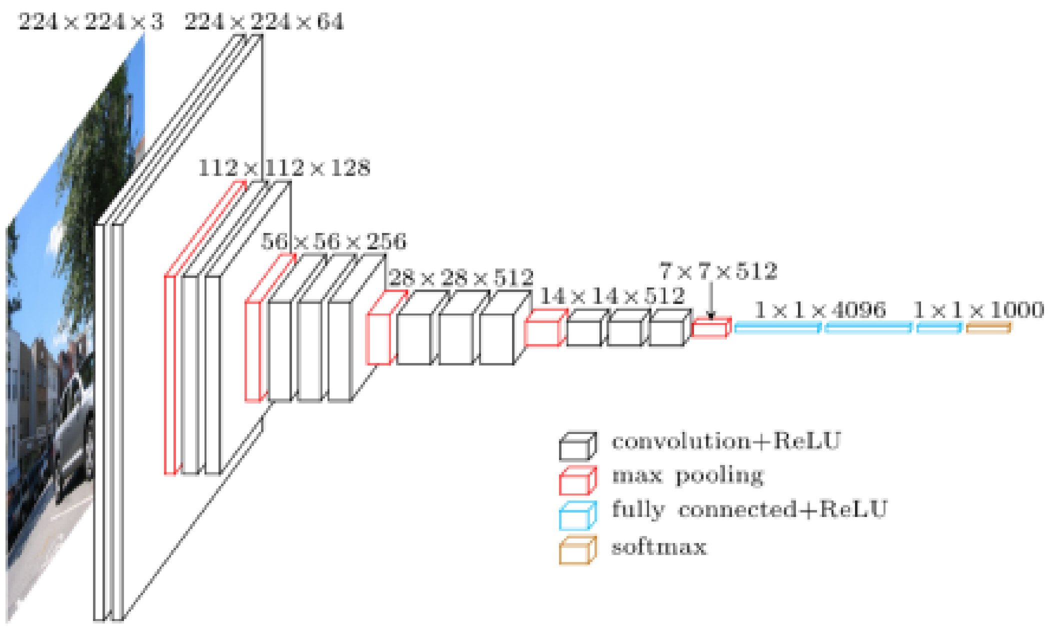


Figure 6.1 VGG architecture [14]

We have decided to use the VGG architecture to extract features because it has shown to deliver superior performance in image classification tasks, as compared to simpler CNN models or analogs such as AlexNet. One of the reason for this is the use of multiple 3x3 kernel-sized filters in instead of larger kernel-sized filters, which provides better feature learning capability. Furthermore, pre-trained VGG neural networks can be utilized to improve prediction results with a smaller number of images.

6.3 Libraries

- TensorFlow: TensorFlow is an open-source software library for machine learning and artificial intelligence that provides a flexible and efficient platform for training, testing, and deploying models. We have used TensorFlow to develop and train our models, including the VGG model that we have used in our bachelor work. TensorFlow has enabled us to define complex models and optimize their performance using advanced techniques such as gradient descent and backpropagation.
- SciPy: SciPy is a library for scientific computing in Python that provides a wide range of numerical and statistical algorithms, as well as tools for linear algebra, optimization,

and signal processing. We have used SciPy to perform various mathematical and statistical operations on the data, such as fitting models, computing distances and correlations, and generating plots.

- os: The os library is a built-in Python library that provides functions for interacting with the operating system, such as reading and writing files, creating and deleting directories, and executing external commands. We have used os to access and manipulate the files and directories in our project, as well as to communicate with external programs and resources.
- tqdm: tqdm is a library for displaying progress bars and status updates during long-running tasks. We have used tqdm to provide feedback and updates on the progress of our experiments and data processing tasks, as well as to monitor the performance of our models.
- util: The util library is a custom library that we have developed for our project. It contains various utility functions and classes that we have found useful for our tasks, such as data loading, preprocessing, and visualization. We have used the util library to streamline the development and testing of our models and to simplify the codebase.
- matplotlib: matplotlib is a library for plotting and visualizing data in Python. We have used matplotlib to generate various charts, plots, and figures to illustrate the results of our experiments and to communicate our findings to the reader.
- requests: requests is a library for making HTTP requests in Python. It simplifies the process of sending and receiving HTTP messages, and provides support for various authentication and encoding schemes. We have used requests to communicate with web servers and APIs, and to fetch data from the web.
- selenium: selenium is a library for automating web browsers. It allows us to write scripts that simulate user interactions with a web page, such as clicking links, filling forms, and scrolling. We have used selenium to scrape data from websites and to perform functional testing of our web applications.
- urllib: urllib is a Python library for working with URLs. It provides functions for parsing and building URLs, as well as for sending HTTP requests and receiving responses. We have used urllib to access web resources, as well as to encode and decode data for transmission over the web.

Chapter 7

Implementation

This chapter outlines the software implementation for predicting compatibility between objects in interior design. The implementation comprises several key components, including a data scraper for collecting images from web resources, the training of the YOLOv7 detector for furniture recognition, the training of VGG neural networks for feature extraction, and the training of a Graph Convolutional Neural Network (GCN) for predicting object compatibility.

