**National College of Computer Studies**

**Tribhuvan University**

**Institute of Science and Technology (IoST)**



Project Report On

**Furniture Classification System**

**Submitted To:**

National College of Computer Studies

Department of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT)

**In partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT)**

**Submitted By:**

Utsarga Manandhar

(BSc.CSIT 6th Semester)

# **Abstract**

This project presents a deep learning-based approach to furniture classification using Convolutional Neural Networks (CNNs). The goal is to develop an automated system capable of identifying and categorizing various types of furniture, such as chairs, tables, and sofas, from images. The model is trained on a dataset of labeled furniture images, incorporating techniques such as data augmentation, batch normalization, and dropout to enhance generalization and reduce overfitting. The CNN architecture extracts key features from the images, progressively learning from simple patterns to complex structures. Performance metrics, including accuracy, precision, recall, and F1-score, are used to evaluate the model's ability to classify furniture categories effectively. The model achieves high classification accuracy and demonstrates the potential for real-time applications in fields such as e-commerce, interior design, and augmented reality. This project highlights the effectiveness of CNNs in image classification tasks and offers a foundation for future improvements in automated furniture recognition systems.

# **Acknowledgement**

We would like to extend our heartfelt thanks and gratitude to our supervisor, Mr. Mausam Rajbanshi, for granting us the valuable opportunity to work on **“Furniture Classification System”** project. Their insightful guidance and support have greatly contributed to the success of this project, allowing us to explore new tools and technologies.

Our sincere appreciation goes to NCCS College for their constant supervision, guidance, and the necessary resources provided, which played a crucial role in the completion of this project. The support from the library and staff members of NCCS has been invaluable, and we are grateful for their cooperation and encouragement. Here, we would like to show an attitude of reverence to all of them who have contributed directly or indirectly to make the study a reality.

Finally, we shall be grateful to those people who read this project and who shall benefit from this project at present and in future.

Yours sincerely,

Utsarga Manandhar

**TABLE OF CONTENT**

Contents

[**Abstract** i](#_Toc191042139)

[**Acknowledgement** ii](#_Toc191042140)

[**1. Introduction** 1](#_Toc191042141)

[**2. Objective** 2](#_Toc191042142)

[**3. Methodology** 3](#_Toc191042143)

[**3.1. Data** 3](#_Toc191042144)

[**3.1.1. Data Collection** 3](#_Toc191042145)

[**3.1.2. Data Properties** 3](#_Toc191042146)

[**3.2. Neural Network Architecture** 3](#_Toc191042147)

[**3.2.1. Block Representation** 3](#_Toc191042148)

[**3.2.2. Layer Explanation** 5](#_Toc191042149)

[**3.2.3. Input and Output** 8](#_Toc191042150)

[**3.3. Activation Functions** 8](#_Toc191042151)

[**3.4 Loss Function** 9](#_Toc191042152)

[**3.4.1. Importance** 9](#_Toc191042153)

[**3.4.2. Function Used** 9](#_Toc191042154)

[**3.5. Forward Propagation** 10](#_Toc191042155)

[**3.6. Backpropagation** 10](#_Toc191042156)

[**4. Testing** 11](#_Toc191042157)

[**5. Results** 12](#_Toc191042158)

[**6. Conclusion** 14](#_Toc191042159)

[**7. References** 15](#_Toc191042160)

# **1. Introduction**

Furniture classification project involves identifying various types of furniture like chair, table, etc. Furniture classification using deep learning involves developing a model that can automatically identify and categorize different types of furniture from images. By utilizing advanced techniques like Convolutional Neural Networks (CNNs), the model learns to recognize patterns, textures, and features specific to various furniture types. This approach is essential for applications in e-commerce, interior design, and augmented reality, where automated image classification can enhance user experience by enabling faster and more accurate furniture identification. Through training on large, labeled datasets, deep learning models can achieve high accuracy and efficiency in classifying furniture items.

# **2. Objective**

The objectives of this project are

* **Develop a neural network model** to automatically classify furniture images.
* **Utilize deep learning techniques** to build the model, leveraging Convolutional Neural Networks (CNNs).
* **Categorize images into predefined furniture categories** such as chairs, tables, sofas, etc.
* **Achieve high accuracy** by optimizing hyperparameters and improving model performance.
* **Enhance feature extraction techniques** to better capture the characteristics of different furniture types.
* **Ensure the model's efficiency** in real-time furniture image classification applications.

