FAI Final Report: Investigating the Urban Score in Different Parts of a City

As a civil engineering student, besides dealing with structures and soil, we also have to study urban planning, including traffic flow and characteristics of different areas in a city. It is particularly important in countries like Japan because of the frequent occurrence of disasters, hence when we design a city or a neighbourhood, we also have to consider if residents are able to evacuate properly. In this report, I would like to focus on urban planning, and use deep learning to investigate the density of infrastructures in the area. The following code is just a simple concept of what I want to achieve in machine learning, meaning that my code still needs further development and more polishing.

Urban Planning

To start off, we have to first understand what urban planning is. According to Wikipedia, urban planning is a technical and political process that is focused on the development and design of land use and the built environment. It is a study that requires meticulous thinking and predictions, which also affects the livelihood of people. A badly designed neighbourhood can have adverse effects on the person's long-term mental health, increase local crime rates, or even increase the risk of chronic diseases such as respiratory issues due to factors such as bad air quality and noise pollution. A neighbourhood has to be carefully designed in a way that there is no overcrowding, with adequate green spaces and sufficient facilities to suit everyone's needs. Urbanization is also related to economic development and disaster control; hence it is a subject that cannot be taken lightly.

The Problem

One of the most straightforward ways to study a city is from its infrastructure. Knowing the density of buildings and in reverse, the number of open spaces in that area and how far apart each cluster is from each other can help urban designers in a lot of things, such as designing the networks for public transportation, electricity, and water distribution and such. A recent example is investigating the clusters of Covid-19 infections. Since it can be transmitted easily, scientists have to analyse whether the buildings within a certain radius of the infection will be affected so that residents can be evacuated. Furthermore, in risk assessments of infrastructures, such as building a factory nearby, planners have to first check which part of the area is suitable for constructing a factory, how the process will affect the residents nearby, and the hazards that may occur to people in a certain radius of damage. With a rapid rate of population growth and other changes in modern society, it is difficult to keep track of everything. If you count the number of buildings from a satellite map manually, it may change after a day. Hence, it is important for us to come up with a way to monitor the changes in the area so that our data can be up to date. In order to achieve that, I would like to use deep learning to analyse images of satellite maps so that it can identify and map out the buildings for us rather than finding someone to do it manually.

Machine learning vs Deep learning

The reason why I have chosen deep learning is that it involves image recognition. First, let me explain the differences between machine learning and deep learning. In the past lectures, we have come across both of the models. For machine learning, it refers to instructing the computer to use algorithms to perform certain tasks without any complicated structures. An example would be predictive modelling, which we learned in class using regression to investigate the relationship between variables and predict an outcome. Deep learning, however, is much more complex than machine learning. Deep learning utilises artificial neural networks, or “ANN” for short, which simulate the neural network of a human brain. You can imagine how complex it can be. While a model in machine learning has to be programmed explicitly for that specific task, deep learning models are made to allow the machine to “think and act”. There are layers of neural networks connected to each other, resulting in a “non-linear outcome”. Deep learning models also require a large amount of data to train the system, so that it can “learn”. Compared to machine learning, it may take some time to "feed" the network to train it, but it can generate more complex outputs without human intervention.

So, how does image recognition work with deep learning models? In deep learning, there is a type of network called the "Convolutional Neural Network". This type of network works specifically well with images, as it divides the image into different layers or elements, adjusts it until the computer can "understand" the image. In the following part, I would like to demonstrate how the computer can recognize and label the objects found in an image using the object detection API from TensorFlow. I have also referred to the tutorial by TensorFlow regarding object detection.

Code

Since the exact code can be found on the website, I will just highlight a few parts of the code since my main focus will be the end result.

