MASTER OF TECHNOLOGY (INTELLIGENT SYSTEMS)

PROJECT REPORT (CA1)

Tutor: Dr Tan Jen Hong

Furniture image instance segmentation based on Mask-RCNN

TEAM MEMBERS

YE CHANGHE
ZHANG HAIHAN
ZHANG LE

Contents

EXECUTIVE SUMMARY

Nowadays, there are many places prepared for people's rest, take a break and discuss. People are not only work in the office but also in some open places, such as a well-designed park. Therefore, we need to calculate how many empty seats are there for people to sit. In some place, seats are not only the wooden ones, but also a sofa, and different shape and material. Therefore, we need some object detection method to detect empty chairs to be used. Furthermore, with the advancement in technology, image searching technology allows users to search empty in more location such as coffee, park, rest corner, shopping center, we can have a monitor on searching empty chair and count the amount.

Instance segmentation can be separated into two parts: object detection and semantic segmentation. The first task is to classify individual objects and localize each object using a bounding box, and the second task is to classify each pixel into a fixed set of categories without differentiating object instances [1]. A mask-region-based convolutional neural network (Mask R-CNN) is a recently developed DL algorithm that can deal with the instance segmentation task (He et al., 2017). This method efficiently detects the objects in an image while simultaneously generating a high-quality segmentation mask for each instance.

In this project we mainly explored the use of Mask-RCNN to segment 2 types of furniture (sofa and chair) from pictures based on a self-build dataset and tune the model by serval sets of hyper-parameters to get the best performance. Furthermore, the transfer learning approach will be applied to load the COCO weight file so that the model can be trained quickly and hit higher accuracy. The self-build dataset contains more than 300 images collected from IKEA store and internet; totally contain more than 200 instances in each category. The GitHub repository address is given here which contains the training dataset with the training & testing script and with the required library python file imported.

https://github.com/ychcnm/IRS-RTAVS-2020-04-11-ISY5004-GRP-2Z1Y_MaskRCNN

PROJECT APPROACH

1.1) PROJECT OBJECTIVE

For this project, we are going to build our own COCO format dataset and develop a deep learning model by using Mask-RCNN to do instance segmentation. Finally, the model should be capable of recognizing sofa and chair with mask and bonding box presented in the given images.

1.2) DATA UNDERSTANDING

This dataset totally contains 310 furniture images for sofas and chairs, more than 200 instances in each category (see in Figure1). Around 40% of the images are shoot at IKEA Alexandra and 60% obtained from the Web via the Bing Search API (see in Figure2). The maximum image and batch sizes are predetermined, and the searching is performed in batch. After which, a link set is returned from the API and a looping algorithm is applied to the link set to download the images. Each image might contains more than one type of items.

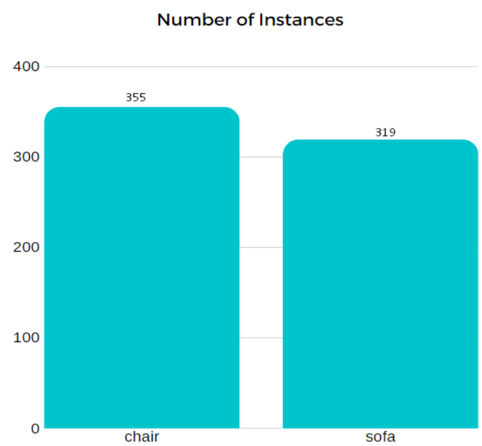

Figure 1 Number of Instances.

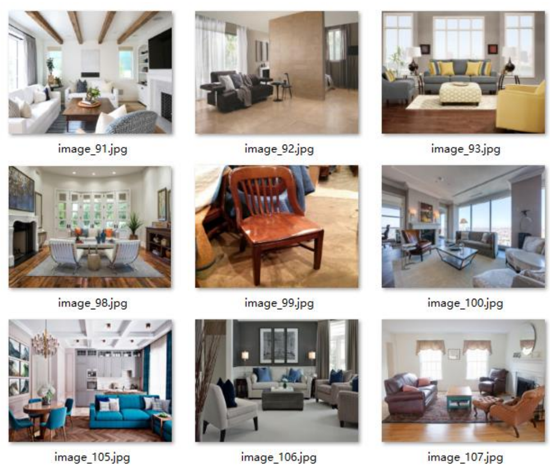

Figure 2 Image Example.

The dataset images were labeled with a data annotation tool called Colaber. There will be a json file for each image to indicate the instance category and mask location. A script was written to transform the label json file into COCO dataset format. COCO is a large-scale object detection, segmentation, and captioning dataset. COCO dataset including not only bonding box annotation information, but also segmentation (which is shown as polygon) annotation.

