Speakable Maps

Emily Ong Hui Qi

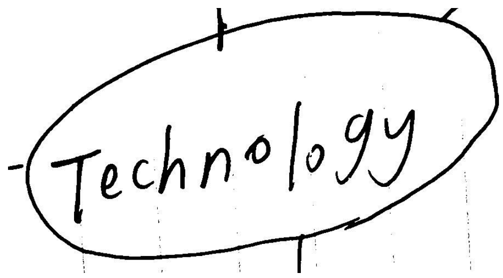
Introduction

A mind map is a non-linear visual representation of concepts, often originating from a single idea and explicitly showcases the relationships between various concepts. However, one problem is that it is hard for the blind to interact with mind maps.

Proposed Solution

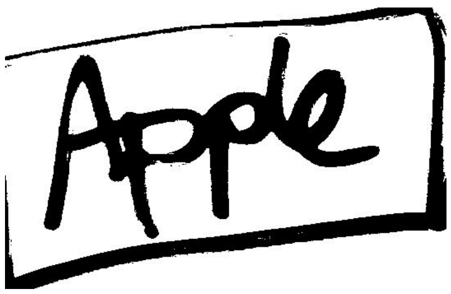
The proposed solution defined in this project takes a sketch/image of a mind map and converts it to interactive elements on a HTML canvas, which supports mind-mapping and text-to-speech. To generalise the structure of a mind map, we define three key features:

1. Ellipse (to represent a main



topic/sub-topic)

Fig. 1: Ellipse



2. Rectangle (to represent an idea)

Fig. 2: Rectangle

3. Arrows/Lines (to represent

connections between nodes)

Fig. 3: Arrows/Lines

An example of a defined mind map:

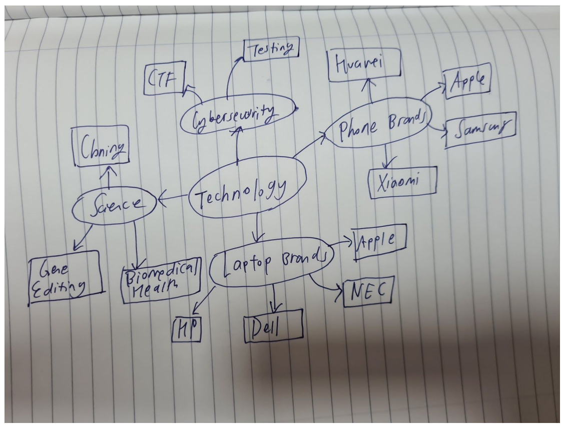


Fig. 4: Mind map

Data Collection

An immediate solution for the training data would be "Google's Quick, Draw!"[1] dataset, which contains numerous labelled data files of the required object classes (ellipse, rectangle, line). However, it is not fully representative of a mind map, as the ellipses and rectangles are non-empty, and possibly contain text/scribbles.

[1] https://quickdraw.withgoogle.com/data

Therefore, we created our own dataset, containing 30 different images, of which 10 images are taken from the internet and 20 images are hand-drawn. The hand-drawn images consist of a mixture of marker ink and pen ink to reflect the differences in brush sizes.

Data Processing

Image Labelling

To annotate the images, we use an open-source software LabelImg[2], and create bounding boxes for each feature in the image.

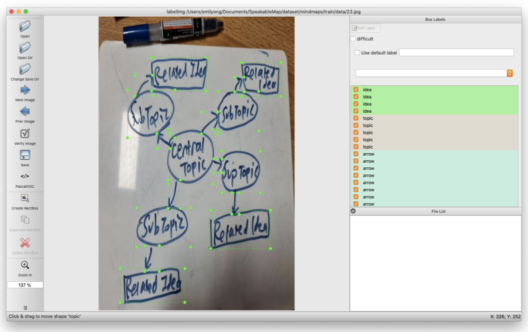


Fig. 5: LabelImg

Image Compressing

To speed up training, we reduce the resolution of the image files.

