JUNE, 2022

ASITEM (Architectural Sketch image to 3D model**)**

A PROJECT REPORT

Submitted by

REG. NUMBER: BL.EN.U4ECE18512	NAME: VASUMAN SENAPATHY
BL.EN.U4ECE18094	M.BESH VIGNESH REDDY
BL.EN.U4ECE18179	YANGALA PREM CHAND

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

“Electronic and Communications” ENGINEERING

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDYAPEETHAM

BENGALURU 560 035

JUNE-2022

JUNE, 2022

Annexure 2A
AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ENGINEERING, BENGALURU, 560035

BONAFIDE CERTIFICATE

This is to certify that the project report entitled “**ASITEM”** submitted by

VASUMAN SENAPATHY	BL.EN.U4ECE18512
M.BESH VIGNESH REDDY	BL.EN.U4ECE18094
YANGALA PREM CHAND	BL.EN.U4ECE18179

in partial fulfillment of the requirements for the award of the **Degree Bachelor of Technology** in “**Electronics and Communication Engineering.** “ is a bonafide record of the work carried out under my(our) guidance and supervision at Amrita School of Engineering, Bangalore. and “…name of the organization in case of external work”.

SIGNATURE
Dr. Nizampatnam Neelima
Department of ECE

SIGNATURE
Dr. Navin Kumar
Department of ECE

This project report was evaluated by us on ……… (Date...)

EXAMINER 1 EXAMINER 2 EXAMINER 3 EXAMINER 4

ACKNOWLEDGMENT

We would like to express our sincere gratitude to every person who has helped us with our project. Our heartfelt thanks to AMMA for her showers blessings on us.

Firstly, we would extend our sincere thanks to our principal Dr. Sriram Devanathan and the entire management for giving us the confidence and motivating us to undertake this project.

We would also like to convey our sincere thanks to Dr. Navin Kumar, Chairperson of the Department of Electronics and Communication for their constant support and assistance in all the efforts made for the project.

Our heartfelt thanks to our project guide, Dr. Nizampatnam Neelima, for all the guidance and encouragement provided throughout the project without which it would not have been possible for us to complete our project. The blessing, help, and support given by her from time to time shall carry us a long way in the journey of life on which we are about to embark. We would also like to express our heartfelt gratitude to Ms. Jeyanthi and Ms. Bhavana V, Project Coordinators of the Electronics and Communication Department.

In the process of doing the project, we have received suggestions and guidance from many teachers. We would like to thank all of them for their kind support and assistance in helping us do our project. Last but not least we would like to thank our family and friends for helping us to give our best in all our attempts in completing the project.

ABSTRACT

The model has always been important as it transfers knowledge to end-users. Yet, transforming the information that one includes into another has still major problems. It requires precise data, qualified personnel, and human intervention. This research aims to represent an automated reconstruction from low-level data sources to higher-level digital models to eliminate the architectural problems. This auto-conversion process only examines the architectural usage and makes a sample of its usability in different fields.

2D floor plans and elevation drawings in raster format, which are collected and/or produced from scratch, are used as datasets. These drawings are semantically segmented with three different Convolutional Neural Networks to obtain relevant architectural information since Deep Learning shows promising success in solving a wide range of problems. Semantically segmented drawings are then transformed into 3D by using Digital Geometry Processing methods. Lastly, a web application is introduced to allow any user to obtain a 3D model with ease. vi Semantic segmentation results in 2D and two case studies in 3D are evaluated and compared separately with different metrics to represent the accuracy of the process

To conclude, this research has proposed an automated process for the reconstruction of 3D models the state-of-the-art methods and made it ready for use even for a person without technical knowledge.

Keywords: Architectural Drawing Dataset, 3D Reconstruction, Semantic Segmentation, Convolutional Neural Networks, Digital Geometry Processing

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1. Overall System/Scenario:

Modeling has always been important in all disciplines since it is a way of representing information. It includes both abstract and concrete knowledge and conveys this knowledge to end-users. Sharing information through different models is a vital feature for all design-related industry stakeholders. Yet, transforming 2D information into 3D had some defects in perception even for a qualified human. Therefore, today a new level of information is required, which cannot be just seen as a representation but also can be perceived as data to allow this transformation.

Design-related disciplines such as architecture and engineering have continuously improved modes of modeling to broaden perception, collaboration, communication, and implementation. Firstly, 2D and 3D virtual representation styles have been introduced into our lives. In the late 50s and early 60s, Pronto by Hanratty and Sketchpad by Sutherland pioneered the developments in Computer-Aided Design (CAD). While Hanratty raised the first numerically controlled programming tool (Harris & Meyers, 2009), Sutherland made it possible to interact with a human and a machine to design based on line drawings in a 2D virtual environment (Sutherland, 1964) as shown in Figure 1.1. Later, these major events enriched the developments in CAD, and solid modeling methods emerged afterward the technological developments. Representation of edges, boundary surfaces, and primitive objects has broadened the horizon of CAD technologies and, today, designers can work in an environment in which they can create, define and control a 3D digital model (Tornincasa & Di Monaco, 2010).

Even though end users can easily gain perceptual experience with different representation modes via various tools, it is important to address the complexity of perceiving and matching the information of an object in different dimensions. For instance, it is troublesome to imagine a 3D version of 2D technical drawings. Likewise, it is hard to perceive 2D images as Real-life conditions without predicting depth information. In this kind of case, the information in one representation becomes a 4 data source that can be converted into another dimension. However, considerable expertise, time, and precise data are required to transfer low-level information to a high level. Therefore, the motivation of this dissertation is searching for a new way that can be applied to different industries to expose the information conveyed by various dimensions that make a difference in perception

Problem Statement

There is a growing demand for higher-level modeling in which the most applicable and available version is in 3D. Simulations and evaluation are important to simulate real-life conditions. Having a 3D model also allows contributors to understand and control design and management processes. Each stakeholder can state an opinion based on the 3D model. Manufacturing operations can be analyzed with these models. Planned expenditures for construction can be measured. Even real estate agencies now opt for showing digital models to clients for making them gain perspective on a building's condition (Hartmann, Gao & Fischer, 2008). Conservation projects require these models to restore, renovate or remodel the artwork (Pieraccini, Guidi & Atzeni, 2001). Figure 1.3 shows survey results representing the current usage and awareness of different working principles, and 3D modeling usage turns out the highest in both senses.

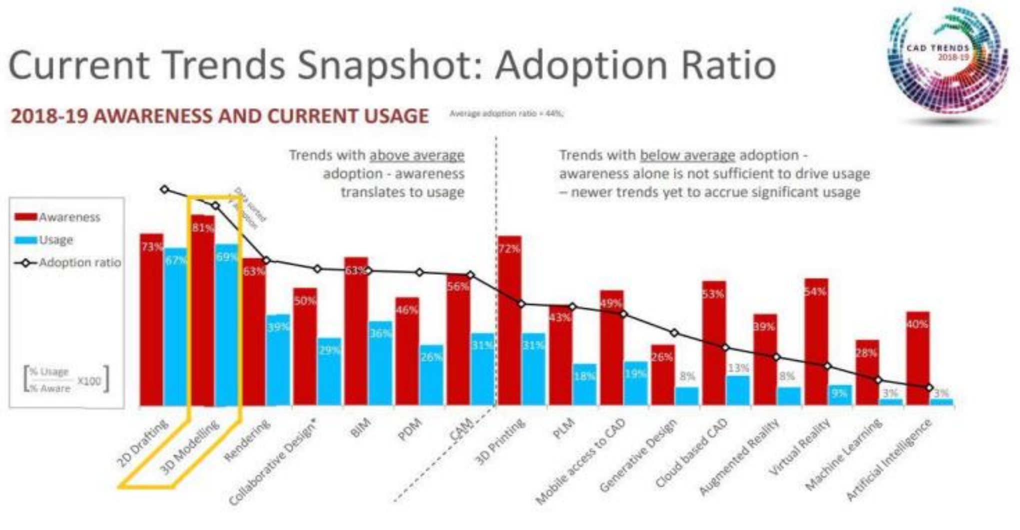


Figure (1.1) Adoption Ratio for 3-D Modelling

Though developing technologies propose new ways of 3D modeling to create a better perception environment for end-users, there are still major problems in benefiting low-level representations as data sources to construct 3D digital models, which is referred to as reconstruction/regeneration in this research. Since the data sources for regeneration processes can be different from one project to another, many drawbacks can be observed. For example, cases using aerial images suffer from an excessive amount of data samples, time consumption, and the need for highly qualified personnel for conversion (Suveg & Vosselman, 2004). In Pu and Vosselman's paper (2009), the problem of manual reconstruction is pointed out. They coined that when working manually, the creation of city models is a slow and expensive process since city models include many buildings in which complexity differs immensely. In their work El-Hakim, Beraldin, Picard & Godin (2004) also mentioned that modeling from scratch with CAD software might result in low-precise 3D models.