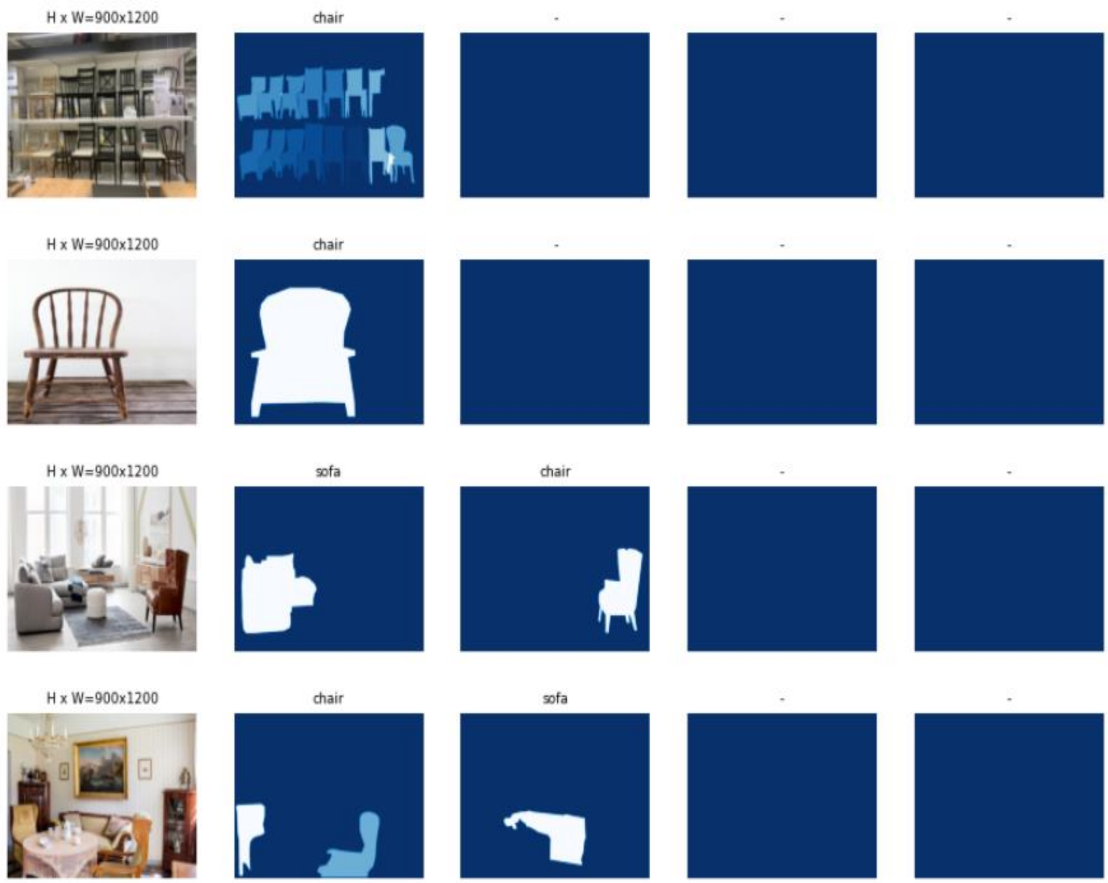

Figure 3 Image Example with Mask.

There is no need to perform exploratory data analysis to understand the important features in the data since they are automatically extracted from the first stage of Mask-RCNN.

1.3) DATA PREPARATION

After obtained all the raw images the OpenCV Library is used to manipulate the images. As the queried images are of various sizes, the first step is to resize the image to 1200 * 900 pixels so that it is much easier to use annotation tool to label.

The image data are partitioned into training and testing data. For data validation, 30% of total images from each class are selected randomly for testing such that the testing and training data have the same class distribution (see in Figure3). for the making of this model. After building the Mask-RCNN model, the training data will be used to train the model, and for the final verification and testing metrics we will use the testing data.

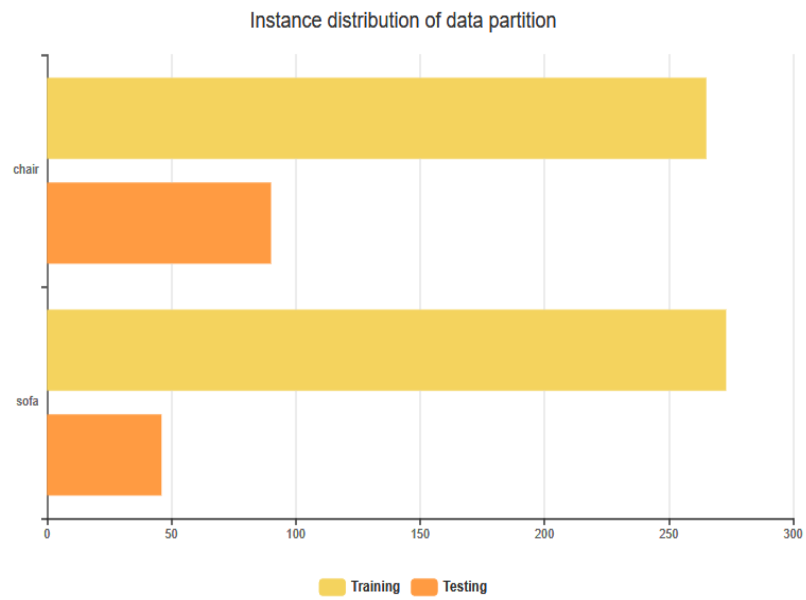

Figure 4 Instance Distribution of Data Partition

1.4) MODELING

In this model, we used Mask-RCNN (see in Figure 5) as the main analysis method to apply object segmentation on our dataset.

Mask R-CNN is an instance segmentation algorithm that can be used to apply object detection, instance segmentation and key point detection.

It is developed based on Faster-RCNN. Compared with Faster-RCNN, Mask-RCNN added Mask branch (FCN) to help generate object masks and changed the RoI pooling to RoI Align to solve the issue that the mask may not aligned with the object in the original image. Even though Mask-RCNN has a more complicated structure than Faster-RCNN, it still can achieve almost the same speed with the latter.