7.1 Scraper

For data collection, we developed a web scraper that extracts images from Google search results. The input for this scraper consists of a list of links and a corresponding list of directories in which to save the images from those links. The scraper accesses each link and saves any images it finds, scrolling down the page to collect all available images. The resulting data can be organized into various categories as desired by the user and be used for training purpose.

7.2 VGG model training

To extract features related to attributes such as style, color, and material, we employed the VGG neural network. The training process was composed of a loader, model, and worker. In the loader, the dataset was loaded, prepared for training, and split into folders for training and validation sets. In the model, we loaded and configured the parameters of VGG16, and worked with its layers. The worker was responsible for performing the training and validation process, as well as calculating the accuracy. The following sections will provide more detailed explanations of each component of the training process.

7.2.1 Loader

We use transforms from the PyTorch torchvision library to prepare the images for training. Specifically, we use the Compose function to chain together a series of transformations that are applied to each image before it is fed into the VGG neural network. The transformations that we use are as follows:

1. RandomResizedCrop: We first resize each image to a standard size of 224x224 pixels, which is the format required for the VGG neural network. We also use the RandomResizedCrop transformation to randomly crop the image to a slightly smaller size, with a random scaling

factor applied. This is done to ensure that the network is exposed to a variety of image sizes and scales during training, which can improve its ability to generalize to new images.

2. RandomHorizontalFlip: We then apply a random horizontal flip to the image with a 50% probability. This transformation is used to create mirrored versions of the original image, which effectively doubles the size of the training set and can improve the model's ability to recognize objects at different orientations.

3. ToTensor: The transformed image is then converted to a PyTorch tensor, which is a multi-dimensional array that can be processed by the neural network.

4. Normalize: Finally, we normalize the pixel values of the image using the mean and standard deviation values of the entire dataset. This is done to bring the pixel values of the images to a similar range, which can help to stabilize and speed up the training process. Specifically, we subtract the mean value of the dataset from each pixel, and then divide the result by the standard deviation of the dataset.

Overall, this series of transformations allows us to preprocess the input images in a way that can help to improve the performance and stability of the VGG neural network during training.

Next, the dataset was randomly split into training and validation sets, with 80% of the data used for training and 20% used for validation. This was achieved by generating a random split of indices for the dataset and then using these indices to create two samplers, one for training and one for validation. The samplers were used to create two DataLoader objects, which were used to facilitate the training process.

7.2.2 Models

This is a module containing the implementation of the vgg16 model, as well as some helper functions for defining its layers and initializing its weights.

The vgg16 model is a convolutional neural network architecture consisting of 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. It takes an input image and produces a classification output. The vgg16 class inherits from the nn.Module class in the PyTorch library and implements the forward pass of the model in its forward method.

The cnn_layers function is a helper function that creates the convolutional layers for the vgg16 model. It takes the number of input channels and a boolean flag indicating whether to use batch normalization and returns a sequential container of the convolutional layers. The function defines the configuration of the convolutional layers as a list of integers and the string ”M” (indicating a max-pooling layer) and iterates through this configuration to create the convolutional layers. If the batch normalization flag is set to True, batch normalization and ReLU activation are added to each convolutional layer.

The fc_layers function is a helper function that creates the fully connected layers for the vgg16 model. It takes the number of classes and returns a sequential container of the fully connected layers. The function defines a fixed architecture for the fully connected layers and returns it in a sequential container.

The vgg16 class also contains methods for initializing the weights of its layers (init_weights), loading pretrained weights (load_weights), and freezing layers for transfer learning (freeze_cnn_layers). The memory_usage method is a utility function that prints the memory usage of the model.

Overall, models.py provides an implementation of the vgg16 model with helper functions for defining its layers and initializing its weights, making it a useful module for training and evaluating image classification models.

7.2.3 Worker

The ”worker” component of the codebase is responsible for managing the training process of a machine learning model. When the training process is initiated, the worker first loads the model that is to be trained. The model may have been previously saved at some checkpoint during a previous training session, and the worker provides the ability to load this checkpoint and continue training from that point.

During the training process, the worker also performs validation to assess the performance of the model. This involves passing a set of validation data through the model and computing various performance metrics, such as accuracy and the confusion matrix. These metrics provide valuable insights into how well the model is generalizing to new data and can guide decisions on how to adjust the training process.

Overall, the worker component plays a critical role in the training process of a machine learning model. By managing the loading of the model and providing tools for checkpoint and validation, the worker enables more efficient and effective training, leading to better performance of the final model. The use of scientific methods and rigorous evaluation techniques ensures that the trained model is reliable and useful for its intended purposes.

7.3 Detecting patterns and furniture

To extract relevant features from various patterns and attributes in the field of interior design, it is necessary to first detect and crop the relevant regions of interest. This can be accomplished through the use of an object detection algorithm, such as YOLOv7, which is specifically designed to locate and identify objects within an image.