As implied above, it is tricky to perceive discrete information even for a qualified human perception. Moreover, employing one data source with a possibility of missing information makes a reconstruction process even trickier. The need for precise data, qualified personnel, or human intervention is inevitable in most cases. 6 Nonetheless, obtaining them is not always possible, and errors, precision, and time-consuming problems can emerge altogether. In this context, an automated reconstruction process with state-of-the-art methods can solve the problems indicated above. The following research questions should be answered to achieve this:

- Can deep learning and digital geometry processing techniques take place in the automation process of transforming low level to high(er) level?
- Which deep learning and digital geometry processing techniques are more suitable for the automation process?
- How much time will be spent on the process?
- Do the accuracy and precision of the reconstructions meet the needs of industry?

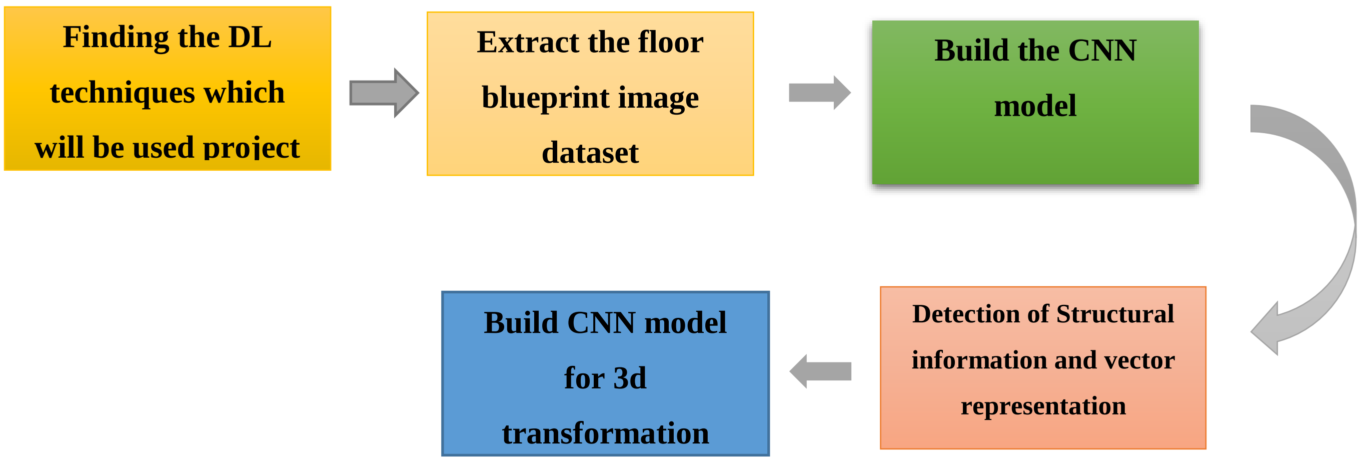
1.2 Objective:

This study aims to transform low-level information to a higher level by enabling a precise and automated reconstruction process with state-of-the-art automation methods, which all the industries can exploit from. While low-level information is extracted from a 2D environment, high-level information is constituted in 3D. In other words, 3D digital model reconstructions can be automatically obtained with 2D technical drawings by using deep learning and digital geometry processing techniques. Yet, only architectural usage is covered in the scope of this dissertation since different usage areas need distinct datasets.

Overall the objectives can be shortlisted as -

- OBJECTIVE FOR MID: To search for an appropriate deep learning architecture to automatically extract relevant information from 2D architectural drawings.
- OBJECTIVE FOR END-SEM: To generate a dataset to facilitate the 3D reconstruction process.
- OBJECTIVE FOR NEXT-SEM(i):-To explore convenient digital geometry processing algorithms for transforming low-level information gathered from the 2D dataset into a higher level in 3D.
- OBJECTIVE FOR NEXT-SEM(ii) To see the level of complexity of the reconstructed 3D model

1.3 Methodology:


Finding the DL techniques which will be used project
Extract the floor blueprint image dataset
Build the CNN model
Detection of Structural information and vector representation
Build CNN model for 3d transformation

Finding the DL technique which will be used project:

The different techniques that we would use in the project are R-CNN(Region-Based Conventional Neural Networks), FAST R-CNN, U-NET, SEGMENT, TERNAUS NET

Extract the floor blueprint image dataset:

In this, we will try to give the input as an image format that is fed into the CNN for analysis.

Build the CNN model

Generate the synthetic dataset.

Detection of Structural information and vector representation

In this section, the model determines the structural and also segments the semantic information to generate vector representation using CNN techniques.

Build CNN model for 3d transformation

Finally transforming the following vector representation into a 3D model.

1.4 Expected Result/Obtained Result:

After architectural drawings are semantically segmented with CNN architectures in a 2D pixel environment, 3D conversion becomes possible to obtain 3D reconstructed models… In this research, the vectorization of a predicted image firstly relies on morphological transformations for image enhancement. Later, enhanced images are contoured.

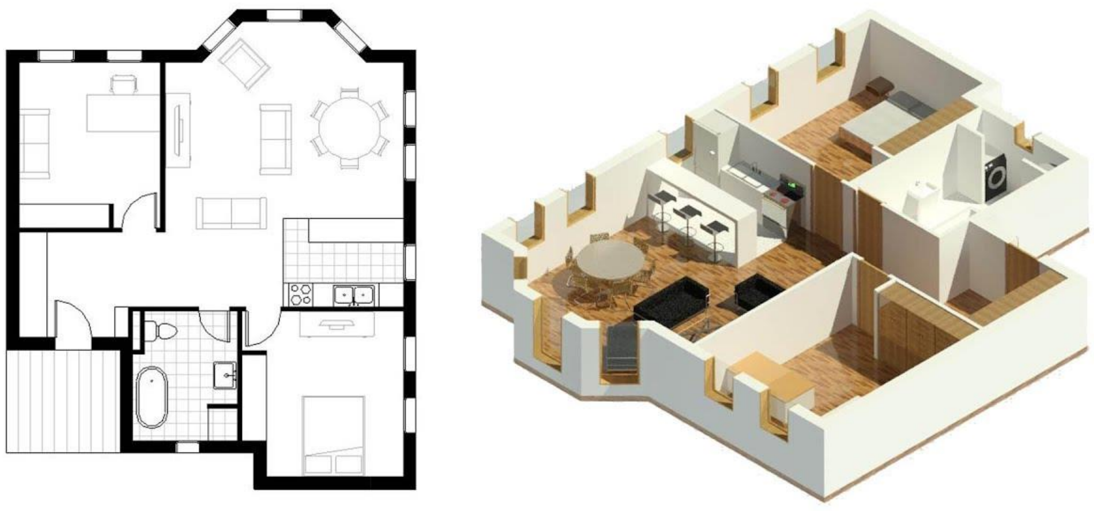


Figure (1.4) Expecting a 3d blueprint after following the vector representation

Contours include 2D point information in an image. Thus, these points provide an environment for 2D to 3D conversion.

An input floor plan image, semantic segmentation result with post-processing, reconstructed vector graphics representation with a room type annotation,and the corresponding 3D reconstruction. The different colors of the segmented and reconstructed vector graphics result represent different elements, and the palette information can be found. We evaluate the generalization ability of the system by processing floor plan images from other data sources.

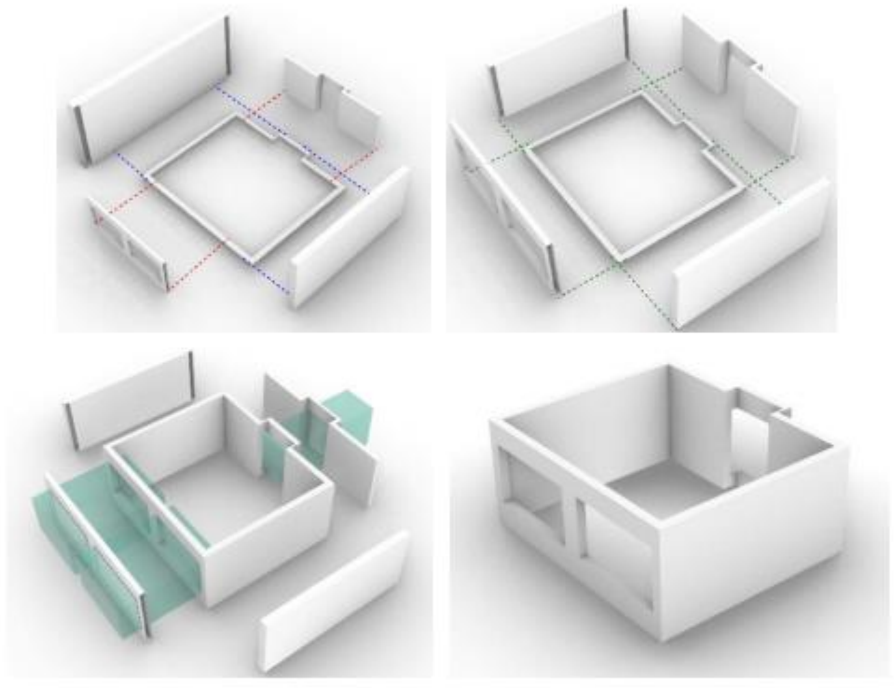


Figure (1.4 a) Expected 3D views

1.5 Time table:

7th Sem Mid-Term	• Identifying the problem statement. • Identifying Deep Learning techniques to implement. • Comparing different types of CNN techniques.
7th Sem end-sem	• Extracting the input dataset. • Using the required CNN i.e, VGG encoder for semantic segmentation and vector representation
8th Sem	• Transforming the vectored output to a 3D model

CHAPTER 2

STATE OF ARTS

2.1 Introduction

What is CNN?