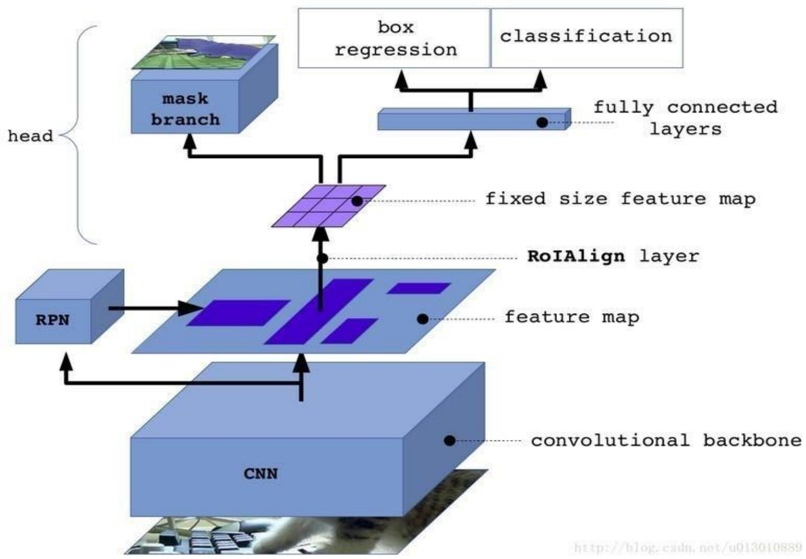

Figure 5 Structure of Mask-RCNN

Two stages be applied in Mask-RCNN approach:

1. Region proposal network (RPN) to proposes candidate object bounding boxes.

In this stage, original images are passed through a convolutional network (usually ResNet or VGG) which help to extracts features from raw images. Region Proposal Network (RPN) (see in Figure 6) uses a convolutional network to generate the Region of Interest (RoI) using a lightweight binary classifier base on 9 anchor boxes on each pixel generated from another CNN. The classifier returns object/no-object scores. Then Non-Max suppression is applied to anchors with high object score to decide the output ROI.

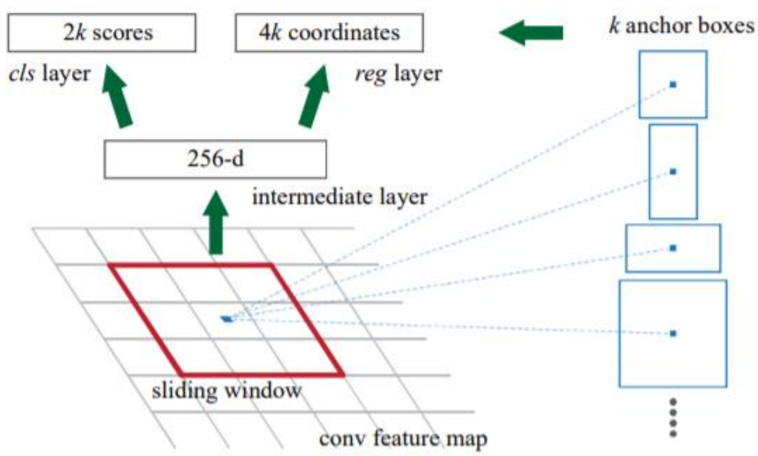

Figure 6 Structure of RPN [2]

2. Binary mask classifier to generate mask for every class

At the second stage, RoIAlign network help to warp the ROI in feature map into a fixed dimension. Then these warped feature maps are fed into fully connected layers to make classification using softmax and boundary box prediction is further refined using the regression model. These feature maps are also fed into a mask branch, which consists of two CNN's to output a binary mask for each RoI.

Mask-RCNN has 3 outputs: For each candidate object, a class label and a bounding-box offset and object mask (Figure 7)

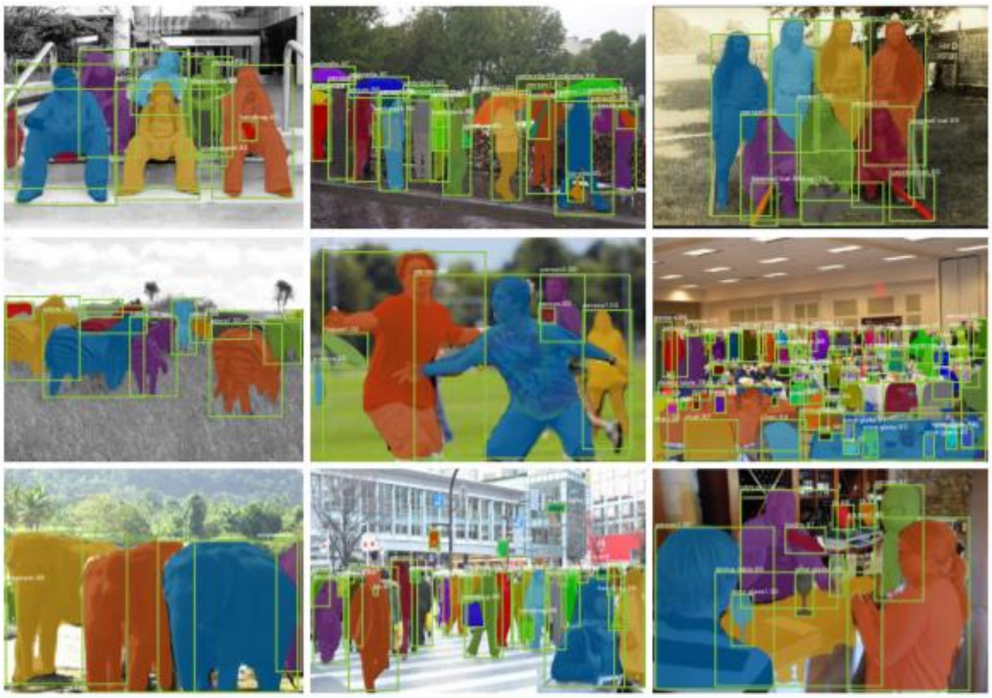

Figure 7 Expected output of Mask-RCNN [3]

CODE DESCRIPTION AND API DESIGN

The whole project consists of two scripts: furniture.py which serves as the library to be imported and Final_tunning.ipynb the Jupyter notebook recorded the training & testing steps of the segmentation model.