# **3. Methodology**

## **3.1. Data**

### **3.1.1. Data Collection**

The dataset consists of images stored in folders representing different furniture categories. The dataset was structured such that each folder contains images belonging to a specific furniture class. Images were collected from various sources, including open datasets like Kaggle.

### **3.1.2. Data Properties**

* **Total images**: Extracted from the dataset and consists of Almirah, Chair, Fridge, Table, and TV dataset. Each dataset consists of 3000 images each.
* **Image dimensions**: Varies, resized to a standard size 180X180 pixels for consistency.
* **Color space**: RGB (3 channels) to retain color information.
* **Number of categories**: Multiple furniture classes representing various furniture types such as Almirah, Chair, Fridge, Table, and TV.
* **Augmentation Techniques**: Rotation, flipping, zooming, and brightness adjustments applied to improve model generalization.

## **3.2. Neural Network Architecture**

### **3.2.1. Block Representation**

* **Input Layer**: The input layer receives the image data, typically with dimensions of 180x180 pixels. Since image data can come in various formats (RGB, grayscale, etc.), the input size is standardized to 180x180 pixels with 3 color channels (for RGB). This ensures that all input images have consistent dimensions, which is critical for the neural network to process them properly. The images will be resized if needed, and pixel values are often normalized to a range between 0 and 1 for better convergence during training.
* **Convolutional Layers**: Convolutional layers apply filters (or kernels) to the input image to detect specific features like edges, textures, or patterns. As the model goes deeper, the convolutional layers detect more complex and abstract features. The first convolutional layer typically learns simple features (such as edges), while deeper layers learn more intricate features (such as shapes or objects). In your model, there are multiple convolutional layers, each applying filters to the feature maps from the previous layer. Eg: After the first convolution (Conv2D): (None, 180, 180, 16) – The number of channels increases as the model learns more complex features.
* **Batch Normalization**: Batch normalization stabilizes and accelerates the training process. It normalizes the activations of each layer in the network, ensuring that they have a mean of 0 and a variance of 1. This is done on a mini-batch level, which helps reduce internal covariate shifts and leads to faster convergence. Batch normalization can also act as a regularizer, potentially reducing the need for other regularization techniques like dropout. By reducing the internal covariate shift, batch normalization can help the model learn faster and improve generalization by maintaining stable distributions of activations throughout the network.
* **MaxPooling Layers**: MaxPooling layers perform a downsampling operation by reducing the spatial dimensions (height and width) of the feature maps, while retaining the most important features (the maximum value) from each region in the input. For example, a 2x2 max-pooling operation will reduce the size of the feature map by a factor of 2, preserving only the most prominent features from that area. MaxPooling reduces the computational cost and helps the model generalize by extracting the most important spatial features. MaxPooling reduces the dimensionality of the data and helps to make the model invariant to small translations in the image, improving robustness. Eg: After the first MaxPooling2D layer: (None, 90, 90, 16)
* **Dropout Layers**: Dropout is a regularization technique used to prevent overfitting during training. It works by randomly setting a fraction of the input units to zero (or "dropping out") at each update during training. This forces the model to learn more robust features that are not reliant on specific neurons, helping to generalize better to unseen data. The dropout rate determines the proportion of neurons to drop out, typically set to values like 0.5. In this model, dropout is applied after the third MaxPooling layer, before the fully connected layers, helping to prevent overfitting. Dropout helps to improve the generalization of the model, reducing the likelihood of overfitting.
* **Fully Connected Layers (Dense)**: Fully connected layers are used to combine the features extracted by the convolutional and pooling layers and perform the final classification. These layers consist of neurons that are fully connected to all activations from the previous layer. The dense layers transform the feature maps into a vector, which can be passed to the output layer for classification. Example Output Shape: (None, 128) – The first dense layer has 128 neurons, each representing a learned feature. After this, the model learns to classify based on these features. Fully connected layers aggregate the features learned by the convolutional layers and make the final decision on the class of the image.
* **Output Layer**: The output layer is the final layer in the network, which produces the prediction for the classification task. It uses the **softmax** activation function, which converts the raw outputs (logits) into probabilities for each class. The softmax function outputs a vector where each element represents the probability of the image belonging to a specific class, with all probabilities summing to 1. Softmax ensures that the model provides a clear probabilistic classification, making it easier to interpret the model's predictions.