To train the YOLOv7 algorithm to detect the relevant attributes and patterns in the field of interior design, we utilized the ADE20k dataset. This dataset contains a large number of images annotated with object labels, making it an ideal choice for training object detection models. By training the YOLOv7 algorithm on this dataset, we were able to teach the algorithm to accurately identify and locate the relevant attributes and patterns within the images.

Once the YOLOv7 algorithm has been trained, it can be used to detect and crop the relevant regions of interest in new images. By generating a combination of these cropped regions, we can extract the relevant features from the images and use them to inform further analysis or design decisions in the field of interior design.

7.3.1 Training YOLOv7

For training YOLOv7 on the ADE20k dataset, we needed to create labels for each image in YOLO format. Each label consists of the class ID, center coordinates (x,y), and the width and height of the bounding box. The ADE20k dataset provides a segmentation of polygons, which is represented as a list of x and y coordinates. To obtain the bounding box of each object, we obtained the minimum x and y values and maximum x and y values of the polygon. we then translated these values into the YOLO format by computing the center coordinates, width, and height of the bounding box using the following formulas:

$$
\begin{aligned}
center_x &= \frac{min\_x + max\_x}{2 \times image\_width} \\
center_y &= \frac{min\_y + max\_y}{2 \times image\_height} \\
width &= \frac{max\_x - min\_x}{image\_width} \\
height &= \frac{max\_y - min\_y}{image\_height}
\end{aligned}
$$

These formulas calculate the center coordinates and size of a bounding box (in terms of the object's width and height). By dividing the center coordinates and size by the dimensions of the image, we are normalizing the values to be in the range [0, 1], which is necessary for the YOLO algorithm to work properly. By computing these values for each object, we was able to generate YOLO-formatted labels for the ADE20k dataset, which could be used to train the YOLOv7 algorithm to detect and localize the relevant attributes of interior design in images.

ADE20k dataset has a variety of scenes, but for this project, only images from the Living Room, Bedroom, and Child Room classes were used. There are over 100 attributes present in each of these classes, but for the purposes of this project, 14 attributes were selected, as they are the most relevant for interior design and appear in nearly every room. These attributes have the largest number of instances in the dataset, and include: *wall*, *bed*, *table*, *clock*, *chair*, *sofa, couch, lounge*, *lamp*, *plant, flora, plant life*, *vase*, *pot, flowerpot*, *rug, carpet, carpeting*, *blanket, cover*, *coffee table, cocktail table*, and *ottoman, pouf, pouffe, puff, hassock.*

Using the labels and images from the ADE20k dataset, we trained the YOLOv7 model and was able to extract attributes. These attributes can be used in the future for creating a combination of patterns and attributes.

7.4 GCN

The primary goal of this work is to generate combinations of interior objects by developing a Graph Convolutional Neural Network (GCN) to predict compatibility between images of furniture. To accomplish this, we utilize a series of previously developed networks. Initially, we employ the YOLOv7 detector to extract furniture from interior images. We then extract features such as style, color, room type, and material to create a dataset that can be represented as a graph.

7.4.1 Creating the graph

To begin, we must represent our data from the CSV file as a graph where each node corresponds to an image with its associated features. We create edges between nodes based on these features. If images share the same room type and style, we create edges between them, and the edge weight increases if they have the same color or material. This process results in a graph with weighted edges.

Each node must also include the image itself. This is achieved by employing a pre-trained ResNet-18 model with the last layer removed to represent the image as a feature vector. The feature vector of the image is concatenated with existing features such as room type, style, color, and material to create a representation for each node.

7.4.2 Creating GCN

Subsequently, we create a PyTorch Geometric Data object, which acts as an input for the PyTorch Geometric framework. The Data object is initialized using the following parameters: x, representing the feature vectors of the nodes; edge_index, denoting the indices of the nodes connected by edges; and edge_weight, indicating the weights assigned to the edges. These parameters, which were generated earlier, are now employed to construct a comprehensive graph representation.

The output of the GCN is a compatibility matrix, which is constructed according to the following rules: If the images belong to the same interior, their compatibility is set to 100. Otherwise, their compatibility increases proportionally to the number of shared features between the images. This approach captures the degree of compatibility between images based on the similarity of their attributes.

Now that we have the input Data and the output compatibility matrix, we can build a GCN model implemented as a PyTorch module. Below, you can see the structure of the GCN model:

```
class GCN(torch.nn.Module):
  def __init__(self, num_features):
    Initialize GCNConv layers and BatchNorm1d layers
    Initialize a fully connected layer
  def forward(self, data):
    Extract x, edge_index, edge_weight from data
    Pass x through the first GCNConv layer and BatchNorm1d layer
    Apply Leaky ReLU activation and dropout
    Pass x through the second GCNConv layer and BatchNorm1d layer
    Apply Leaky ReLU activation and dropout
    Pass x through the third GCNConv layer and BatchNorm1d layer
    Apply Leaky ReLU activation
    Pass x through the fully connected layer
    Scale x using sigmoid function and map to range 0 to 100
    Compute compatibility_matrix using torch.mm function
    return compatibility_matrix
```

In this pseudo code, you can observe that the data passes through three convolution layers for learning graph representation, followed by batch normalization to improve training stability, a Leaky ReLU activation function, and dropout for regularization. After that, the data is passed through the fully connected layer, and the output is scaled using the sigmoid function, mapping it to a range from 0 to 100. In the end, a compatibility matrix between each pair of nodes is created using the torch.mm function.