1.4) furniture.py

This is the library file which contains 5 APIs please see the description below:

```
####################################################################
#  Configurations
####################################################################


class FurnitureConfig(Config):...


####################################################################
#  Dataset
####################################################################
class FurnitureDataset(utils.Dataset):

    def load_data(self, annotation_json, images_dir):...

    def load_mask(self, image_id):...

    def display_mask(self):...

    def evaluate_mAP(self, model, config, nums):...

####################################################################
#  Inference
####################################################################
def inference(model, real_test_dir, dataset):...
```

Figure 8 Code of furniture.py

Class **FurnitureConfig**:

```
"""Configuration for training on Furniture COCO.
    Derives from the base Config class and overrides values specific
    to the Furniture COCO dataset.

"""
```

Funcation **load_data**:

```
""" Load the coco-like dataset from json
    Args:
        annotation_json: The path to the coco annotations json file
        images_dir: The directory holding the images referred to by the json file

"""
```

Funcation **load_mask**:

```
""" Load instance masks for the given image.
    MaskRCNN expects masks in the form of a bitmap [height, width, instances].
    Args:
        image_id: The id of the image to load masks for
    Returns:
        masks: A bool array of shape [height, width, instance count] with
            one mask per instance.
        class_ids: a 1D array of class IDs of the instance masks.

"""
```

Function **display_mask**:

```
""" Display the image with mask randomly inside the dataset.

"""
```

Function **evaluate_mAP**:

```
""" Randomly choose n images and calculate the overall accuracy based on given model and configuration
    Args:
        model: The model to calculate the overall accuracy
        config: The model configuration when doing inference
        nums: The number of images want to test
    Returns:
        mAP: the mean of the accuracy after test n images

"""
```

Function **inference**:

```
""" Get the inference result for all images inside the real_test_dir
        Args:
            model: The model to do inference
            real_test_dir: The directory holding the images for inference
            dataset: The dataset object

"""
```

1.5) fine_tuning.ipynb

This is the Jupyter notebook for training the fine-tuning model and run the inference on test images.

1. Pre-loading
   a) Import library

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import os
import sys
import time
import imgaug
import skimage
import imgaug.augmenters as aug
import mrcnn.model as modellib
from mrcnn import utils
from tensorflow.keras.callbacks import TensorBoard

import furniture as ft
```

```
Using TensorFlow backend.
```

b) Path defines

In [2]:
```
# Root directory of the project
ROOT_DIR = os.path.abspath("./")

# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library

# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# For TensorBoard
NAME = 'Mask-RCNN-furniture-{}'.format(int(time.time()))
tensorboard = TensorBoard(log_dir='logs/{}'.format(NAME))

# Local path to trained weights file
COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")
# Which weights to start with?
init_with = "coco"  # imagenet, coco, or last
if not os.path.exists(COCO_MODEL_PATH):
    utils.download_trained_weights(COCO_MODEL_PATH)

DATA_DIR = os.path.join(ROOT_DIR, "dataset")

real_test_dir = './dataset/real_test/'
```

c) Configuration

In [3]:
```
class FinalConfig(ft.FurnitureConfig):

    IMAGE_MAX_DIM = 256
    IMAGE_MIN_DIM = 256

    POST_NMS_ROIS_TRAINING = 10

    DETECTION_MIN_CONFIDENCE = 0.8

    BACKBONE = 'resnet50'


config = FinalConfig()
config.display()
```

2. Data loading and display

a) Load data

In [4]:
```
dataset_train = ft.FurnitureDataset()
dataset_train.load_data('./dataset/train/coco.json', './dataset/train/')
dataset_train.prepare()

dataset_val = ft.FurnitureDataset()
dataset_val.load_data('./dataset/val/coco.json', './dataset/val/')
dataset_val.prepare()
```

b) Data augmentation

In [5]:
```
augmentation = imgaug.augmenters.Sometimes(1 / 2, aug.OneOf(
    [
        imgaug.augmenters.Fliplr(1),
        imgaug.augmenters.Flipud(1),
        imgaug.augmenters.Affine(rotate=(-45, 45)),
        imgaug.augmenters.Affine(rotate=(-90, 90)),
        imgaug.augmenters.Affine(scale=(0.5, 1.5))
    ]
))
```

c) Display image

In [6]:
```
display = dataset_train
display.display_mask()
```

3. Model Creation

a) Create the model

In [7]:
```
# Create model in training mode
model = modellib.MaskRCNN(mode="training", config=config,
                          model_dir=MODEL_DIR)
```

b) Load the pre-train weight

In [8]:
```
if init_with == "imagenet":
    model.load_weights(model.get_imagenet_weights(), by_name=True)
elif init_with == "coco":
    # Load weights trained on MS COCO, but skip layers that
    # are different due to the different number of classes
    # See README for instructions to download the COCO weights
    model.load_weights(COCO_MODEL_PATH, by_name=True,
                       exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
                                "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":
    # Load the last model you trained and continue training
    model.load_weights(model.find_last(), by_name=True)
```

c) Train model

In [ ]:
```
# Fine tune all layers
# Passing layers="all" trains all layers. You can also
# pass a regular expression to select which layers to
# train by name pattern.
start_train = time.time()

