# Hand Gesture Modelling – Documentation

## Source of Data

This study utilizes two sources of images to construct the dataset for hand gesture modelling, by personally collecting the images and by scrapping images from the web via Google Images.

## Description of Dataset

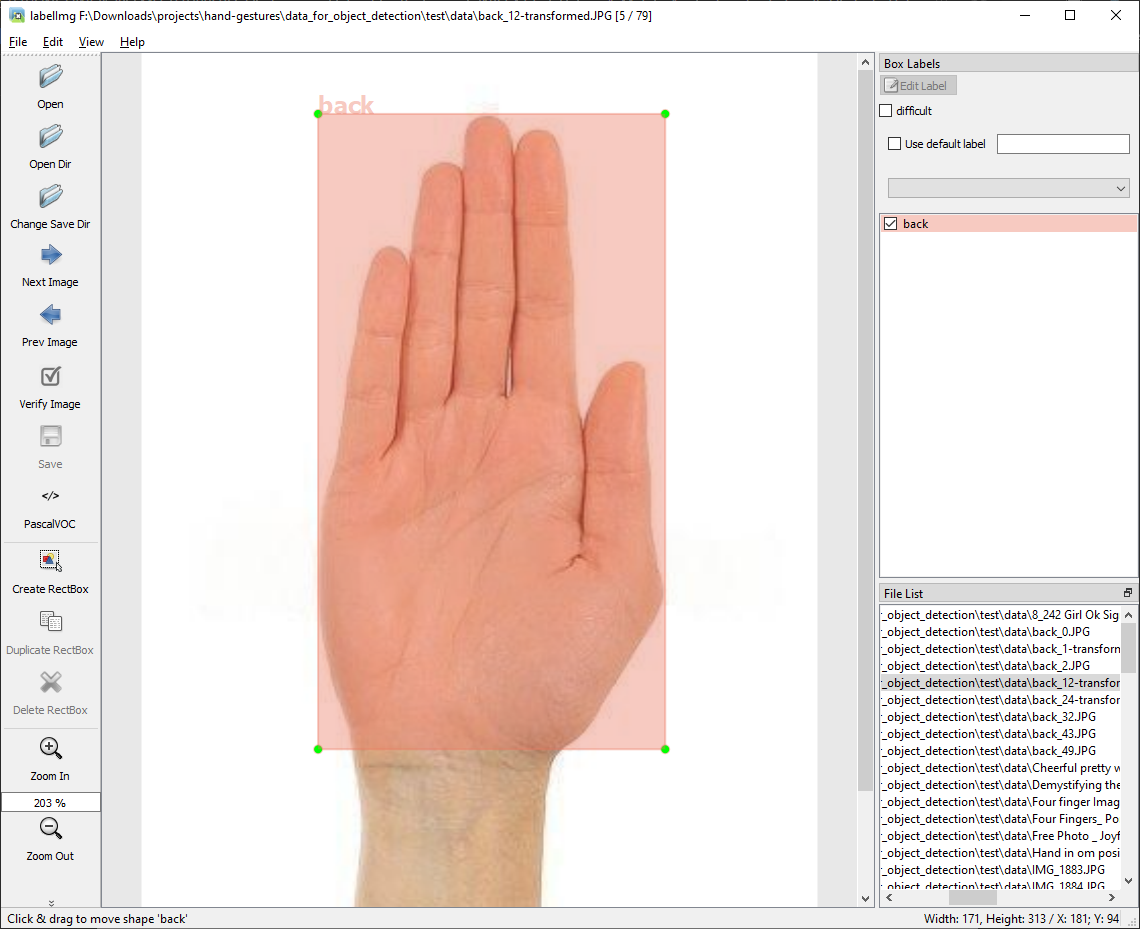
The constructed image dataset consists of eight classes for different hand gestures: one-finger, two-fingers, three-fingers, four-fingers, five-fingers, fist, okay and thumbs up (feel free to rename these to your own ones). The dataset contains a total of 880 images with each class having exactly 110 images per class, as shown in Table X. The resolution of the images differs greatly with few images that exceed 2400×2400 pixels. The sample images are illustrated in Table X.

|  |  |
| --- | --- |
| **Classes** | **Count of Images** |
| One-finger | 110 |
| Two-finger | 110 |
| Three-finger | 110 |
| Four-finger | 110 |
| Five-finger | 110 |
| Fist | 110 |
| Okay | 110 |
| Thumbs up | 110 |
| **Total** | **880** |

|  |  |
| --- | --- |
| **Classes** | **Sample Images** |
| One-finger |  |
| Two-finger |  |
| Three-finger |  |
| Four-finger |  |
| Five-finger |  |
| Fist |  |
| Okay |  |
| Thumbs up |  |

## Data Pre-Processing

This study performs data pre-processing to improve the quality of data before training the deep learning algorithms. As a result, this study hand-picked images of different angles and resolutions to ensure the algorithms are trained as thoroughly as possible. LabelImg, an open-source tool for image annotation was used to annotate the images in the constructed dataset, as shown in Figure X. Consequently, a total of 880 bounding boxes were created in the Pascal VOC format. The same number images within each class ensures a class is not a majority or a minority as it hinders the performance of the model. The dataset will be split into training and test set with 800 images utilized for training and 80 for testing.