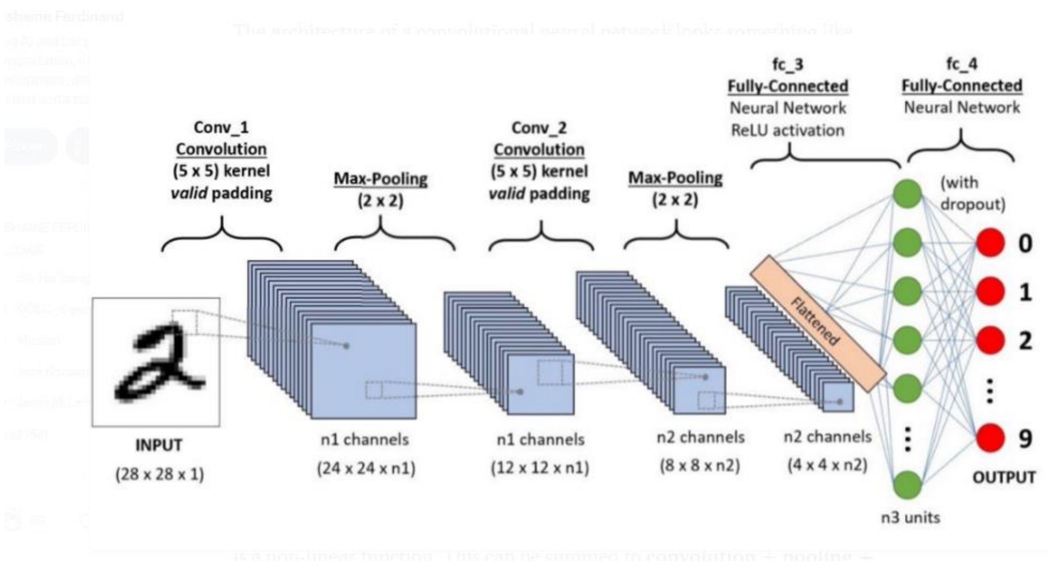
A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

Why should we use it?

So why would we use a CNN over another type of neural network, say, a multi-layer perceptron? This is because CNNs can extract the features of an image, which an algorithm like a multi-layer perceptron (MLP) or a recursive neural network (RNN) cannot do.

The architecture of a convolutional neural network looks something like this:



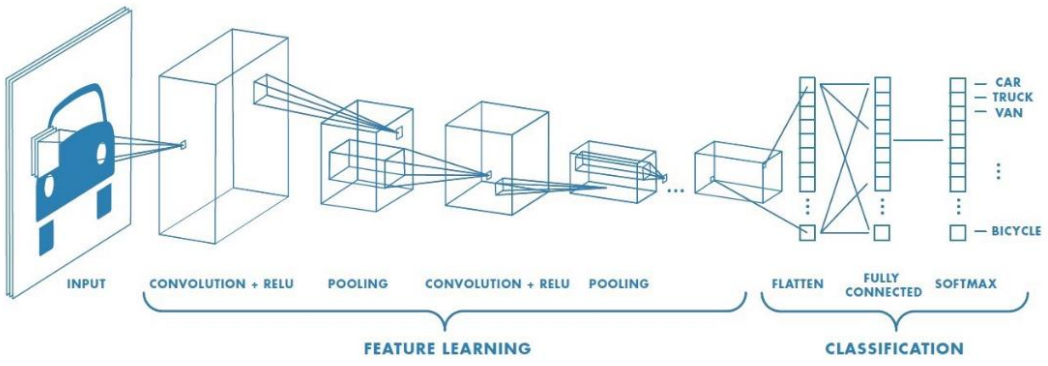


Figure (2.1) CNNs extracting the features of an image

Convolutional layer:

Within deep learning the convolution operation acts on the filters/kernels and image data array within the convolutional layer. Therefore a convolutional layer is simply a layer that houses the convolution operation that occurs between the filters and the images passed through a convolutional neural network. Batch Normalisation layer: Batch Normalization is a technique that mitigates the effect of unstable gradients within a neural network through the introduction of an additional layer that performs operations on the inputs from the previous layer. The operations standardize and normalize the input values, after that the input values are transformed through scaling and shifting operations.

MaxPooling layer:

Max pooling is a variant of sub-sampling where the maximum pixel value of pixels that fall within the receptive field of a unit within a sub-sampling layer is taken as the output. The max-pooling operation below has a window of 2x2 and slides across the input data, outputting an average of the pixels within the receptive field of the kernel.

Flatten layer:

Takes an input shape and flattens the input image data into a one-dimensional array.

Dense Layer:

A dense layer has an embedded number of arbitrary units/neurons within. Each neuron is a perceptron. The code snippet represents the Keras implementation of the AlexNet CNN architecture.

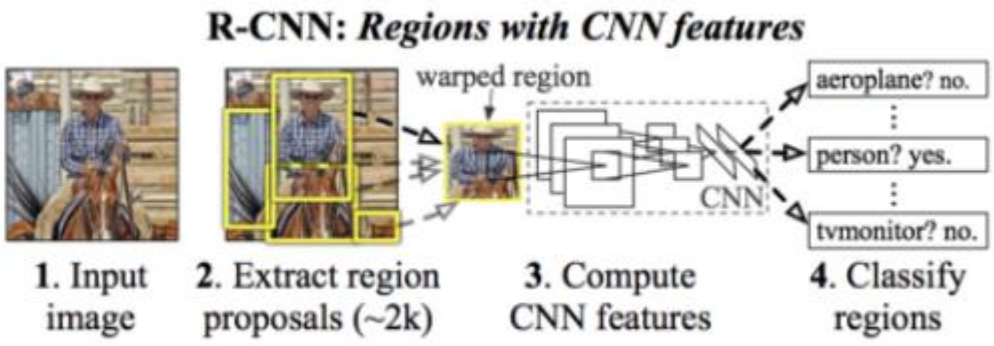
2.2 Basic background with few citations

There are various architectures of CNNs available which have been key in building algorithms that power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

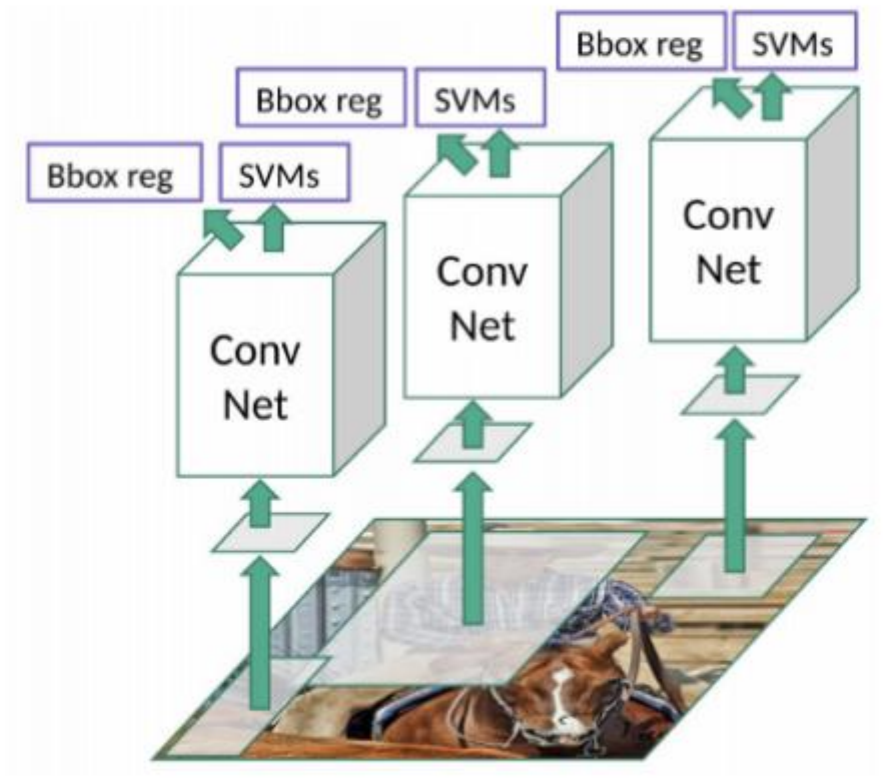
1. R – CNN
2. FAST R– CNN
3 . FASTER – CNN
4 . Mask R-CNN
5. YOLO
6. VGG-16 MODEL

R – CNN:

R-CNN, or **Regions with CNN Features**, is an object detection model that uses high-capacity CNNs to bottom-up region proposals to localize and segment objects. It uses Selective search to identify several bounding-box object region candidates (“regions of interest”), ad then extracts features from each region independently for classification.