model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=200,
            layers="all",
            custom_callbacks=[tensorboard],
            augmentation=augmentation)

end_train = time.time()
minutes = round((end_train - start_train) / 60, 2)
print(f'Training took {minutes} minutes')
```

4. Inference and evaluation

a) Configuration

In [ ]:
```
class InferenceConfig(ft.FurnitureConfig):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
    IMAGE_MIN_DIM = 256
    IMAGE_MAX_DIM = 256
    DETECTION_MIN_CONFIDENCE = 0.7
    POST_NMS_ROIS_TRAINING = 10

inference_config = InferenceConfig()
```

b) Create model and load weight from last training

In [ ]:
```
# Recreate the model in inference mode
inference_model = modellib.MaskRCNN(mode="inference",
                          config=inference_config,
                          model_dir=MODEL_DIR)
```

In [ ]:
```
# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = os.path.join(ROOT_DIR, ".h5 file name here")
model_path = inference_model.find_last()

# Load trained weights (fill in path to trained weights here)
assert model_path != "", "Provide path to trained weights"
print("Loading weights from ", model_path)
inference_model.load_weights(model_path, by_name=True)
```

c) Inference on testing image

In [ ]:
```
ft.inference(inference_model, real_test_dir, dataset_val)
```

d) Evaluation

In [ ]:
```
accuracy = dataset_val.evaluate_mAP(inference_model, inference_config, 100)
print("The overall accuracy of the model is : {}".format(accuracy))
```

MODEL TUNING

There are several hyper-parameters involved in Mask R-CNN which are to be tuned carefully based on the application. Thus, in this section we are mainly exploring how would each of the hyper-parameter impact the model accuracy. In purpose of get the training result quickly and hit higher accuracy Here we use the transfer learning method to load the COCO weight which is a pre-trained model weight based on large dataset and choose the “head” layer from the architecture during training so that we can get the result fast and the fine tuning model would base on this experiment result.

3.1) Backbone

The Backbone is the ConvNet architecture that is to be used in the first step of Mask R-CNN which will help to extract the feature map from input images. In our model, we choose ResNet50 and ResNet101 to test the performance.