### **3.2.2. Layer Explanation**

* **Sequential Layer:** The output shape is (None, 180, 180, 3). The Sequential layer is the container for the entire model. It specifies that the input image has a size of 180x180 pixels with 3 color channels (RGB). The None represents the batch size, which can be variable. This layer does not perform any operations; it simply holds the sequence of layers that follow.
* **Rescaling Layer:** The output shape is (None, 180, 180, 3). This layer performs image normalization by scaling the pixel values of the input image to a range between 0 and 1. It divides all pixel values by 255, making the data more suitable for training the neural network. This step helps with convergence during training by making the data less sensitive to large input variations.
* **Conv2D (First Convolutional Layer):** The output shape is (None, 180, 180, 16). The param is 448. This layer applies 16 convolutional filters (kernels) to the input image to extract local features like edges, corners, and textures. Each filter generates a 2D feature map, which is stacked to create the 4D output tensor. The number of parameters (448) is calculated based on the kernel size (3x3) and the number of input channels (3, RGB), with each filter having weights and biases.
* **MaxPooling2D (First Max Pooling Layer):** The output shape is (None, 90, 90, 16)**.** The params is 0.This layer applies **max pooling** to the feature map generated by the first convolutional layer. A typical 2x2 pooling window is used, which reduces the spatial dimensions (height and width) of the feature map by half, from 180x180 to 90x90. This operation retains the most prominent features and reduces the computational load without losing too much spatial information.
* **Conv2D (Second Convolutional Layer):** The output shape is (None, 90, 90, 32)**.** The params is 4,640.This layer applies **32 convolutional filters** to the feature map from the first max-pooling layer. The filters learn more complex patterns and details, increasing the depth of the feature map from 16 to 32. The number of parameters (4,640) is computed based on the kernel size (3x3) and the number of channels (16 from the previous layer).
* **MaxPooling2D (Second Max Pooling Layer):** The output shape is (None, 45, 45, 32)**.** The params is 0.This max-pooling layer reduces the size of the feature map generated by the second convolutional layer. It uses a 2x2 pooling window to reduce the spatial dimensions from 90x90 to 45x45, retaining the most important features while reducing computational complexity.
* **Conv2D (Third Convolutional Layer):** The output shape is (None, 45, 45, 64)**.** The params is 18,496.This layer applies 64 convolutional filters to the output of the second max-pooling layer. The filters capture even more complex features, such as intricate textures or higher-level patterns. The depth of the feature map increases to 64, and the number of parameters (18,496) comes from the kernel size (3x3) and the number of input channels (32).
* **MaxPooling2D (Third Max Pooling Layer):** The output shape is None, 22, 22, 64)**.** The params is 0.This max-pooling layer reduces the size of the feature map generated by the third convolutional layer. Using a 2x2 pooling window, it decreases the spatial dimensions from 45x45 to 22x22 while maintaining the important features. The depth remains 64.
* **Dropout Layer:** The output shape is (None, 22, 22, 64). The params is 0. The Dropout layer randomly disables a fraction of neurons during training to prevent overfitting. It helps the model generalize better by forcing it to rely on different parts of the network rather than memorizing specific patterns. The output shape remains the same as the previous layer, but with random neurons set to zero during training.
* **Flatten Layer:** The output shape is (None, 30976)**.** The params is 0**.** The Flatten layer converts the multi-dimensional feature map (22x22x64) into a one-dimensional vector of 30,976 values, which can be passed to the fully connected layers for classification. Flattening is essential as dense layers expect a vector as input rather than a multi-dimensional array.
* **Dense (Fully Connected Layer):** The output shape is (None, 128). The params is 3,965,056. This fully connected (dense) layer takes the flattened vector and connects it to 128 neurons. Each neuron is connected to all 30,976 values from the previous layer, resulting in a large number of parameters (3,965,056). This layer is responsible for learning high-level representations and combining features from the previous layers for classification.
* **Dense (Output Layer):** The output shape is (None, 5). The params is 645. The final Dense layer is the output layer, with 5 neurons corresponding to the number of classes for classification. It uses a softmax activation function, which outputs a probability distribution over the 5 classes. The number of parameters (645) is calculated based on the number of neurons in the previous layer (128) and the output neurons (5).