R-CNN



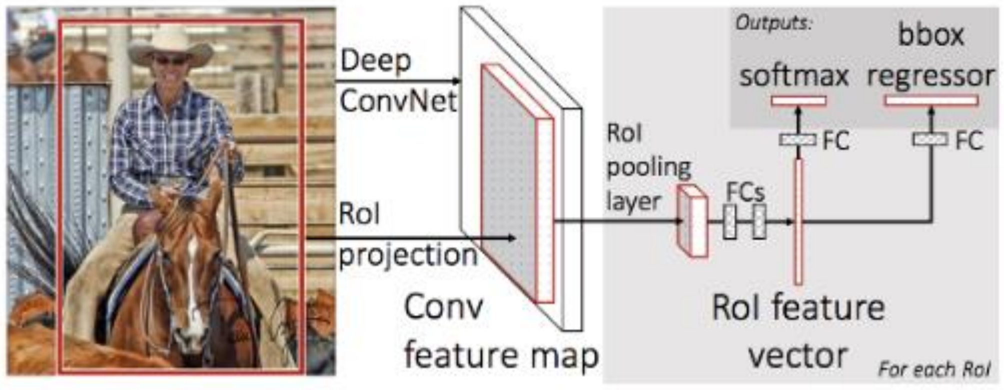
R-CNN

In short the procedure of R-CNN

•Image --> Conv (feature map) --> ROI feature vector --> bounding box regressor (SVM)

FAST R– CNN:

Fast R-CNN **evaluates the network and extracts features for the whole image once**, instead of extracting features from each RoI, which are cropped and rescaled every time in R-CNN. It then uses the concept of RoI pooling, a special case of pyramid pooling used in SPPNet, to give a feature vector of the desired length.



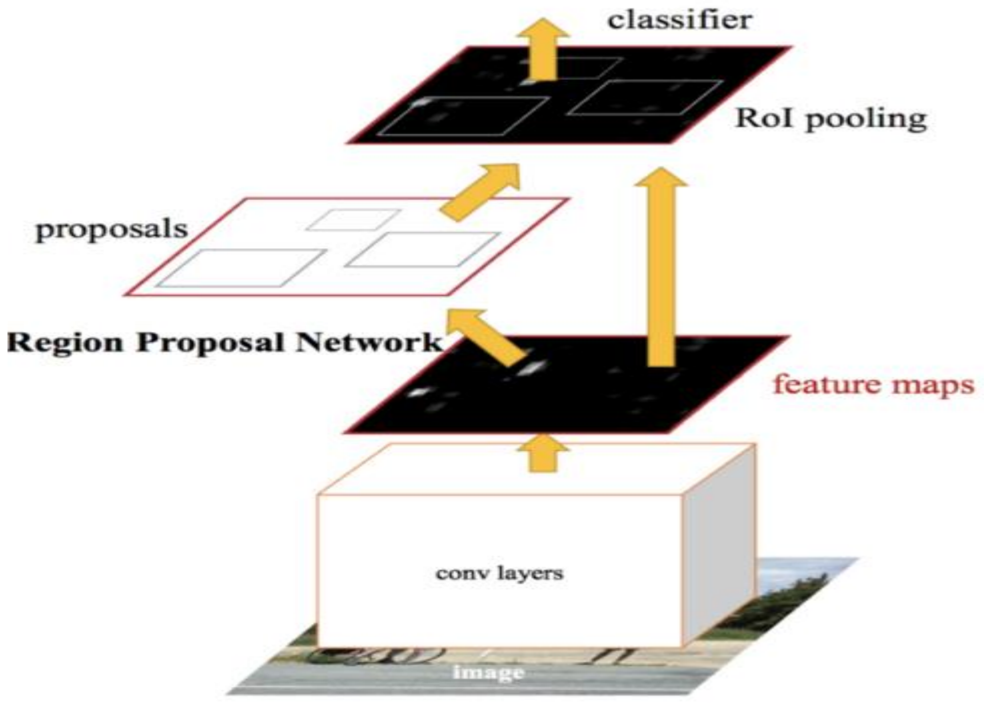
Fast R-CNN

In short the procedure of Fast R-CNN

•Image --> Conv (feature map) --> ROI feature vector --> bounding box regressor (Softmax)

FASTER R– CNN:

Faster R-CNN was introduced in 2015 by k He et al. After the Fast R-CNN, the bottleneck of the architecture is selective search. Since it needs to generate 2000 proposals per image. It constitutes a major part of the training time of the whole architecture. In Faster R-CNN, it was replaced by the region proposal network. First of all, in this network, we passed the image into the backbone network. This backbone network generates a convolution feature map. These feature maps are then passed into the region proposal network. The region proposal network takes a feature map and generates the anchors (the center of the sliding window with a unique size and scale). These anchors are then passed into the classification layer (which classifies whether there is an object or not) and the regression layer (which localizes the bounding box associated with an object).



Faster R-CNN

In short the procedure of Faster R-CNN

•Image --> convolution (feature map) --> RPN --> ROI feature vector --> bounding box regressor

Mask R-CNN:

It is very similar to Faster R-CNN except there is another layer to predict segmented. The stage of region proposal generation is the same in both the architecture and the second stage which works in parallel predict class, generates a bounding box as well as outputs a binary mask for each RoI.

In short the procedure of Mask R-CNN

•Image --> convolution (feature map) --> RPN --> ROI feature vector --> bounding box regressor and spatial layout(mask)

YOLO:

YOLO algorithm is **an algorithm based on regression**, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in one run of the Algorithm. ... Ultimately, we aim to predict a class of an object and the bounding box specifying object location

What is VGGNet?

VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network(CNN) architecture with multiple layers. The “deep” refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers.

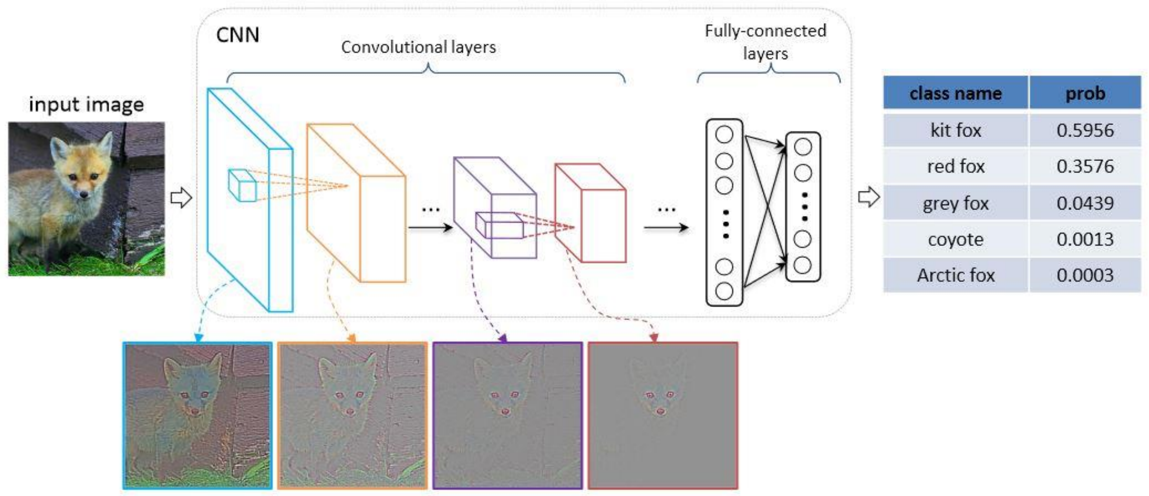
- The VGG architecture is the basis of ground-breaking object recognition models. Developed as a deep neural network, the VGGNet also surpasses baselines on many tasks and datasets beyond ImageNet. Moreover, it is now still one of the most popular image recognition architectures.

- VGG16 is used for object detection and classification algorithm which is can 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.

- The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer.

VGG Architecture:

VGGNets are based on the most essential features of convolutional neural networks (CNN). The following graphic shows the basic concept of how a CNN works:



class name	prob
kit fox	0.5956
red fox	0.3576
grey fox	0.0439
coyote	0.0013
Arctic fox	0.0003

The architecture of a Convolutional Neural Network: Image data is the input of the CNN; the model output provides prediction categories for input images.

The VGG network is constructed with very small convolutional filters. The VGG-16 consists of 13 convolutional layers and three fully connected layers.