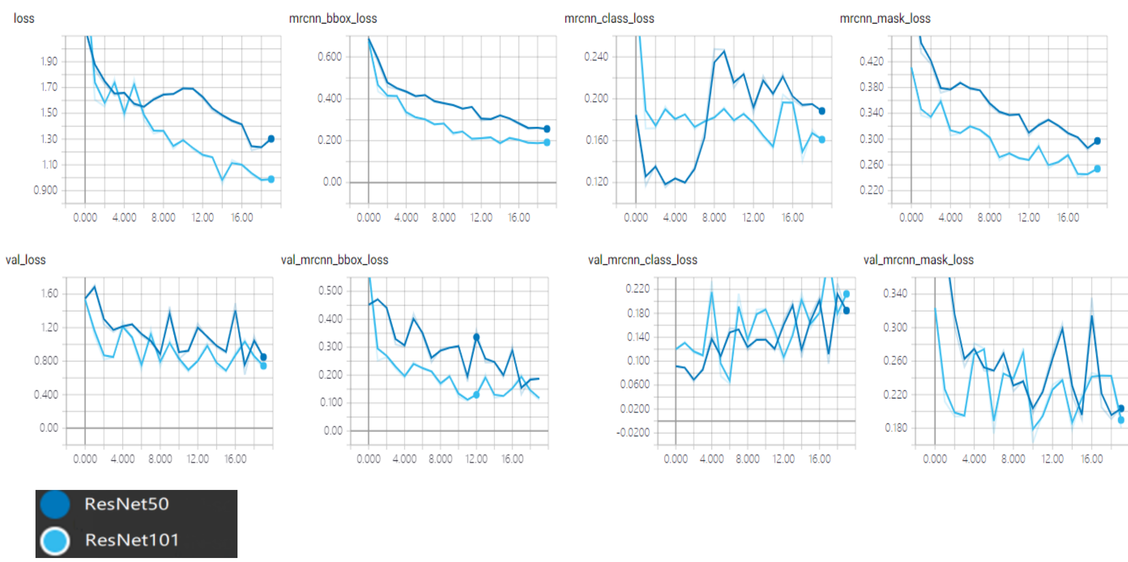

Figure 9 Training Result for Different Backbone.

	RESNET50	RESNET101
LOSS	1.302	0.9884
MRCNN_BBOX_LOSS	0.2552	0.1925
MRCNN_CLASS_LOSS	0.8139	0.7447
MRCNN_MASK_LOSS	0.2992	0.2554
VAL_LOSS	0.8139	0.7239
VAL_MRCNN_BBOX_LOSS	0.1882	0.1125
VAL_MRCNN_CLASS_LOSS	0.1797	0.2182
VAL_MRCNN_MASK_LOSS	0.2049	0.1805
DURATION	25m 50s	30m 5s

Table 1 Training Result for Different Backbone.

From the loss output (Figure 9) we know that ResNet101 got a very big improvement on accuracy (smaller loss) because of the number of layers, but ResNet50 took relatively lesser time. If there are no pre-trained weights involved and basic parameters like learning rate and number of epochs are well tuned, ResNet101 might be a better choice but since we used coco pretrained weight as initialization weight, ResNet50 would work faster and better on our model.

3.2) Train_ROIs_Per_Image

This is the maximum number of ROI’s, The Region Proposal Network will generate for the image, which will further be processed for classification and masking in the next stage. If the number of instances is limited, it can be reduced to reduce the training time.

Figure 10 Training Result for Different ROI Value.

	ROI = 10	ROI = 100	ROI = 1000
LOSS	1.131	1.155	1.314
MRCNN_BBOX_LOSS	0.2919	0.2576	0.2552
MRCNN_CLASS_LOSS	0.06883	0.1434	0.1872
MRCNN_MASK_LOSS	0.2922	0.3166	0.2992
VAL_LOSS	0.5786	1.106	0.8139
VAL_MRCNN_BBOX_LOSS	0.1282	0.2353	0.1882
VAL_MRCNN_CLASS_LOSS	0.05029	0.2442	0.1797
VAL_MRCNN_MASK_LOSS	0.1441	0.3047	0.2049
DURATION	21m 3s	19m 26s	25m 50s

Table 2 Training Result for Different ROI Values.

From the loss output (Figure 10) as we reduced the ROI value to 10 the overall loses all reduced since the furniture images won’t contain to many instances and as we limit the region of the interest, the model will focus on less instances so that require shorter training time and perform better.

3.3) Image_Min_Dim and Image_Max_Dim

The default settings resize images to squares of size 1024x1024. Smaller images can be used to reduce memory requirements and training time. The ideal approach would be to train all the initial models on smaller image sizes for faster updating of weights and use higher sizes during final stage to fine tune the final model parameters.

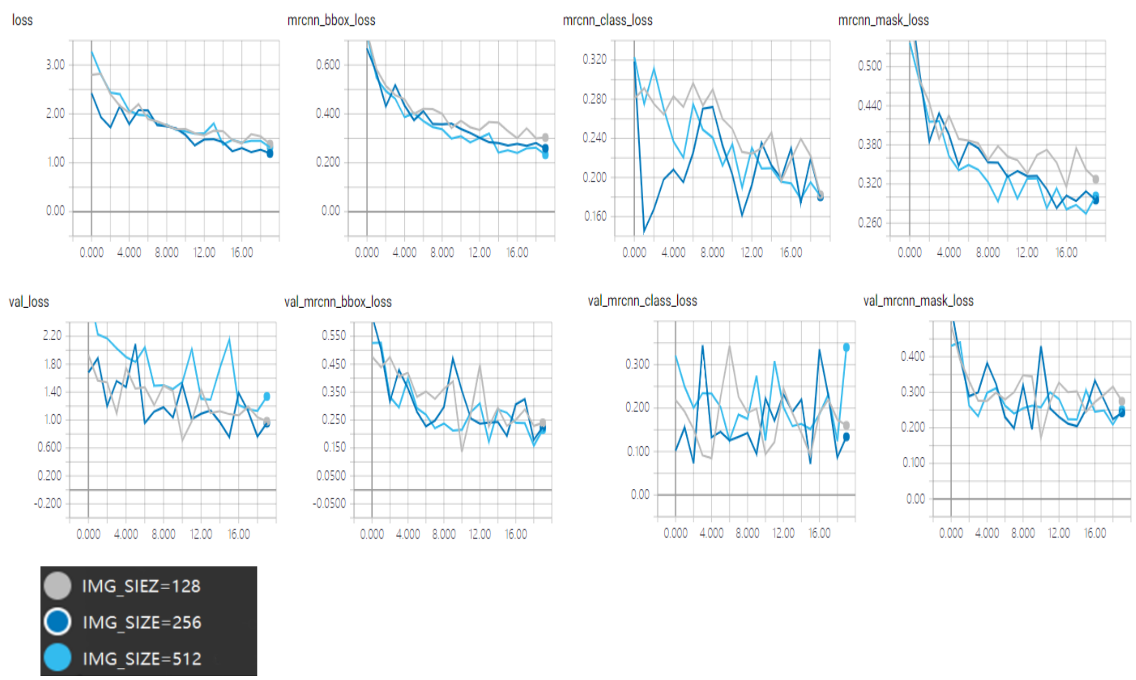

Figure 11 Training Result for Different Image Size Value.