## Model Development

This study utilizes the TensorFlow Lite Model Maker library for the task of model development as it automatically converts the trained models into a TensorFlow Lite format that can be run on mobile devices. The EfficientDet-Lite[0-4] algorithms will be trained on the constructed dataset. The main difference between the EfficientDet-Lite algorithms lie in their complexity, with EfficientDet-Lite4 being the most complex. As a result, the average precision, model size and latency differ between the algorithms. The algorithms will be trained for 50 epochs with a batch size of eight.

# Model Evaluation

This study utilizes the mean average precision (mAP) and inference time to evaluate the trained models. mAP being a commonly used evaluation metric for object detection tasks, a higher mAP indicates a better performing model. In particular, the COCO mAP evaluation will be adopted. The TensorFlow Lite Model Maker automatically evaluates the mAP of the models on the test data. (more info: [Mean Average Precision (mAP) Explained: Everything You Need to Know (v7labs.com)](https://www.v7labs.com/blog/mean-average-precision) & [Mean Average Precision (mAP) in Object Detection (learnopencv.com)](https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/))

Formula of a typical mAP: <https://blog.paperspace.com/content/images/2020/09/Fig13-1.jpg>

The different metrics in mAP COCO: <https://learnopencv.com/wp-content/uploads/2022/08/mean-average-precision-map-coco-metric.png>

Overlap (or IOU) for object detection tasks: <https://929687.smushcdn.com/2633864/wp-content/uploads/2016/09/iou_stop_sign.jpg?lossy=1&strip=1&webp=1> => AP of 50% for a class means on average the overlap is half between the ground-truth (original) and predicted.

The inference time indicates the time it takes the model to infer the images, with lower inference time being favourable in real-time systems. In this study, the inference time will be measured in milliseconds over 50 inferences and calculated by taking the average.

# Results

## mAP

Table X (or Figure X) shows the evaluation metrics for the trained models on the test data. We can observe that all the models achieved an mAP of 82% and above, with EfficientDet-Lite4 achieving the highest mAP of 87%. Although the EfficientDet-Lite0 achieved the lowest mAP value, the difference between the top-performing model is not significant indicating the least complex model performs almost as good as the most complex one. A notable observation is that there is no much difference between the EfficientDet-Lite[2-4] models.

|  |  |  |  |
| --- | --- | --- | --- |
| **name** | **AP** | **AP@50** | **AP@75** |
| EfficientDet-Lite0 | 0.825286 | 0.941674 | 0.914654 |
| EfficientDet-Lite1 | 0.833719 | 0.938586 | 0.92558 |
| EfficientDet-Lite3 | 0.863767 | 0.948887 | 0.947307 |
| EfficientDet-Lite2 | 0.86955 | 0.956028 | 0.95297 |
| EfficientDet-Lite4 | 0.873883 | 0.950745 | 0.947101 |

Background pattern

Description automatically generated

Table X shows the mAP values of the trained models on the test data within the individual classes. We can observe that the models had a difficulty in distinguishing one-finger (index finger) and four-fingers. This could be due to the fact that three-fingers resemble four-fingers greatly. As a result, the models have a difficulty to recognise their differences in some cases. [you can mention anything interesting you find in the tables]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **name** | AP\_/option1 | AP\_/option2 | AP\_/option3 | AP\_/option4 | AP\_/punch | AP\_/thumb | AP\_/ok | AP\_/back |
| EfficientDet-Lite0 | 0.708798 | 0.852636 | 0.902165 | 0.773388 | 0.893846 | 0.819058 | 0.752663 | 0.899737 |
| EfficientDet-Lite1 | 0.683847 | 0.880741 | 0.9033 | 0.71972 | 0.880428 | 0.880657 | 0.803157 | 0.917905 |
| EfficientDet-Lite3 | 0.692625 | 0.872802 | 0.914003 | 0.837449 | 0.871287 | 0.934606 | 0.849764 | 0.937601 |
| EfficientDet-Lite2 | 0.698252 | 0.880446 | 0.897131 | 0.819059 | 0.907591 | 0.922459 | 0.908465 | 0.922999 |
| EfficientDet-Lite4 | 0.731876 | 0.893831 | 0.929731 | 0.754279 | 0.92479 | 0.939367 | 0.944087 | 0.873102 |

## Inference Time

The average inference time was calculated by taking the average inference time of 50 inferences by each model, as shown in Table X. We can observe that the inference time varies greatly between each model, with EfficientDet-Lite4 reaching up to 10 seconds to infer (or predict) a single image. The trend observed can be linked to the complexity of each model. As mentioned previously, the models differ in their complexity and as such, more complex models require more computational resources from the mobile device for detection. The EfficientDet-Lite0 model achieves the fastest inference time on average, with less than a second.

|  |  |
| --- | --- |
| **Model Name** | **Average Inference Time (in ms)** |
| EfficientDet-Lite0 | 729.76 |
| EfficientDet-Lite1 | 1315.78 |
| EfficientDet-Lite3 | 2009.18 |
| EfficientDet-Lite2 | 4085.24 |
| EfficientDet-Lite4 | 10047.46 |

Figure X shows a scatter plot of the mAP performance of each model against the average inference time in seconds. We can observe a clear trade-off between the performance and latency, models with higher mAP resulting in higher inference durations.

Chart, scatter chart

Description automatically generated

## Mobile Application

Screenshots Full Quality: [Imgur: The magic of the Internet](https://imgur.com/a/mC14Wg2)

Another screenshot: [TGyU3Uf.png (1080×2220) (imgur.com)](https://i.imgur.com/TGyU3Uf.png)