First, we have to import the library from TensorFlow because we are using its API:

```
import tensorflow as tf
```

Next, we import the trained machine learning models from TensorFlow (we need this because the pre-trained models are useful for the neural networks, including the image classification models):

```
import tensorflow_hub as hub
```

To import the image, we can ask our computer to download the image itself by only providing the url of the image. The computer can then download the image and save it to a temporary location, and can resize it accordingly.

```
import matplotlib.pyplot as plt
import tempfile
from six.moves.urllib.request import urlopen
from six import BytesIO
```

Since our goal here is to not only identify the objects but to label them in the given image, the following libraries from numpy and PIL (Python Imaging Library). Our computer would first identify the object(s) in the image, determine its location, and label it by drawing a frame around it.

```
import numpy as np
from PIL import Image
from PIL import ImageColor
from PIL import ImageDraw
from PIL import ImageFont
from PIL import ImageOps
```

Last but not least, we can import the time module from Python as it can come in handy when measuring the runtime and/or inference time for the model to process the data.

```
import time
```

Here is what the entire setup will look like:

```
#@title Imports and function definitions

# For running inference on the TF-Hub module.
import tensorflow as tf

import tensorflow_hub as hub

# For downloading the image.
import matplotlib.pyplot as plt
import tempfile
from six.moves.urllib.request import urlopen
from six import BytesIO

# For drawing onto the image.
import numpy as np
from PIL import Image
from PIL import ImageColor
from PIL import ImageDraw
from PIL import ImageFont
from PIL import ImageOps

# For measuring the inference time.
import time
```

Below are some helper functions that can come in handy later:

After setting it up, we can move forward to download an image from the Internet and resize it to the desired dimensions.

```
image_url = "https://www.satellitetoday.com/wp-
content/uploads/2018/04/0003_AUS_Sydney_Jan06_2015_WV3_30cm-1.jpg"  #@param
downloaded_image_path = download_and_resize_image(image_url, 1280, 856, True)
```

In order for it to work, we have to apply an object detection module on the downloaded image. According to the documentation, there are two modules, which are:

- **FasterRCNN+InceptionResNet V2**: gives higher accuracy but slow inference time
- **ssd+mobilenet V2**: a light module with fast inference time but lower accuracy

In this case, I would like to try out both of the modules and see how each of them fare against each other. To test the modules, I am going to use a satellite image of Sydney, Australia found here. Let's call the first module "Module A" and the second one "Module B".

To use the module, we first have to create a "handle" to connect to the module. After that, we can load it into the TensorFlow hub we imported at the beginning. Here, I will be using Module A.

```
module_handle =
"https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"
detector = hub.load(module_handle).signatures['default']
```

Below are the functions that help the computer to understand the image. In the `load_img(path)` function, it reads the image and decode the image into 3 color channels. The attribute `channels` accepts 4 values, which are `0`, `1`, `3`, and `4`, which stands for:

- 0: the default number of color channels in the image
- 1: grayscale

- 3: RGB
- 4: RGBA

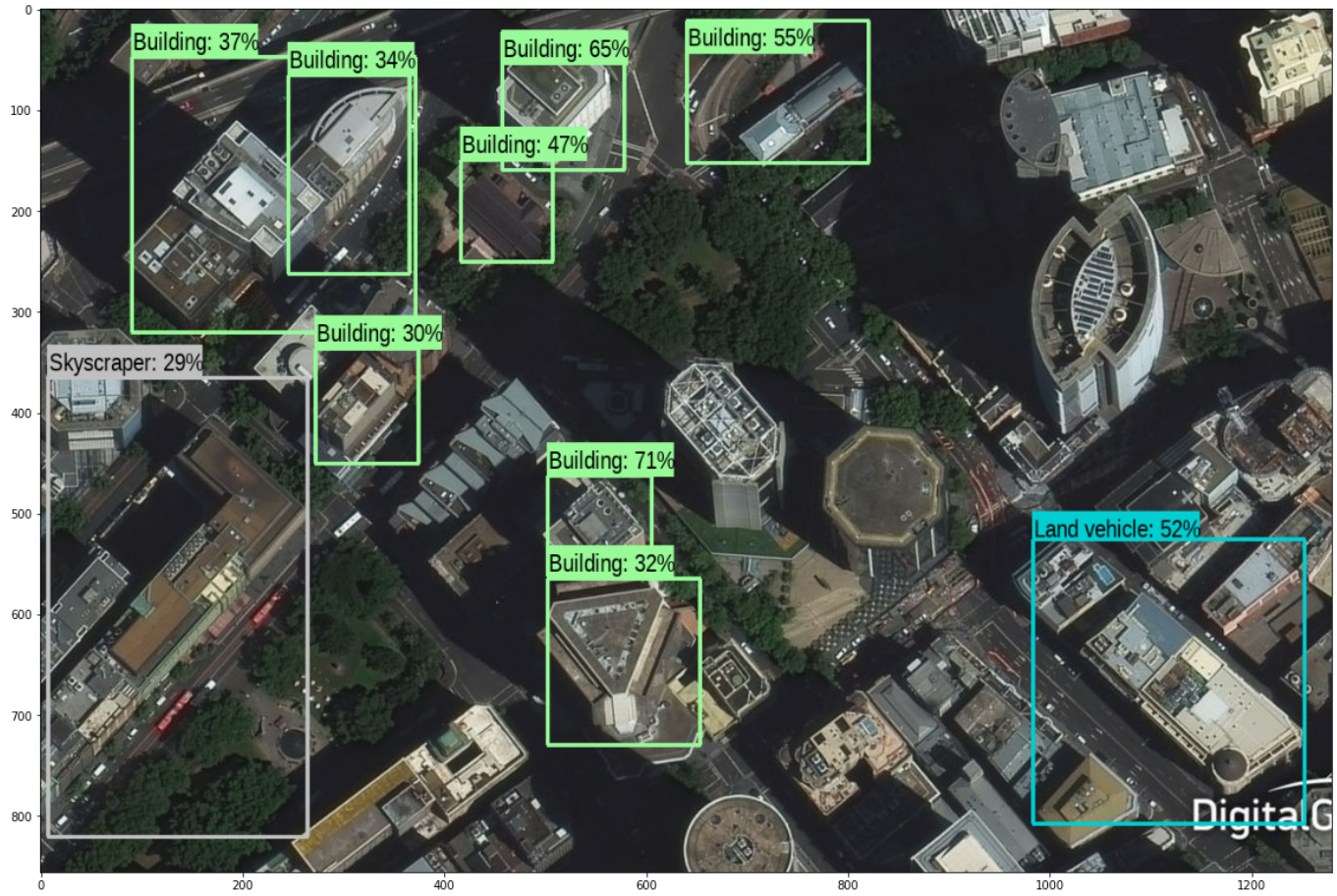
The `run_detector(detector, path)` function is the main function in the code. The computer analyses the image and lable them at the same time. A timer also starts when the code starts to run to calculate the inference time.

```
def load_img(path):
  img = tf.io.read_file(path)
  img = tf.image.decode_jpeg(img, channels=3)
  return img

def run_detector(detector, path):
  img = load_img(path)

  converted_img  = tf.image.convert_image_dtype(img, tf.float32)[tf.newaxis, ...]
  start_time = time.time()
  result = detector(converted_img)
  end_time = time.time()

  result = {key:value.numpy() for key,value in result.items()}

  print("Inference time: ", end_time-start_time)

  image_with_boxes = draw_boxes(
      img.numpy(), result["detection_boxes"],
      result["detection_class_entities"], result["detection_scores"])

  display_image(image_with_boxes)
```

Now, we can run the detector:

```
run_detector(detector, downloaded_image_path)
```