7.4.3 Algorithm summary

We have developed an algorithm that utilizes the YOLOv7 detector to extract patterns and attributes from images. The detector generates bounding boxes and class labels for the detected attributes, which are subsequently cropped from the original image. A set of four VGG models is then applied to the cropped attribute images to predict their style, material, color, and room type. These cropped images and their corresponding features are then used to construct a Graph Convolutional Neural Network (GCN) that generates a compatibility matrix for each element of interior design.

In Figure 7.1, you can see the scheme of the algorithm.

As a result of this approach, we are able to predict the compatibility between every element of interior design, thereby allowing us to create new combination.

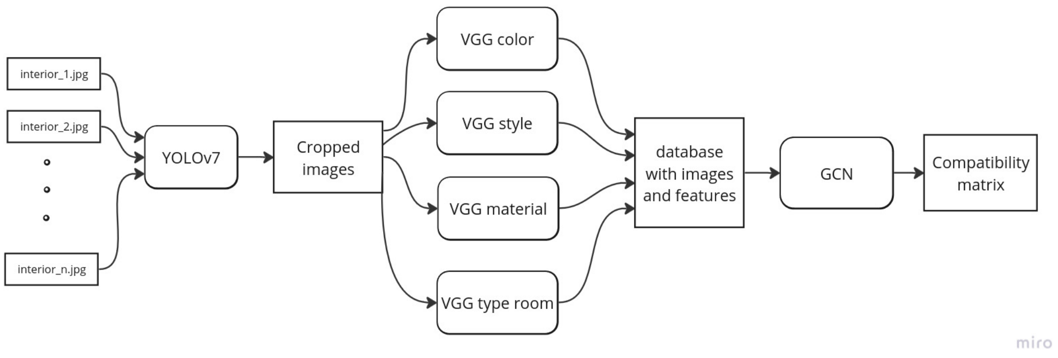


Figure 7.1 Algorithm

Chapter 8

Results and experiments

In this chapter, we will present the results of our experiments on detecting attributes of interior design and feature extraction. The experiments were conducted using the YOLOv7 detector and four VGG models trained for style, material, color, and type of room recognition. Additionally, we conducted experiments using the GCN to create interior design combinations based on the predicted object compatibility. We provide a table in the attachment that lists the sources of all images used in this section.

8.1 Extracting features

In this section, we present the results of training the VGG models for feature extraction. Specifically, we report the accuracy scores for each category of color, style, material, and type of room.

To evaluate the accuracy of each VGG model, we generated a confusion matrix. The algorithm first computed the total number of classes (ncls). Then, for each class in the matrix, the algorithm computed the number of correctly predicted instances (acc) by taking the diagonal element corresponding to the class. Finally, the algorithm summed up the acc values for all classes and divided the result by ncls to obtain the total accuracy. The process of creating the confusion matrix is outlined in the following pseudocode:

```
ncls = len(conf_mat)
tot_acc = 0
for we in range(ncls):
acc = conf_mat[i][i]
if we > 0:
    acc += conf_mat[i][i-1]
if we < ncls - 1:
    acc += conf_mat[i][i+1]
tot_acc += acc
tot_acc = tot_acc / ncls
print("%.2f" % (tot_acc * 100))
```

8.1.1 Color model

After training the color model using the new dataset, we calculated the accuracy using the confidence map and obtained an accuracy of 97.7%.

Figure 8.1 shows a visualization of the color model at work. Additional visualizations can be found in the file vizual_color.ipynb.

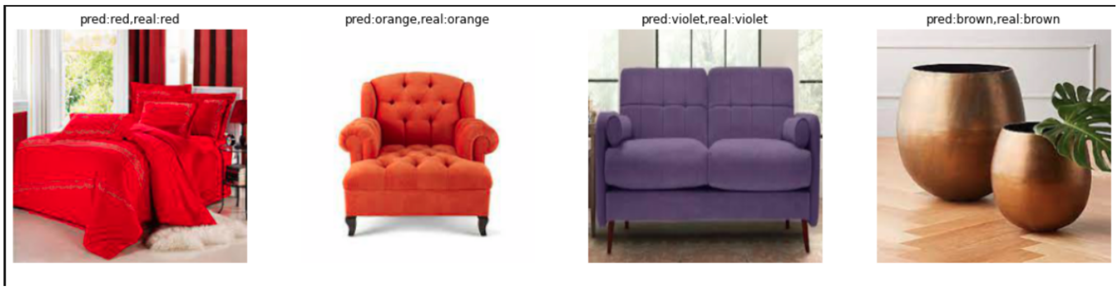


Figure 8.1 Vizualization color model

8.1.2 Material model

Using the same method, a material model achieved an accuracy of 97.1%. The visualization of this model in action is shown in 8.2. However, achieving high accuracy with a material model is challenging because objects often consist of multiple materials. Therefore, it is necessary to consider the several highest values when predicting the materials.