### **3.2.3. Input and Output**

* **Input**: The input to the model consists of resized furniture images, each of which is fed into the neural network after undergoing preprocessing. These images are resized to a consistent dimension of 180x180 pixels to ensure uniformity across all input data. Typically, the images are also normalized (scaled to a range of 0 to 1) to improve the efficiency and stability of the training process. This step ensures that the pixel values, which originally range from 0 to 255, are divided by 255 to make them more manageable for the neural network.
* **Output**: The output of the model is the predicted furniture category, represented by a probability distribution over the possible classes. The neural network assigns a probability to each of the predefined classes based on the features it has learned during training. This is achieved through the final softmax layer in the network. The softmax activation function takes the raw output values (logits) from the last dense layer and converts them into a range between 0 and 1, ensuring the sum of all probabilities equals 1. The class with the highest probability is considered the model's prediction. For example, if the model predicts the class 'Table' with the highest probability, the output will show the predicted class as 'Table' with a confidence value based on the model's calculated probability.

## **3.3. Activation Functions**

* **ReLU (Rectified Linear Unit)**: ReLU is an activation function commonly used in the hidden layers of neural networks. It introduces non-linearity by outputting the input directly if it is positive, and zero otherwise. This activation function helps the model learn complex patterns by allowing it to better handle both positive and negative values, preventing the model from being limited to linear relationships. One of the key advantages of ReLU is its ability to mitigate the vanishing gradient problem, which is common in older activation functions like sigmoid and tanh. In these functions, gradients become very small during backpropagation, leading to slow or halted learning. ReLU’s gradient is either 0 or 1, which helps the network learn efficiently, especially in deep networks where gradients can otherwise vanish or explode.
* **Softmax**: Softmax is an activation function typically used in the output layer of neural networks for multi-class classification tasks. It transforms the raw output values (called logits) from the final dense layer into a probability distribution. The function computes the exponential of each logit and normalizes these values so that they sum to 1. The result is a vector of probabilities, where each element represents the likelihood of the corresponding class being the correct classification.

## **3.4 Loss Function**

### **3.4.1. Importance**

A loss function is a crucial component of any machine learning model as it quantifies the error or discrepancy between the predicted output and the actual ground truth values. The primary role of the loss function is to guide the optimization process during training. By calculating the error, the model can adjust its weights to minimize the loss, thereby improving its performance over time. The optimization process (typically using algorithms like gradient descent) relies on the gradients of the loss function to update model parameters in a direction that reduces the error. In essence, the choice of the loss function directly influences how effectively the model learns and generalizes.

### **3.4.2. Function Used**

* **Categorical Crossentropy**: For multi-class classification tasks, such as predicting the category of a furniture image, the Categorical Crossentropy loss function is commonly used. This loss function is particularly suitable when the output is a probability distribution across multiple classes, which is exactly the case with the softmax activation in the output layer. Categorical Crossentropy measures the difference between the true label (in one-hot encoded format) and the predicted distribution for each class.

## **3.5. Forward Propagation**

During forward propagation, the input data (the image of furniture) is passed through the network layer by layer, starting from the input layer to the output layer. Each layer performs a series of operations (such as convolutions, activation functions, and pooling) to progressively transform the data and extract relevant features. These features are learned representations that capture important patterns, edges, textures, or shapes within the image, which are useful for classification. The learned weights of each layer determine how the input data is transformed. As the image moves through each layer, the network refines these feature representations, helping the model understand higher-level patterns like the shapes of furniture, textures, and structures. In the final output layer, a probability distribution is produced, with the highest probability corresponding to the model's predicted class for the input image.