Output:




```
Inference time:  28.419305562973022
```

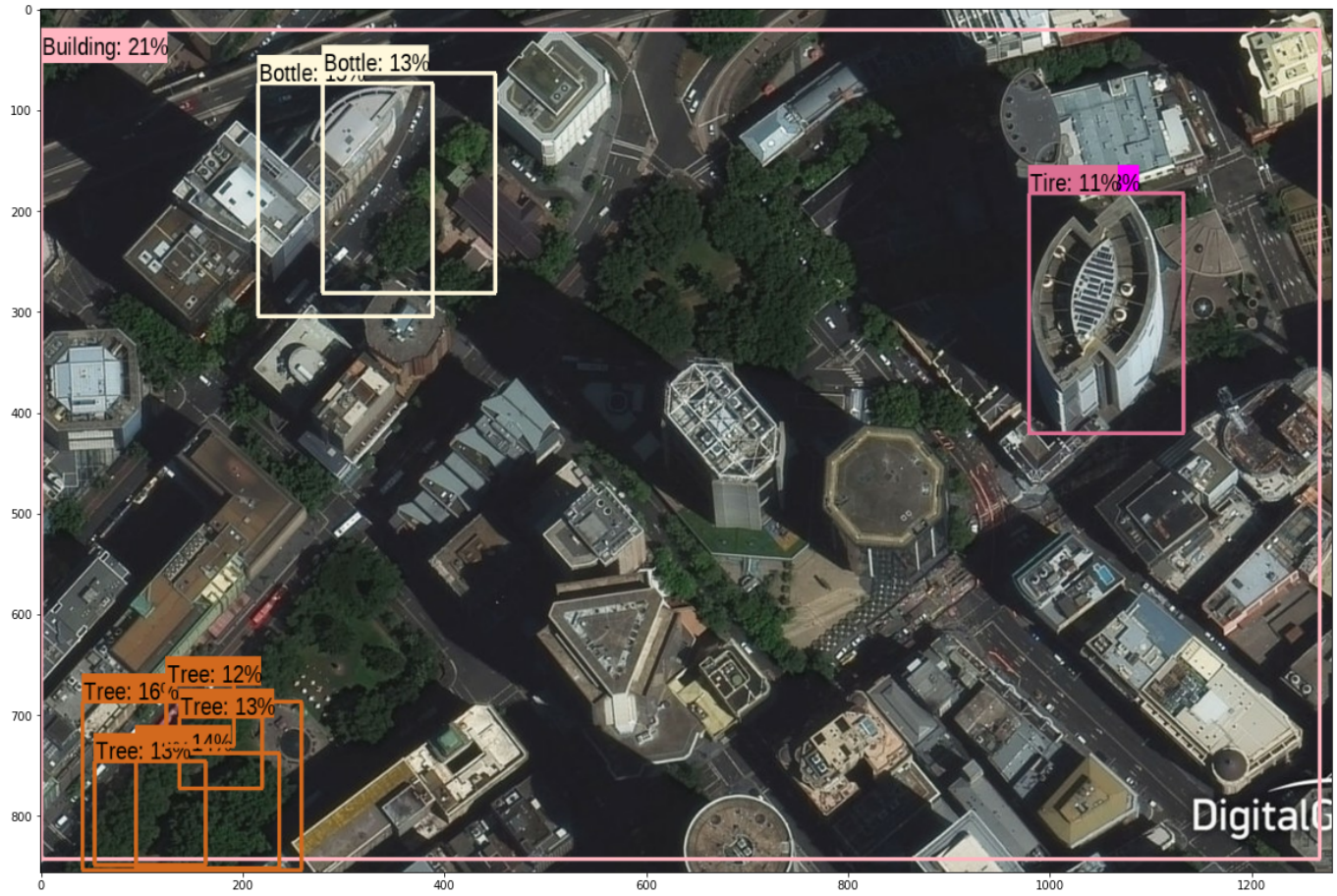
Let's try using Module B. Again, we create a handle first.

```
module_handle = "https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1"
detector = hub.load(module_handle).signatures['default']
```

Since the functions have been defined already, we can call the function directly:

```
run_detector(detector, downloaded_image_path)
```

Output:




```
Inference time:  6.086521148681641
```

As you can see from the results, Module B is nearly 5 times faster than Module A, but it is also much less accurate than Module A.