Let’s take a brief look at the architecture of VGG:

- **Input:** The VGGNet takes in an image input size of 224×224. For the ImageNet competition, the creators of the model cropped out the center 224×224 patch in each image to keep the input size of the image consistent.
- **Convolutional Layers**: VGG’s convolutional layers leverage a minimal receptive field, i.e., 3×3, the smallest possible size that still captures up/down and left/right. Moreover, there are also 1×1 convolution filters acting as a linear transformation of the input. This is followed by a ReLU unit, which is a huge innovation from AlexNet that reduces training time. ReLU stands for rectified linear unit activation function; it is a piecewise linear function that will output the input if positive; otherwise, the output is zero. The convolution stride is fixed at 1 pixel to keep the spatial resolution preserved after convolution (stride is the number of pixel shifts over the input matrix).

- **Hidden Layers:** All the hidden layers in the VGG network use ReLU. VGG does not usually leverage Local Response Normalization (LRN) as it increases memory consumption and training time. Moreover, it makes no improvements to overall accuracy.
- **Fully-Connected Layers:** The VGGNet has three fully connected layers. Out of the three layers, the first two have 4096 channels each, and the third has 1000 channels, 1 for each class.

What is STL?

STL is a file format native to the stereolithography CAD software created by 3D Systems. STL has several backronyms such as "**S**tandard **T**riangle **L**anguage" and "**S**tandard Tessellation **L**anguage". This file format is supported by many other software packages; it is widely used for rapid prototyping, 3D printing, and computer-aided manufacturing. STL files describe only the surface geometry of a three-dimensional object without any representation of color, texture, or other common CAD model attributes.

What is Dtype?

A data type object (an instance of numpy. dtype class) describes how the bytes in the fixed-size block of memory corresponding to an array item should be interpreted. It describes the following aspects of the data: Type of the data (integer, float, Python object, etc.)
Structured data types are formed by creating a data type whose field contains other data types. Each field has a name by which it can be accessed. The parent data type should be of sufficient size to contain all its fields; the parent is nearly always based on the void type which allows an arbitrary item size. Structured data types may also contain nested structured sub-array data types in their fields.

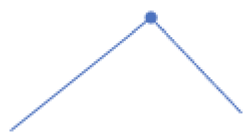
Finally, a data type can describe items that are themselves arrays of items of another data type. These sub-arrays must, however, be of a fixed size.

The main difference between **Type** and **Dtype** are:
Type is a built-in name of the Python language - it is a call that either identifies the type of an object or creates a new type. dtype is the name of a parameter/member of numerical NumPy library objects. Although numpy is very popular, it can not be confused with the language core - ntype is universal - even in NumPy it is an attribute or parameter that will only be

present where it makes sense. Dtype is the abbreviation for data type and is used in numpy and related numeric / scientific biblioitoecas to identify the type of data that will be contained in an object.

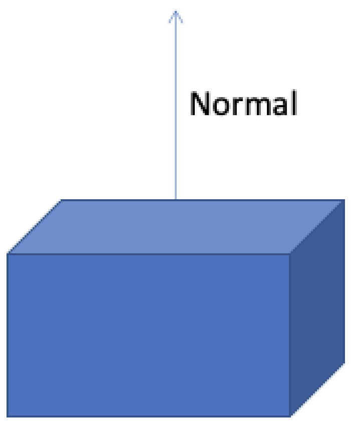
Terms we used to convert to 3d:

A **vertex** is a corner, or more precisely, a point where two lines intersect.



A **Face** is an individual flat surface, part of a solid object.

Vertex Normal is a vector that points in the direction that a face points.



Mesh is a collection of vertices, edges, and faces that describe the shape of a 3D object.

2.3 Related literature in detail:

We followed this paper as a reference “**Deep Floor Plan Recognition Using a Multi-Task Network with Room-Boundary-Guided Attention Zhiliang Zeng Xianzhi Li Ying Kin Yu Chi-Wing Fu The Chinese University of Hong Kong**”.

This paper presents a new approach to recognizing elements in floor plan layouts. Besides walls and rooms, we aim to recognize diverse floor plan elements, such as doors, windows, and different types of rooms, in the floor layouts. To this end, we model a hierarchy of floor plan elements and design a deep multi-task neural network with two tasks: one to learn to predict room-boundary elements, and the other to predict rooms with types. More importantly, we formulate the room-boundary-guided attention mechanism in our spatial contextual module to carefully take room-boundary features into account to enhance the room-type predictions. Furthermore, we design a cross-and-within-task weighted loss to balance the multi-label tasks and prepare two new datasets for floor plan recognition. Experimental results demonstrate the superiority and effectiveness of our network over the state-of-the-art methods

The objectives of this work are as follows. First, we aim to recognize various kinds of floor plan elements, which are not only limited to walls but also include doors, windows, room regions, etc. Second, we target to handle rooms of nonrectangular shapes and walls of non-uniform thickness.

USES OF VGG16 MODEL:

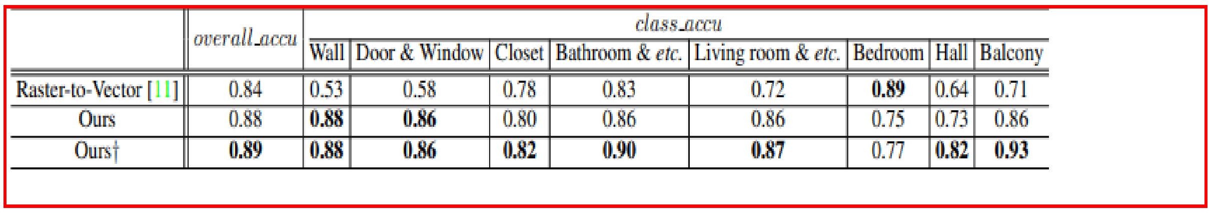
- Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2.

- VGG uses very small receptive fields instead of massive fields like AlexNet. So it uses 3×3 with a stride of 1. The decision function is more discriminative now that there are three ReLU units instead of simply one. There are also fewer parameters (27 times the number of channels vs. 49 times the number of channels in AlexNet).

- Without modifying the receptive fields, VGG uses 1×1 convolutional layers to make the decision function more non-linear.

- VGG model can have a considerable number of weight layers due to the small size of the convolution filters; of course, more layers mean better performance. However, this isn’t an unusual trait. The VGG architecture is a convolutional neural network architecture that has been around for a while. It was developed as a result of research on how to make specific networks denser. The network employs tiny 3 x 3 filters. Aside from that, the network stands out for its simplicity, with simply pooling layers and a fully linked layer as extra components. VGG net deep learning model is one of the most widely employed image-recognition models today.

Last, we also aim to recognize the rooms types in floor plans, e.g., dining room, bedroom, bathroom, etc. Achieving these goals requires the ability to process the floor plans and find multiple non-overlapping but spatially correlated elements in the plans. In our method, we first organize the floor plan elements in a hierarchy, where pixels in a floor plan can be identified as inside or outside, while the inside pixels can be further identified as room-boundary pixels or room-type pixels. Moreover, the room-boundary pixels can be walls, doors, or windows, whereas room-type pixels can be the living room, bathroom, bedroom, etc.;



	overall_accu	*class_accu*							
		Wall	Door & Window	Closet	Bathroom & *etc.*	Living room & *etc.*	Bedroom	Hall	Balcony
Raster-to-Vector [11]	0.84	0.53	0.58	0.78	0.83	0.72	**0.89**	0.64	0.71
Ours	0.88	**0.88**	**0.86**	0.80	0.86	0.86	0.75	0.73	0.86
Ours†	**0.89**	**0.88**	**0.86**	**0.82**	**0.90**	**0.87**	0.77	**0.82**	**0.93**

Table 1. Comparison with Raster-to-Vector method vs VGG16 model. Symbol † indicates our method with post-processing.

Each of the two tasks in our network involves multiple labels for various room-boundary and room-type elements. Since the number of pixels varies for different elements, we have to balance their contributions within each task. Also, there are generally more room-type pixels than room-boundary pixels, so we have to further balance the contributions of the two tasks. Therefore, we design a cross-and-within-task weighted loss to balance between the two tasks as well as among the floor plan elements within each task.

Later, after the floor plan, using Numpy STL we defined how the vertices and face look.
We first list the different vertices of the cube as a Numpy array of [X, Y, Z] coordinates.
Then we create another Numpy array with the faces using the index of each vertex defined earlier. Later using the faces and the vertices we try to create a mesh.