## **3.6. Backpropagation**

Backpropagation is the process through which the model updates its weights to reduce prediction errors. After forward propagation, the loss function calculates the error between the predicted and actual output. Backpropagation then computes the gradients (partial derivatives) of the loss with respect to each weight in the network using automatic differentiation. These gradients show how much each weight contributed to the error. The weights are then updated using an optimization algorithm like gradient descent, which adjusts the weights in the direction that minimizes the loss. This process is repeated iteratively to improve the model's performance.

# **4. Testing**

The trained model is evaluated on a separate test dataset to measure its performance. The evaluation includes multiple metrics such as:

* **Accuracy**: Accuracy is a commonly used metric that measures the overall performance of the model by calculating the percentage of correctly classified instances. It measures the percentage of correctly classified images.
* **Precision**: Precision (also known as positive predictive value) focuses on how many of the positive predictions made by the model are actually correct. It determines the accuracy of positive predictions.
* **Recall**: Recall (also known as sensitivity or true positive rate) measures how well the model identifies all relevant instances of the positive class. It is useful when false negatives are more costly than false positives, such as in medical diagnosis. It measures how well the model identifies all relevant instances.
* **F1-score**: The F1-score is the harmonic mean of precision and recall, providing a balanced measure between the two. It provides a balance between precision and recall.

# **5. Results**

The model's performance is analyzed using:

* **Confusion Matrix**: A confusion matrix is a tool used to visualize the performance of a classification model, especially in terms of how well it distinguishes between different classes. It displays the true positives, false positives, true negatives, and false negatives for each class. This allows you to assess not only the accuracy of the model but also its behavior across different classes, such as which classes are being misclassified and which ones are being correctly predicted. The matrix helps identify patterns of confusion between similar classes and is essential for diagnosing model weaknesses and improving its performance.
* **Accuracy Graphs**: **Accuracy graphs** plot the model’s training and validation accuracy over multiple epochs, providing insights into the model’s learning process. By comparing these two metrics, you can observe if the model is overfitting, underfitting, or generalizing well. If the training accuracy is significantly higher than validation accuracy, it may indicate overfitting, meaning the model is memorizing the training data rather than learning to generalize. On the other hand, if both accuracies are low, the model might be underfitting, suggesting it is not complex enough to capture the underlying patterns in the data.
* **Loss Graphs**: **Loss graphs** visualize how the model's loss function (such as categorical crossentropy) changes over time during training. These graphs show how well the model is minimizing the classification errors. A well-trained model will have a steadily decreasing loss curve. If the loss plateaus or increases, it could indicate issues like learning rate problems or the model's inability to learn effectively. Tracking both training and validation loss is important to assess overfitting—if the training loss decreases but the validation loss increases, it suggests that the model is memorizing the training data and not generalizing well.
* **Experimentation with Hyperparameters**: Experimentation with hyperparameters is a critical part of optimizing model performance. Hyperparameters, such as the learning rate, batch size, and optimizer selection, significantly impact how the model learns. The learning rate controls how big each update step is during training, while batch size affects the number of samples the model processes before updating weights. Selecting an appropriate optimizer (like Adam, SGD, or RMSprop) can also influence how effectively the model converges. Experimenting with these parameters allows you to find the best configuration that maximizes accuracy and minimizes overfitting or underfitting. Hyperparameter tuning often involves running multiple experiments with different combinations and monitoring performance using the metrics and graphs.

# **6. Conclusion**

This project successfully implements a deep learning model for furniture classification using Convolutional Neural Networks (CNNs), proving their effectiveness in image classification tasks. By incorporating **data augmentation**, the model became more robust to variations in the input images, while **batch normalization** helped stabilize training and **dropout** reduced overfitting, improving generalization. Hyperparameter tuning, including adjustments to the learning rate and optimizer, enhanced the model's performance. The evaluation metrics, such as accuracy, precision, recall, and F1-score, confirm the model's ability to reliably classify furniture categories. This project demonstrates the potential of CNNs for classification tasks and can be extended to other image recognition challenges.

# **7. References**

* <https://www.kaggle.com/>
* <https://youtu.be/V61xy1ZnVTM?si=kHytwIY2ZkH-YftH>
* <https://chatgpt.com/>
* <https://github.com/sophiachann/ObjectDetectionProject-IKEAFurnituresRecommender>
* <https://universe.roboflow.com/mokhamed-nagy-u69zl/furniture-detection-qiufc>