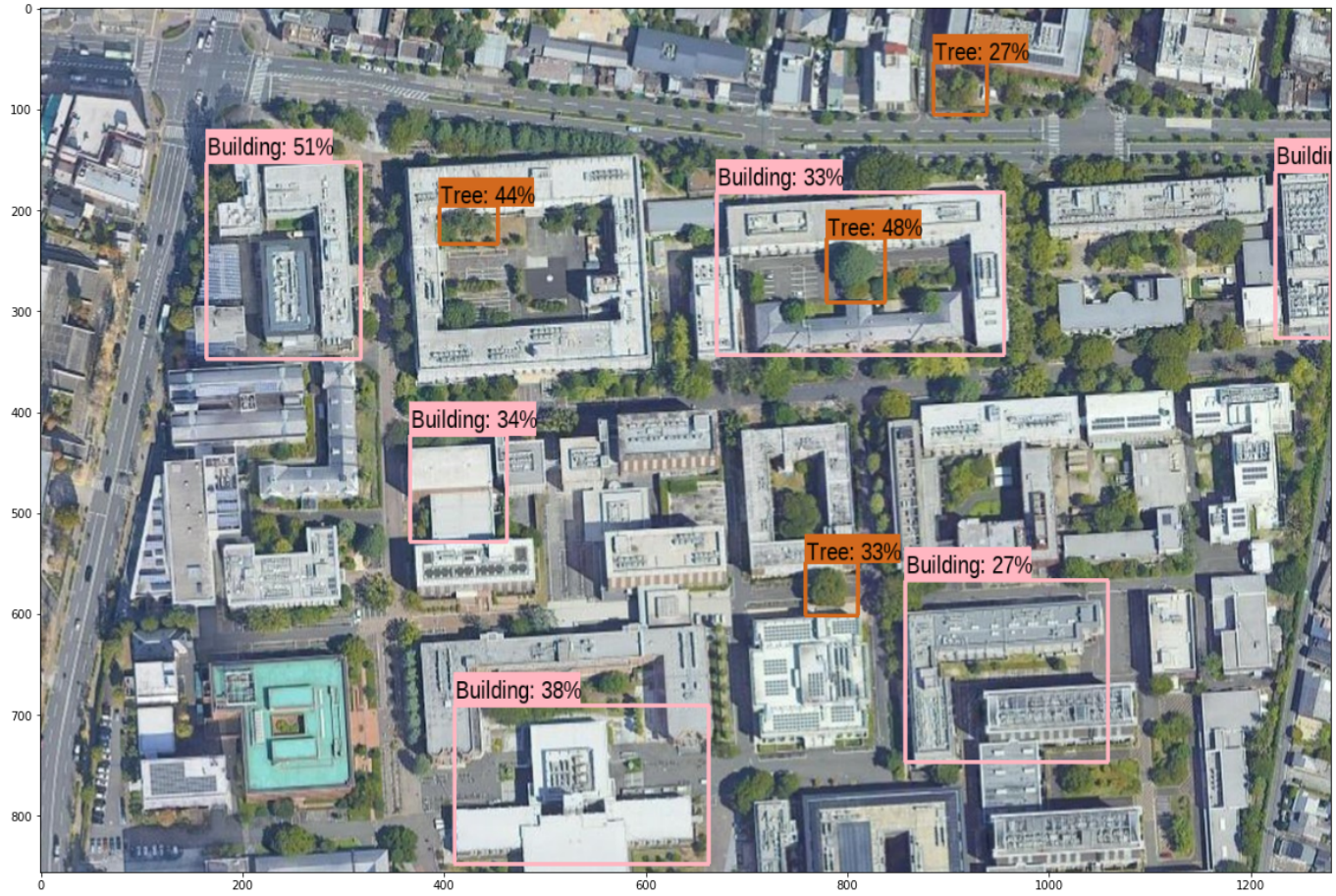
Ways to improve the accuracy of the model

From the results above, even by using Module A, the accuracy of the model does not seem to be very good. It can be due to insufficient data to train, or not enough layeres in the model. To help the model understand more about the image, we can try feeding it more data by using our own training data set that is made for recognising buildings, or adding relevant layers so that it can distinguish between different shapes or even types of buildings. When training the model, we can also try out data augmentation, which changes the already exisiting image randomly, such as rotating the image or flipping. These transformations can help our model understand the data even more, leading to a more accurate result.

More images

Let's try using the two modules on a different image. This time, I have taken a screenshot of the satellite image above Kyoto University using Google Maps.

Module A:




```
Inference time:  28.9584903717041
```

Module B:


```
Inference time:  5.297305345535278
```

Conclusion

After experimenting with the object detection API by TensorFlow, I find it amusing that the computer is able to recognize most of the buildings from the satellite image. This shows that with more improvements to the model, the computer can not only pinpoint the locations of the buildings, it can also trace the outline of the buildings and use the information to make decisions, such as whether the area is too dense or not. After doing some research online, I have also found some interesting models that others made that able to map out building footprints precisely just from a single image. This would assist civil engineers in many ways, especially in disaster risk management by telling them the outline of the buildings so that they can have a better understanding of the geospatial data in the area. It can also help them to visualize what the surroundings of a new building would be, whether it is suitable for people to live in and how easily people can get around in the area. Telecommunication companies and other businesses can also use that information to decide where to expand their business. Hence, a great understanding of the geospatial data of the city can benefit everyone, which is also where artificial intelligence can lend a hand.

Output of the model created by the winning team in the Open Cities AI Challenge.