CHAPTER 3

DESIGN AND ANALYSIS

3.1 Introduction

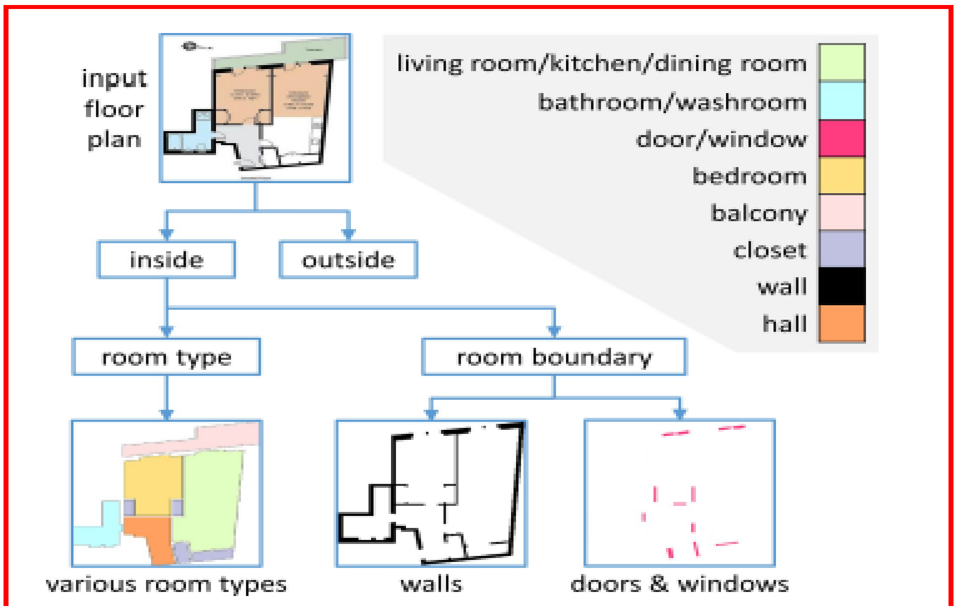


Figure (3.1). Floor plan elements are organized in a hierarchy.

To recognize floor plan elements in a layout requires learning semantic information in the floor plans. It is not merely a general segmentation problem since floor plans present not only the individual floor plan elements, such as walls, doors, windows, closets, etc., but also how the elements relate to one another, and how they are arranged to make up different types of rooms. While recognizing semantic information in floor plans is generally straightforward for humans, automatically processing floor plans and recognizing layout semantics is a very challenging problem in image understanding and document analysis.

```
!pip install matplotlib numpy OpenCV-
python pdbpp tensorboard tensorflow scipy Pillow gdown
import tensorflow as tf
import sys
sys.path.append('./TF2DeepFloorplan/')
from net import *
from data import *
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from argparse import Namespace
import os
import gc
```

```
from rgb_ind_convertor import *
from util import *
from legend import *
from deploy import *
import cv2

!pip3 install numpy-stl
from stl import mesh
from PIL import Image
import matplotlib.pyplot as plt

from mpl_toolkits import mplot3d
from PIL import Image, ImageFilter,ImageOps;
```

Matplotlib :

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

Numpy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Opencv:

OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.

Pdbpp :

pdbpp adds the "sticky" mode, so the debugger can be run in a full-screen mode, in terminal

Tensorboard :

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on the training and inference of deep neural networks.

Scipy:

sciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT,
signal and image processing, ODE solvers, and other tasks common in science and engineering.

Pillow:

Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating,
and saving many different image file formats. It is available for Windows, Mac OS X, and Linux.

gdown:

Now, this article is going to document the package gdown in Python that can be used to download Google Drive files.

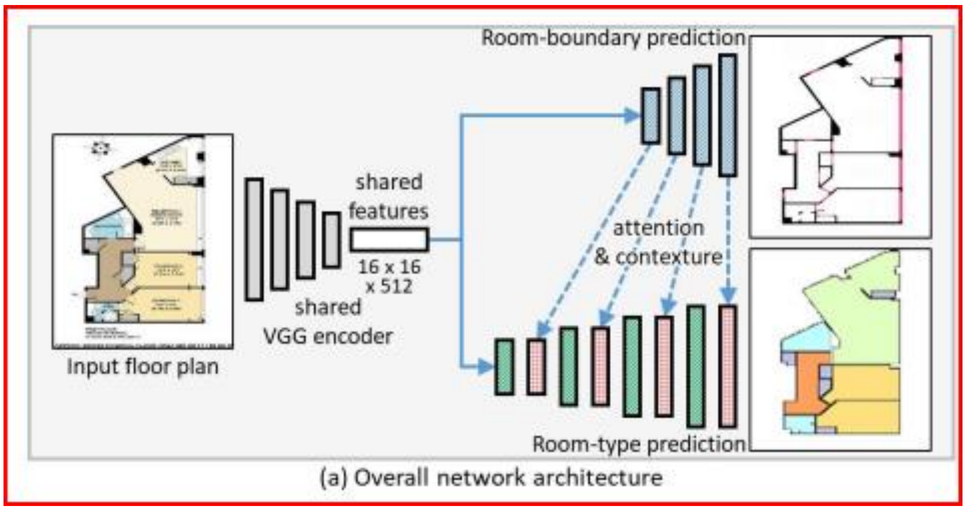
PIL:

PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities.

NumPy-stl:

A simple library to make working with STL files (and 3D objects in general) fast and easy. Due to all operations heavily relying on numpy, this is one of the fastest STL editing libraries for Python available.

3.2 System Architecture:



Figure(3a) **Schematic diagram illustrating our deep multi-task neural network.**

Overall network architecture. Figure (a) presents the overall network architecture. First, we adopt a shared VGG encoder [17] to extract features from the input floor plan image. Then, we have two main tasks in the network: one for predicting the room-boundary pixels with three labels, i.e., wall, door, and window, and the other for predicting the room-type pixels with eight labels, i.e., dining room, washroom, etc.; see Figure 2 for details. Here, room boundary refers to the floor-plan elements that separate room regions in floor plans; it is not simply low-level edges nor the outermost border that separates the foreground and background. Specifically, our network first learns the shared feature, common for both tasks, then makes use of two separate VGG decoders (see Figure 3(b) for the connections and feature dimensions) to perform the two tasks. Hence, the network can learn additional features for each task. To maximize the network learning, we further make use of the room-boundary context features to bound and guide the discovery of room regions, as well as their types; here, we design the spatial contextual module to process and pass the room-boundary features from the top decoder (see Figure 3(a)) to the bottom decoder to maximize the feature integration for room-type predictions.

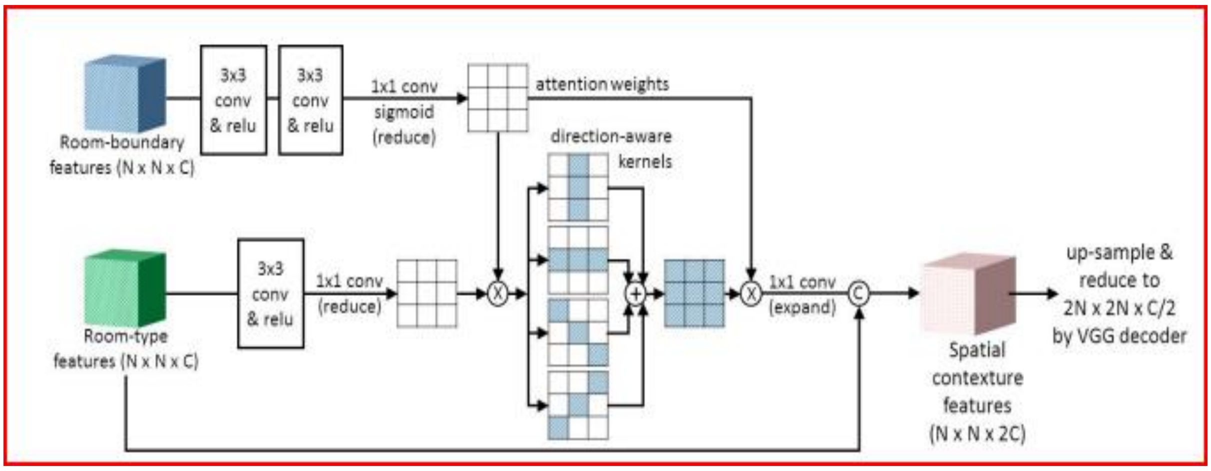

3x3 conv & relu
3x3 conv & relu
1x1 conv sigmoid (reduce)
attention weights
Room-boundary features (N x N x C)
direction-aware kernels
Room-type features (N x N x C)
3x3 conv & relu
1x1 conv (reduce)
1x1 conv (expand)
Spatial contexture features (N x N x 2C)
up-sample & reduce to 2N x 2N x C/2 by VGG decoder

Figure (4)

spatial contextual module with the room-boundary-guided attention mechanism, which leverages the room-boundary features to learn the attention weights for room-type prediction. In the lower branch, we use convolutional layers with four different direction-aware kernels to generate features for integration with the attention weights and produce the spatial contextual features

Spatial contextual module. Figure 4 shows the network architecture of the spatial contextual module. It has two branches. The input to the top branch is the room-boundary features from the top VGG decoder (see the blue boxes in Figures 3(a) & 4), while the input to the bottom branch is the room-type features from the bottom VGG decoder (see the green boxes in Figures 3(a) & 4). See again Figure 3(a): there are four levels in the VGG decoders, and the spatial contextual module (see the dashed arrows in Figure 3(a)) is applied four times, once per level, to integrate the room boundary and room-type features from the same level (i.e., in the same resolution) and generate the spatial contextual features; see the red boxes in Figures 3(a) & 4.
• In the top branch, we apply a series of convolutions to the room-boundary feature and reduce it to a 2D feature map as the attention weights, denoted as a(m,n)at pixel location m, n. The attention weights are learned through the convolutions rather than being fixed.