Data Split

80% of the data (24) is used for training, while 20% of the data (6) is used for testing.

Training

To predict the location of the object along with its class, we use object detection techniques. Object detection is modelled as a classification problem, where a sliding window is used to classify possible objects within the given area. Hence, it is possible to know both the class and location of the objects in the image.

However, as the objects can be of varying sizes, an image pyramid is created by scaling the image until it fits within the chosen window size.

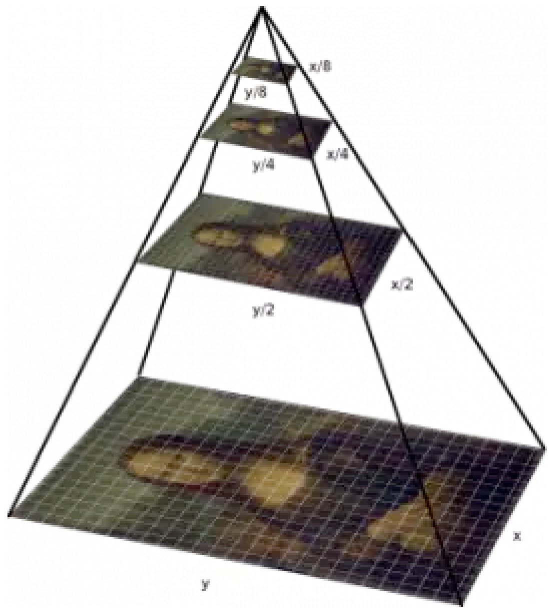


Fig. 6: Scaling image size[3]

To be able to classify objects within an image, a convolutional neural network (CNN) is used. Due to the slowness and computational complexity of CNN, RCNN is developed to use selective search to reduce the number of bounding boxes that are fed to the classifier, based on certain properties of the area within the image (e.g. texture, intensity, color).

In Faster RCNN, selective search is replaced with a very small convolutional network called the Region Proposal Network to generate regions of interest, and uses anchor boxes to handle the variations in aspect ratio and object scale.

[2] https://github.com/tzutalin/labelImg

[3] https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

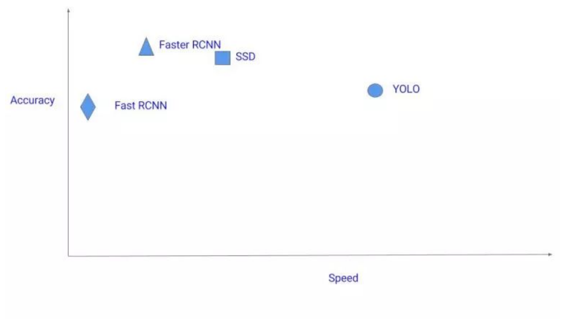


Fig. 7: Model comparison[4]

In comparison to other models, the Faster RCNN model achieved the highest accuracy, albeit at a slower speed, which is a reasonable trade-off for this project.

Thus, we use the Tensorflow Object Detection API (Application Programming Interface), with the Faster RCNN (Region-based Convolutional Neural Network) Inception Resnet model.

Text Recognition

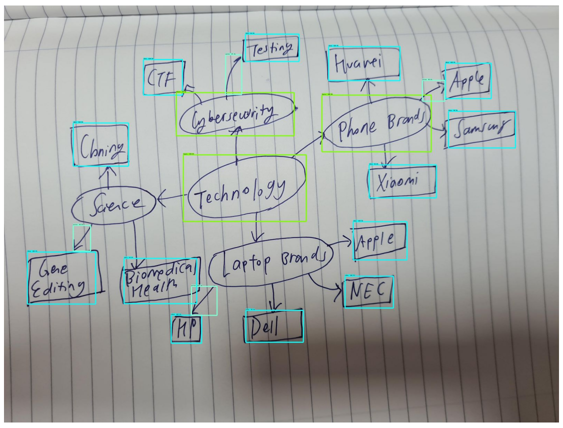


Fig. 8: Visualisation box

To generate the bounding boxes, we use the `visualize_boxes_and_labels_on_image_array` helper from the object_detection library, which generates 100 possible detection boxes. We selectively filter the detection boxes based on a predefined threshold to obtain detection boxes with a reasonable level of accuracy.