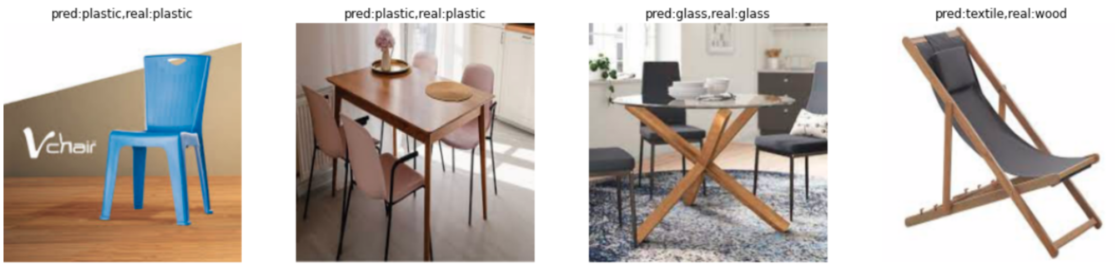


Figure 8.2 Vizualization material model

8.1.3 Style model

Applying the same evaluation method, the accuracy of the style model was found to be 93.1%. This lower accuracy is due to the fact that styles often overlap and it is difficult to determine with certainty which specific style an object belongs to. Additionally, a visualization of the working style model can be seen in 8.3.

8.1.4 Room type model

The final model trained to extract features is the type of room, achieving an accuracy of 93% on validation data. This model was trained on cropped images of objects commonly found in interior design, which may appear in multiple room types. For example, a table may be present in both a living room and a bedroom, leading to a lower accuracy than the color model.

Despite this limitation, the model performs predictably on validation data. To further test its accuracy, we evaluated the model on full room images and achieved a satisfactory accuracy of 92%. Figure 8.4 shows an example of the model's predictions on full room images.

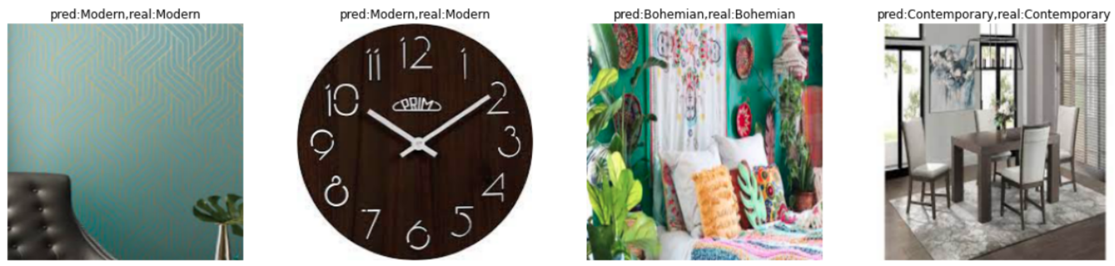

pred:Modern,real:Modern
pred:Modern,real:Modern
pred:Bohemian,real:Bohemian
pred:Contemporary,real:Contemporary

Figure 8.3 Vizualization style model

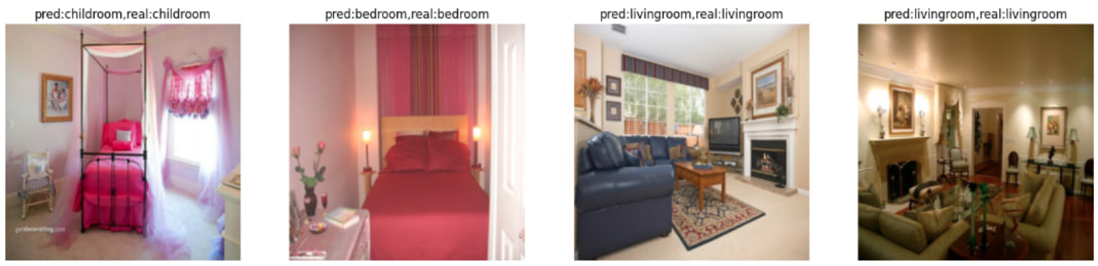

pred:childroom,real:childroom
pred:bedroom,real:bedroom
pred:livingroom,real:livingroom
pred:livingroom,real:livingroom

Figure 8.4 Vizualization type_room model

8.2 Detector

For detecting patterns and attributes, we trained a YOLOv7 model on the ADE20K dataset. However, there are only 2551 images of interior spaces such as bedrooms, living rooms, and children's rooms with attributes. This dataset is not sufficient for training the model to detect 14 classes of attributes. As a result, while training the model, we only achieved an accuracy of 68%, which is not ideal. Furthermore, the accuracy was lowered for objects that were rarely found in the dataset. An example of a detected image is shown in 8.5.