3.3 NOISE REDUCTION USING IMAGE FILTER:

The Image Filter module can be imported from PIL. This module contains definitions for a pre-defined set of filters, which can be used with the Image.filter() method. These filters are used to change the looks and feel of the image.
Since the images are architectural sketches, this filter can enhance a few boundaries or the shape of the room.

The type of filters which can be used are :

image.filter() method:

The current version of the pillow library provides the below-mentioned set of predefined image enhancement filters.

There are two kinds of filters available with the pillow:

Non-parameterized filters:

- BLUR
- CONTOUR
- DETAIL
- EDGE_ENHANCE
- EDGE_ENHANCE_MORE
- EMBOSS FIND_EDGES
- SHARPEN
- SMOOTH
- SMOOTH_MORE

Parameterized filters:

- Boxblur
- Gaussianblur
- Unsharpmask
- Kernel
- Rankfilter
- Medianfilter
- Minfilter
- Maxfilter

EDGE_ENHANCE Filter:

EDGE_ENHANCE filter is used to enhance the edges of an image. It convolves the below-mentioned 3x3 filter on our image in order to generated an image with enhances edges.

EDGE_ENHANCE_MORE:

EDGE_ENHANCE filter is used to enhance the edges of an image. It convolves the below-mentioned 3x3 filter on our image in order to generated an image with more enhances edges

EMBOSS Filter:

Embossing is a technique where each pixel of an image is replaced by a highlight or shadow depending on the light/dark boundaries of it. The low contrast areas of the image will become gray. A filtered image will show the rate of color change at each edge location.

SMOOTH Filter:

The SMOOTH filter is used to reduce noise in data and apply the little blur. SMOOTH filter convolves the below-mentioned 3x3 kernel on our image in order to generate a smooth image.

SHARPEN Filter:

SHARPEN filter is used to make the edges of the image sharp in order to improve its quality. It even increases the contrast between light and dark areas of the image in order to improve the features of an image. SHARPEN filter convolves the below-mentioned 3x3 kernel on our original image in order to generate a sharpened image. Latest camera devices often sharpen the images in order to improve the quality of the image.

MinFilter() method:

This method is used to create a minimum filter. It picks the lowest pixel value in a window with the given size.

for example: ImageFilter.MinFilter(size=3)

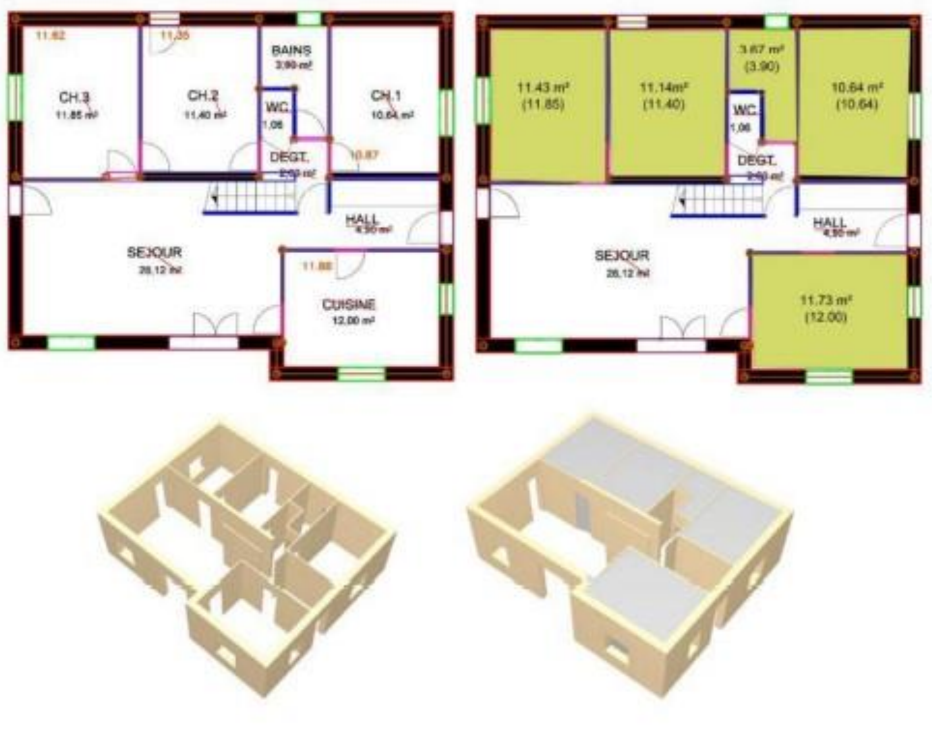
UnsharpMask Filter:

- Unsharp mask is an image sharpening method where a blurred or an unsharp negative image is used as a mask which is combined with our original image in order to generate the final image. The resulting image is generally less blurry than the original image but a less accurate representation of the original image.
- UnsharpMask filter is a parameterized filter and has the below-mentioned 3 parameters that can change the result of the filtering.
  - **radius** - It accepts integer value specifying how many neighbor pixels to consider for the filtering process. Default value if 2.
  - **percent** - It accepts integer specifying strength of unsharp in percentage. The default value is 150.

- **threshold** - It accepts integer specifying minimum brightness change that will be sharpened. The default value is 3.

3.4 Converting it to 3-d:

The output of the 2d-image is converted to grayscale. Since the resize is already done during the process of 2D recognition, we use the pixel intensity of the grayscale image to determine the thickness of each point in the 3D model, thereby creating a 3D model of a photo. We will be looking into what are the names of the fields of the structure, by which they can be accessed. Later, what is the data type of each field, and which part of the memory block each field takes i.e. Little Endian? Using the data types we got we will create a mesh, and combining all the meshes, we get our final 3D output.



To view this image we can use Blender or any such software, but we have used Microsoft 3D viewer to see the final output of the 2D given input image.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Results

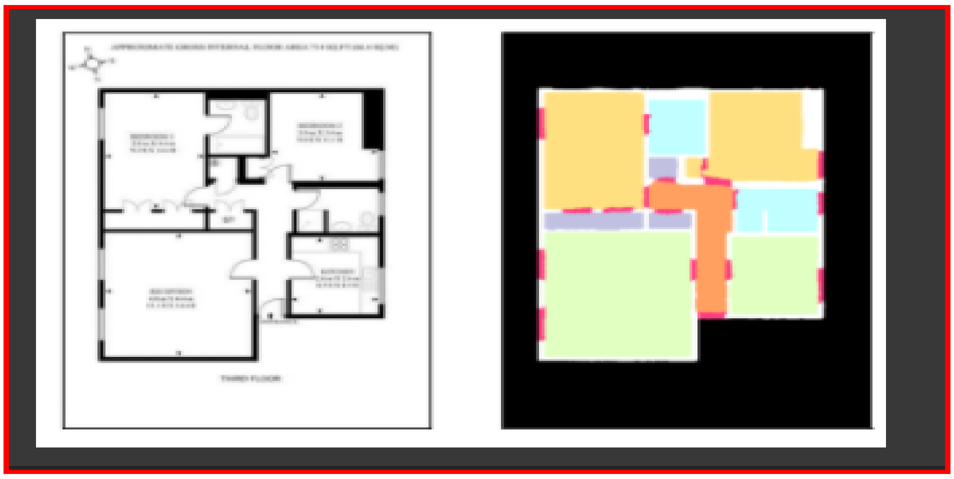
There are three key contributions to this work. First, we explore the spatial relationship between floor plan elements, model a hierarchy of floor plan elements, and design a multi-task network to learn to recognize room-boundary and room-type elements in floor plans. Second, we further take the room-boundary features to guide the room-type prediction by formulating the spatial contextual module with the room-boundary-guided attention mechanism. Further, we design a cross-and-within-task weighted loss to balance the losses within each task and across tasks. In the end, we prepared also two datasets for floor plan recognition and extensively evaluated our network in various aspects. Results show the superiority of our network over the others in terms of the overall accuracy and Fβ metrics. In the future, we plan to further extract the dimension information in the floor plan images and learn to recognize the text labels and symbols in floor plans.

The outputs of these images are later taken and fed into the Numpy created STL to create a bunch of meshes, to finally get the image into a 3D output image.