The image is then cropped using the coordinates of the bounding box and converted to grayscale. We use the Google Cloud Vision API to recognise the text within the image.

Mindmap

For each of the chosen bounding boxes containing arrows, we compute the error for that bounding box in relation to other bounding boxes containing either topics or ideas. To compute the error, we treat the bounding box containing the arrow as the origin, and calculate the Euclidean Distance between the two points (red line). Then, we store an array of all the connections.

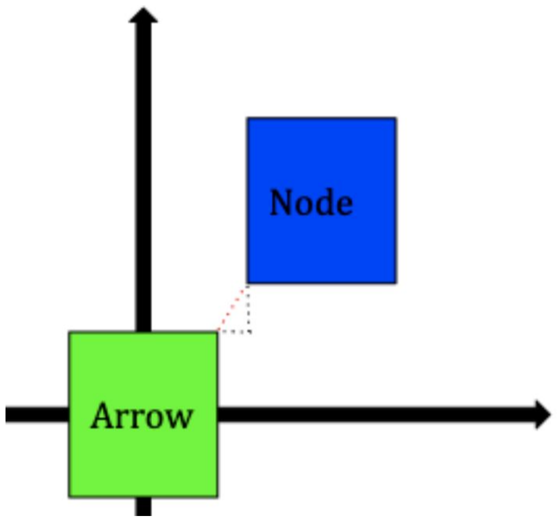


Fig. 9: Error

On the front-end, we use Fabric.JS[5] to provide helpers to generate shapes and text on the canvas, while the visualisation code for the mind map is self-coded.

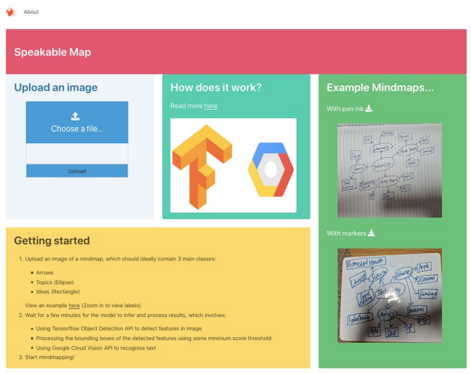
4 https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

5 http://fabricjs.com/

Speech Synthesis

To provide ease of use for the blind, we define a shift+hover action that will sound out the text in the hovered node and all other nodes connected to it.

Front-End Design

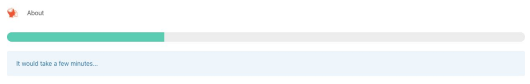


Getting started

1. Upload an image of a mindmap, which should ideally contain 3 main classes:
   - Arrows
   - Topics (Ellipse)
   - Ideas (Rectangle)

   View an example here (Zoom in to view labels)
2. Wait for a few minutes for the model to infer and process results, which involves:
   - Using Tensorflow Object Detection API to detect features in image
   - Processing the bounding boxes of the detected features using some minimum score threshold
   - Using Google Cloud Vision API to recognise text
3. Start mindmapping!

Fig. 10: Main screen



It would take a few minutes...

Fig. 11: Loading screen

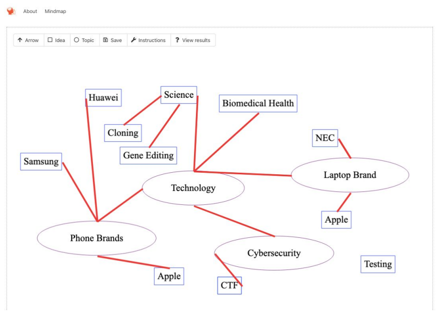


Fig. 12: Mind map

References

- https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#