	IMG_SIZE = 128	IMG_SIZE = 256	IMG_SIZE = 512
LOSS	1.382	1.191	1.319
MRCNN_BBOX_LOSS	0.3032	0.2590	0.2329
MRCNN_CLASS_LOSS	0.1824	0.18002	0.1810
MRCNN_MASK_LOSS	0.3273	0.2590	0.2329
VAL_LOSS	0.9827	0.9557	1.338
VAL_MRCNN_BBOX_LOSS	0.2402	0.2252	0.2157
VAL_MRCNN_CLASS_LOSS	0.1601	0.1340	0.3399
VAL_MRCNN_MASK_LOSS	0.2747	0.2252	0.2157
DURATION	24m 37s	30m 7s	39m 16s

Table 3 Training Result for Different Image Size Value.

From the output it is basically follow the assumption that has higher image resolution can compromise better accuracy overall and need longer training time.

However, when we continue to increase the image max size and min size to 512, the loss of all 4 metrics in training data and 3 of 4 metrics in validation data increased, seems the best size setting in this certain context would be between 128 and 256.

3.4) Detection_Min_Confidence

Detection confidence is the confidence level threshold, beyond which the classification of an instance will happen. Initialization can be at default and reduced or increased based on the number of instances that are detected in the model. If detection of everything is important and false positives are fine, reduce the threshold to identify every possible instance. If accuracy of detection is important, increase the threshold to ensure that there are minimal false positive by guaranteeing that the model predicts only the instances with very high confidence

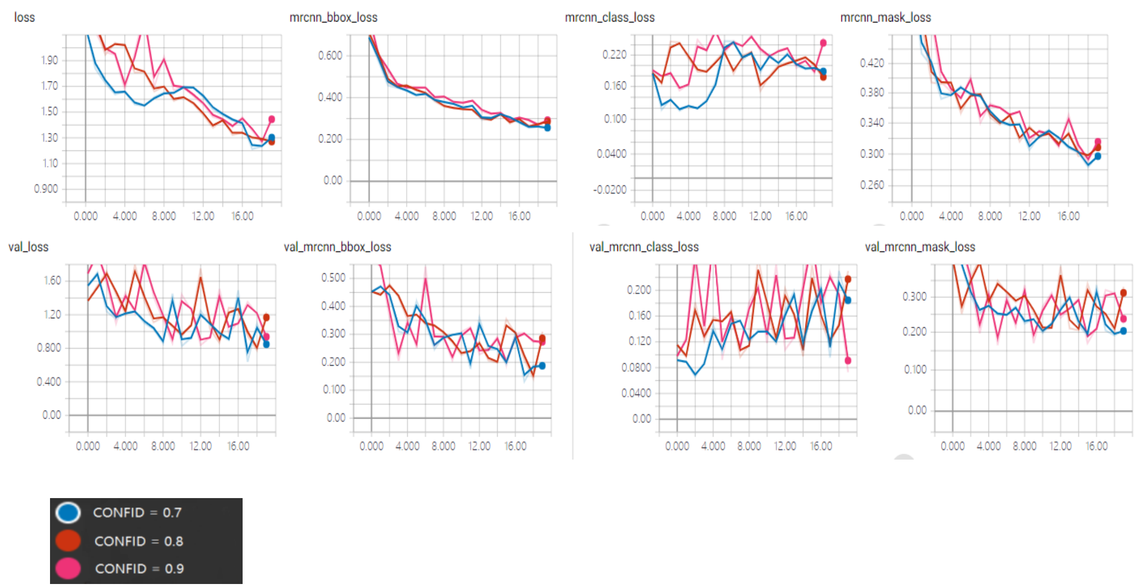

Figure 12 Training Result for Different Detection Confidence.

	CONFID = 0.7	CONFID = 0.8	CONFID = 0.9
LOSS	1.314	1.268	1.475
MRCNN_BBOX_LOSS	0.2552	0.2870	0.2962
MRCNN_CLASS_LOSS	0.1872	0.1736	0.2538
MRCNN_MASK_LOSS	0.2992	0.3101	0.3196
VAL_LOSS	0.8139	1.232	0.8876
VAL_MRCNN_BBOX_LOSS	0.1887	0.1411	0.2717
VAL_MRCNN_CLASS_LOSS	0.1797	0.2299	0.07297
VAL_MRCNN_MASK_LOSS	0.2049	0.3308	0.2242
DURATION	25m 50s	27m 36s	27m 55s

Table 4 Training Result for Different Detection Confidence.

While comparing the result across different detection confidence we couldn't see very big difference so that we consider this factor as a minimum impact hyper-parameter.

MODEL EVALUATION

After the hyper-parameters have been defined and settle in the training model, we start to train the final model with the following setting:

- Backbone：ResNet50 (The training machine GPU can't support for ResNet101)