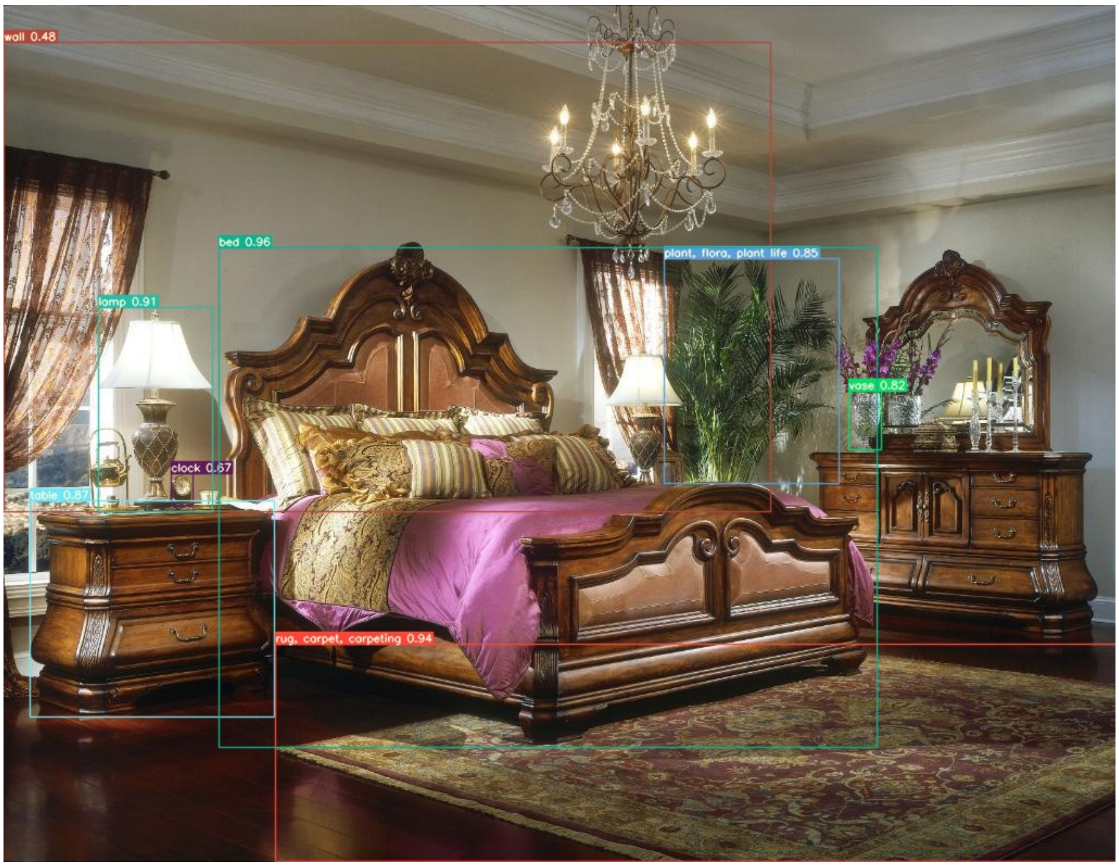


Figure 8.5 Detected attributes of interior design

8.3 Predicting compatibility

The objective of this experiment is to assess the relevance of the predictions made by the GCN. The experimental procedure involves selecting four interior objects and requesting the neural network to choose the most compatible image out of five options.

8.3.1 Method of execution

The experiment will be conducted using the Bonn_Furniture_Styles dataset [12], which has not been used for training. Initially, the VGG models will be employed to predict style, material, room type, and color, thereby generating a CSV file containing the features.

Also have created 20 distinct interior designs for the purpose of conducting this experiment. The primary objective is not to create flawless interiors, but rather to evaluate the model's ability to predict image compatibility and assess how well the selected images correspond with the given input images.

The experiment involves selecting each of the combinations we have created and examining the attribute type of the last element in the list of images, which then becomes the predicted element. Next, we randomly choose four objects of the same type from my dataset. This data is subsequently processed through a Graph Convolutional Neural Network (GCN), which generates a prediction matrix. From this matrix, we select the attribute that has the highest value in relation to the given images.

By doing so, we aim to gain insights into the model's performance and its potential for practical application in the field of interior design.

8.3.2 Method of evaluating

A challenge in evaluating this experiment lies in the difficulty of assessing the design through computational methods. Therefore, the experiment will be evaluated by a colleague who works as an interior designer. Additionally, this experiment will be conducted on two different models, with the aim of comparing their performance and determining the more effective model in predicting compatibility among interior objects.

I have generated 100 distinct interior designs for the purpose of conducting this experiment. The primary objective is not to create flawless interiors, but rather to evaluate the model's ability to predict image compatibility and assess how well the selected images correspond with the given input images. By doing so, we aim to gain insights into the model's performance and its potential for practical application in the field of interior design.