4.2 Results in a sequence of analysis of chapter 3

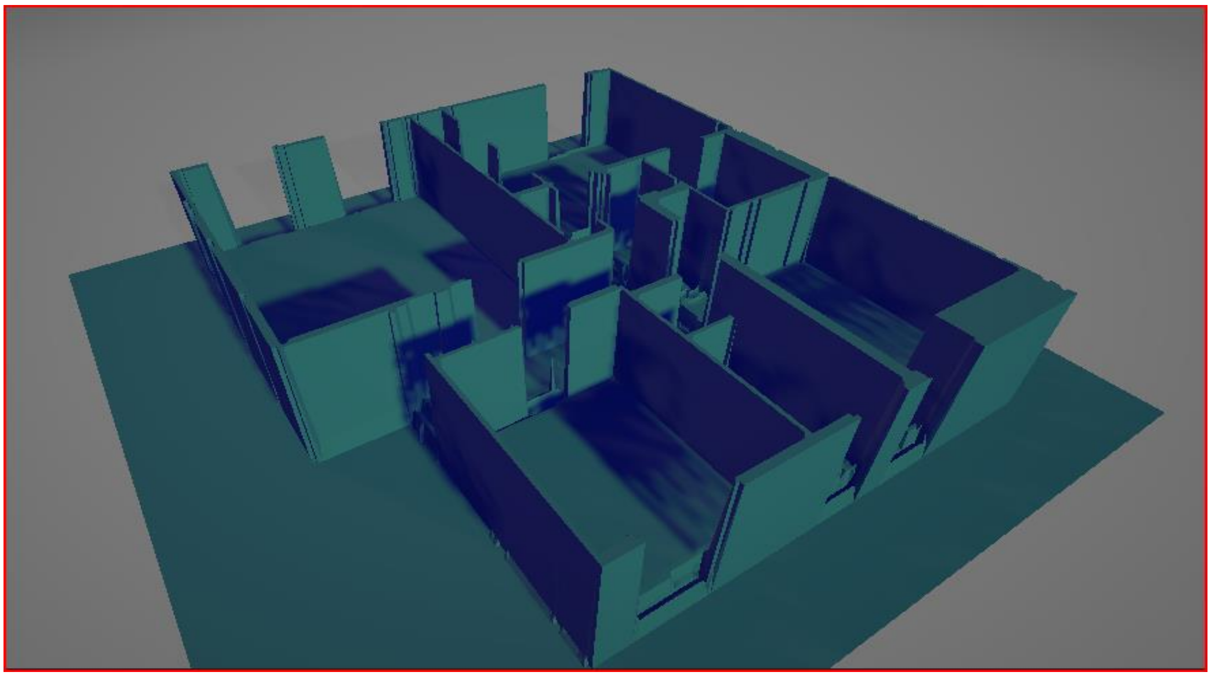
INPUT 1 AND STANDARD OUTPUT :

We have taken a 2D blueprint of certain dimensions and then considered the semantic information & Spatial analysis, The 2D image is iterated consecutively with respective pre-processings and final output is achieved . From the below result, we can see that model achieves higher accuracy, and post-processing further improved the performance.

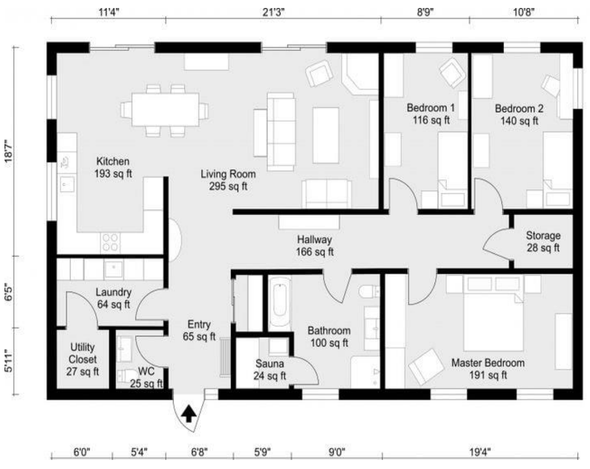


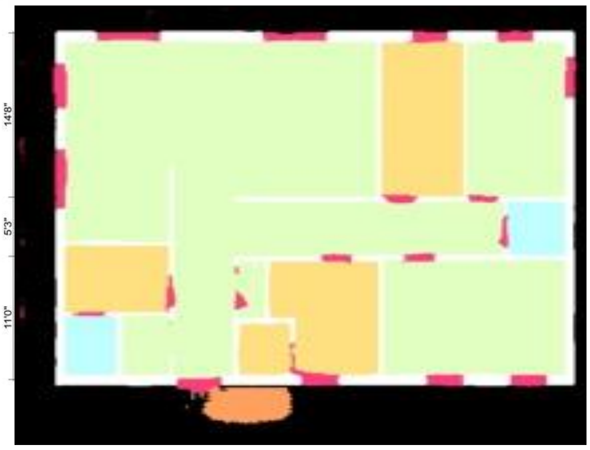
Later the output of the above image is taken as input to convert it into a higher dimension i.e,

3-D MODEL



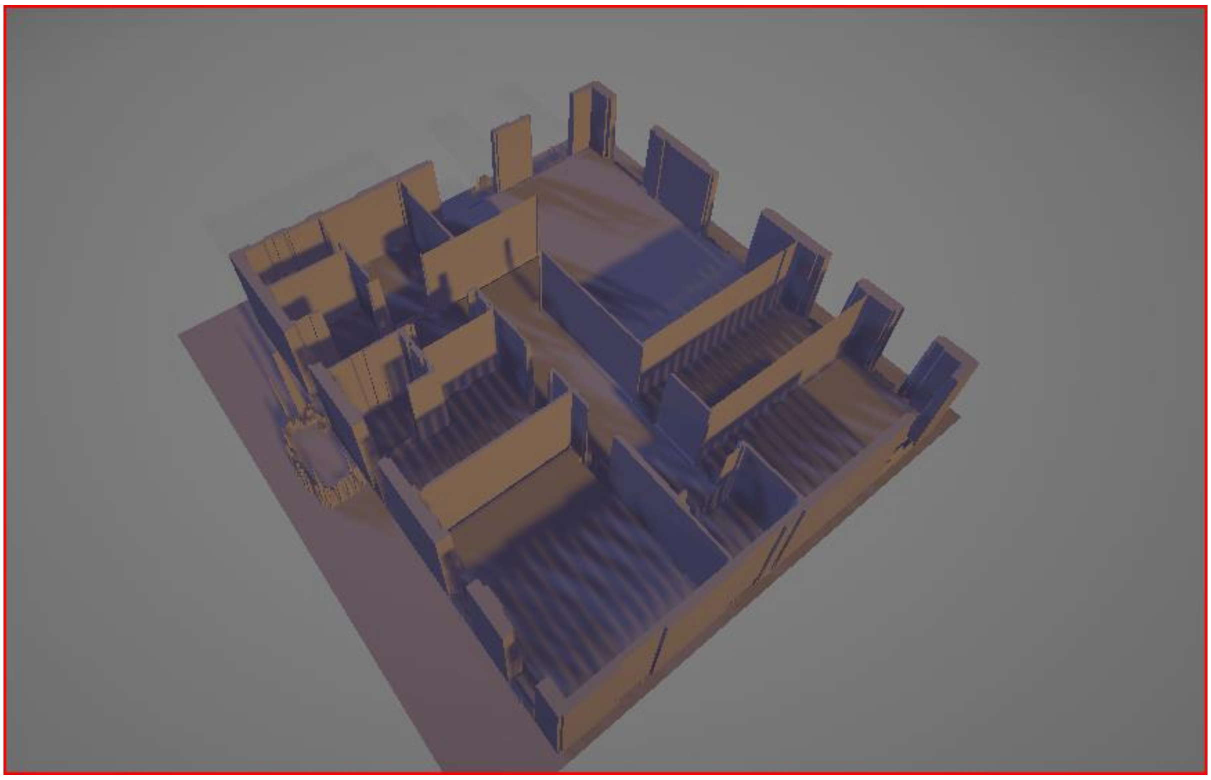
INPUT 2 AND STANDARD OUTPUT





Taking the above output image as an input image and making a 3D output :

3-D MODEL



CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion:

The Model for recognizing floor plan elements is presented here. There are three key contributions to this work. First, we explore the spatial relationship between floor plan elements, model a hierarchy of floor plan elements, and design a multi-task network to learn to recognize room-boundary and room-type elements in floor plans. Second, we further take the room-boundary features to guide the room-type prediction by formulating the spatial contextual module with the room-boundary-guided attention mechanism. Further, we design a cross-and-within-task weighted loss to balance the losses within each task and across tasks.

The 2d output contains some noise in the figure since the model does the prediction from pixel to pixel, it is likely to mishap the boundary walls. However, we adopted a image filter from the python package 'PIL' and enhanced the 2d output. So we can reduce the noise in order to get a well structured 3d model. This output is taken as an input to create meshes that are integrated to make the final 3D output.

5.2 Future scope

Further segmented colors should be represented in the model mesh, to distinguish between the rooms in the model.