- ROI: 10
- Image Size: 256
- Detection Confidence: 0.8
- Layers: all

After the training finished, we tested the model with 6 randomly download images with chair and sofa from internet which is not side our training and testing dataset and look at the performance of the instance segmentation (see in Figure 13):

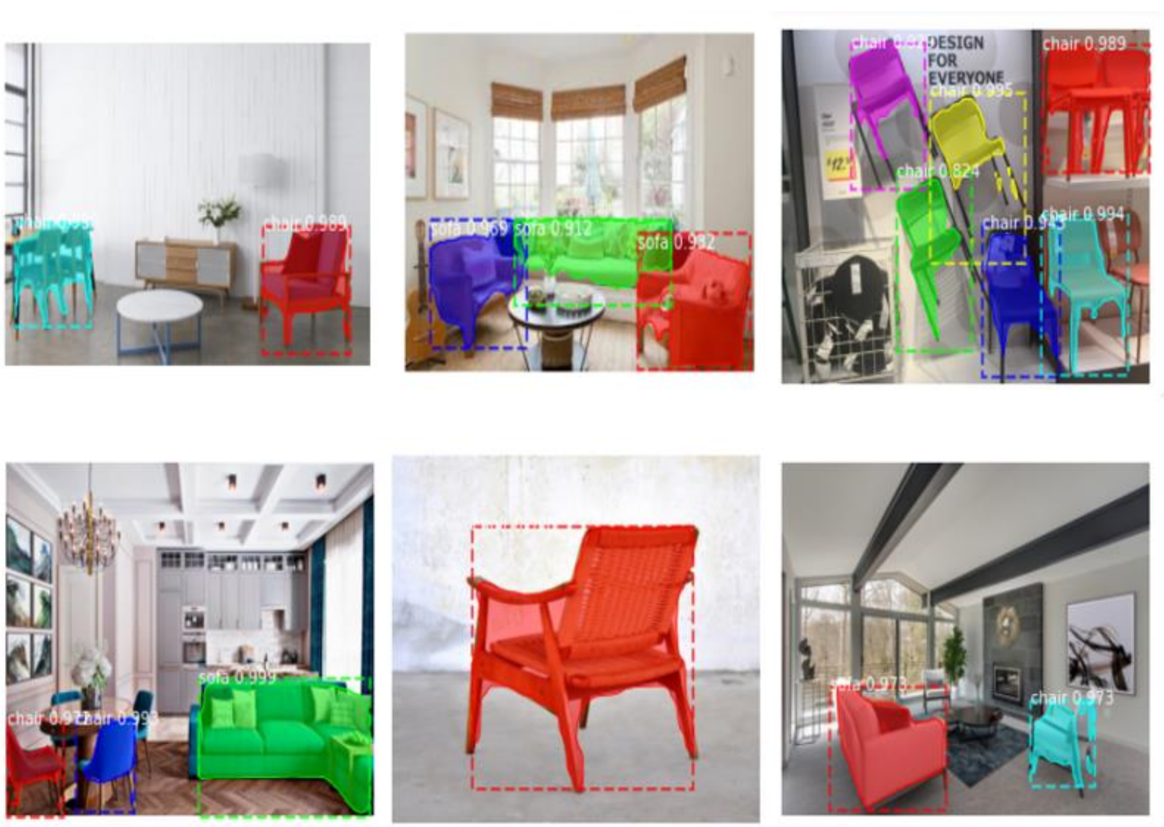

Figure 5 Instance Segmentation by Final Model.

As we can see from the testing images, our model is able to segment almost all the chairs and sofas inside the picture and the mask cover the instances very well with very high confidence to the classification.

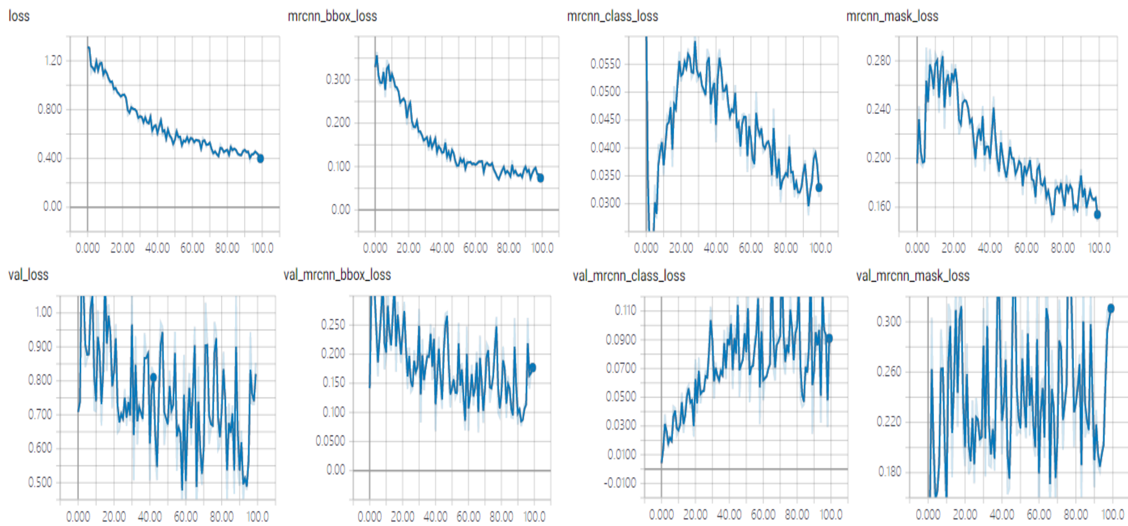

Figure 14 Training Result for Final Model.

	FINAL
LOSS	0.3880
MRCNN_BBOX_LOSS	0.07362
MRCNN_CLASS_LOSS	0.03122
MRCNN_MASK_LOSS	0.1485
VAL_LOSS	0.8532
VAL_MRCNN_BBOX_LOSS	0.1806
VAL_MRCNN_CLASS_LOSS	0.1087
VAL_MRCNN_MASK_LOSS	0.3143
DURATION	1h 50m 0s
EPOCH	100
OVERALL ACCURACY	0.7555211644503805

Table 5 Training Result for Final Model.

From the tensor board we can see the model loss and validation result all quite decent, the overall accuracy is around 75.6% based on the whole testing dataset.

CONCLUSION

In conclusion Mask R-CNN is a very good architecture to do instance segmentation. However, to achieve its potential we need to spend a lot of time to proper tuning of hyper-parameters. Methods like GridSearch with cross validation might not be useful in cases of CNN because of huge computational requirements for the model and hence it is important to understand the hyper-parameters and their effect on the overall prediction [4]. Within the project we mainly testified on four hyper-parameters:Backbone, ROIs_Per_Image, Image_Dim and Detection_Min_Confidence and apply the transfer learning method to train the model as fast as possible also get decent accuracy. With the visualization training result from tensor board we easily pick up the parameters setting which are most suitable for our cases.

REFERENCE

[1] Deep learning–based image instance segmentation for moisture marks of shield tunnel lining by ShuaiZhaoDong MingZhangHong WeiHuang

[2] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks by Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

[3] Mask R-CNN by *Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick*

[4] Taming the Hyper-Parameters of Mask RCNN by Ravikiran Bobba