8.3.3 Evaluation results

In this experiment, we created a total of 20 distinct interior designs, each containing 5 images. Consequently, 100 predictions were made based on these interiors. Presenting all of these predictions in this report would require more than 50 pages, so we have chosen to showcase a few representative examples of both positive and negative predictions.

In the provided images8.6, 8.7, 8.8, 8.9, the first row displays the given interior design, while the second row presents the candidate images from which the selection was made. The chosen image is highlighted with a green box. Above each image, the Graph Convolutional Neural Network (GCN) prediction is displayed, indicating the compatibility score with the given images.

In the displayed images, it is evident that the evaluation of these predictions is somewhat subjective, as an objective assessment in design is challenging to achieve. Nevertheless, a friend of mine who works as an interior designer assessed the predictions and determined that 74% of them were positively aligned with the intended design. This provides an indication of the model's effectiveness in predicting compatible images within the context of interior design.

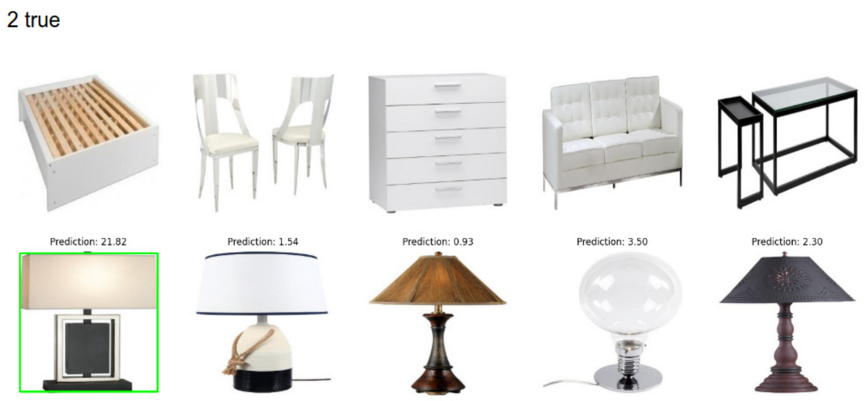


Figure 8.6 Positive predicted lamp

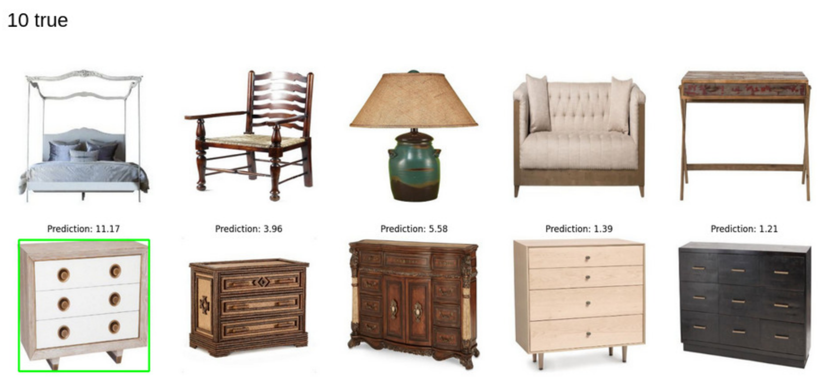


Figure 8.7 Positive predicted dresser

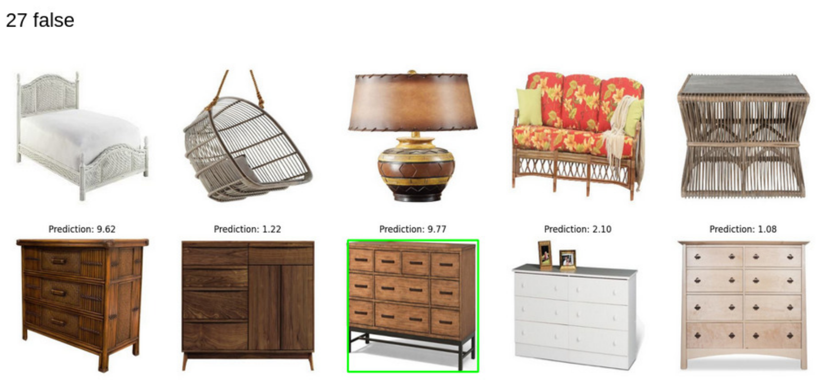


Figure 8.8 Negative predicted dresser

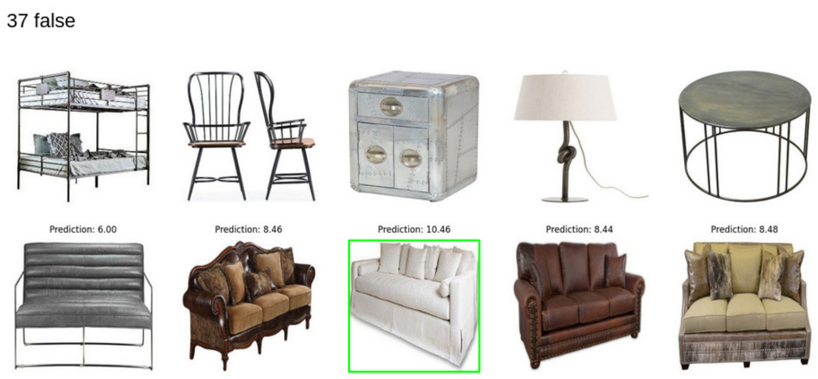


Figure 8.9 Negative predicted sofa

Chapter 9

Conclusion

In this study, we propose an algorithm capable of detecting interior design objects, extracting specific features from these objects, and subsequently utilizing a convolutional graph neural network to predict compatibility between interior design objects. Based on these predictions, various combinations can be generated.

To achieve this, we conducted a comprehensive review of existing literature on the application of neural networks in interior design, which allowed us to understand the main challenges in this domain and gather ideas for implementing our tasks. We also studied various types of neural networks and their use, helping us select the appropriate methods to achieve our objectives.

We believe that the goals of this work have been accomplished, contributing to the problem of selecting patterns and accessories in interior design. We created suitable datasets and implemented an algorithm that predicts compatibility between interior design objects based on extracted features, which assists in generating combinations.

Evaluating results in the design field can be challenging, but an experiment assessed by a designer revealed that our algorithm proposed suitable variants in 74% of cases. While not perfect, this outcome demonstrates the potential of the algorithm. To improve these results, a higher quality and larger dataset for training would be required, which may not be feasible due to resource constraints.

This work establishes a solid foundation for further advancements in this field. It serves as the basis for developing generative models capable of creating complete interior designs tailored to users' preferences, which could potentially transform the landscape of the interior design industry.

Bibliography

1. RACEC, Emil; BUDULAN, Stefania; VELLIDO, Alfredo. Computational Intelligence in architectural and interior design : a state-ofthe-art and outlook on the field. In: 2016.
2. KIM, Jin. Approach to the Extraction of Design Features of Interior Design Elements Using Image Recognition Technique. *CAADRIA proceedings*. 2018.
3. BANAEI, Maryam; AHMADI, Ali; YAZDANFAR, Abbas. Application of AI methods in the clustering of architecture interior forms. *Collection of Frontiers of Architectural Research*. 2017, vol. 6, pp. 360–373.
4. MOURAD, Jbene; DRISSI EL MALINAI, Ahmed; EL HASSOUNI, Mohammed. Fusion of Convolutional Neural Network and Statistical Features for Texture classification. In: 2019, pp. 1–4. Available from DOI: `10.1109/WINCOM47513.2019.8942469`.
5. RAHMAN, Molla Hafizur; YUAN, Shuhan; XIE, Charles; SHA, Zhenghui. Predicting human design decisions with deep recurrent neural network combining static and dynamic data. *Design Science*. 2020, vol. 6, e15. Available from DOI: `10.1017/dsj.2020.12`.
6. KIM, Jinsung; LEE, Jin-kook. Stochastic Detection of Interior Design Styles Using a Deep-Learning Model for Reference Images. *Applied Sciences*. 2020, vol. 10, p. 7299. Available from DOI: `10.3390/app10207299`.
7. AGGARWAL, Charu C. An Introduction to Neural Networks. In: *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018, pp. 1–52. ISBN 978-3-319-94463-0. Available from DOI: `10.1007/978-3-319-94463-0_1`.
8. IOANNOU, Yani. *Structural Priors in Deep Neural Networks*. 2017. Available from DOI: `10.17863/CAM.26357`. PhD thesis.
9. PHUNG, Van Hiep; RHEE, Eun Joo. A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. *Applied Sciences*. 2019, vol. 9, no. 21. ISSN 2076-3417. Available from DOI: `10.3390/app9214500`.
10. KIPF, Thomas N.; WELLING, Max. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. Available from arXiv: `1609.02907 [cs.LG]`.
11. ZHOU, Bolei; ZHAO, Hang; PUIG, Xavier; FIDLER, Sanja; BARRIUSO, Adela; TORRALBA, Antonio. Scene Parsing Through ADE20K Dataset. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
12. AGGARWAL, Divyansh; VALIYEV, Elchin; SENER, Fadime; YAO, Angela. Learning Style Compatibility for Furniture. In: *German Conference on Pattern Recognition*. 2018, pp. 552–566.

13. WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.* 2022. Available from arXiv: `2207.02696 [cs.CV]`.

14. SIMONYAN, Karen; ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition.* 2015. Available from arXiv: `1409.1556 [cs.CV]`.

Contents of the attached media

- `readme.txt` .... a brief description of the media content
- `src` .... implementation
  - `datasets`
  - `features_models` .... training and visualization models for features extraction
  - `scrap_data_vgg` .... directory containing scrappers
  - `create_labels` .... scripts that extract features and create labeled data
  - `GCN` .... training and visualization GCN model
  - `detector` .... directory containing training detector
  - `experiments` .... directory contains visualization of experiments
- `text`
  - `thesis.pdf` .... thesis text thesis text in PDF format