# SSD Object Detection Model for Blender Gestural Input Interface

Marzio Vallero
Politecnico di Torino
C.so Duca degli Abruzzi, 24, 10129, Torino, ITALY
marzio.vallero@studenti.polito.it

## Abstract

*This paper aims at describing the steps required for the training of a custom purpose Object Detector neural network to be used as a mean of interaction for the 3D modeling software Blender. The first part will describe the generation of a custom dataset for training over hand gestures containing images of a single person, explaining techniques used to reduce as much as possible any bias due to this. This dataset will be then used to transfer learn a Single Shot Multibox Detector with Feature Pyramid architecture through Tensorflow's Object Detection API. The results will be evaluated through training statistics and direct experimentation with the trained model, discussing about the approaches used and possible improvements. Demos and code relative to this paper are available at the project's GitHub page.*

## 1. Introduction

Most of modern era 3D software human-computer interfaces rely onto the usage of a mouse and a keyboard. This can prove to be a difficult task to learn, as this mean of manipulation can seems daunting and complex for newcomers.
It has been proven already by Mavinkurve *et al.* [4] that Engineering Graphics are amongst the most difficult tasks to fully grasp for students and that 3D simulated environments improve their learning capabilities.
This paper aims at detailing the steps required to create an object detection neural network able to recognize static hand gestures. Such network is a Single Shot Multibox Detector, as for the work of Liu *et al.* [3], refined by Google Brain researchers Ghiasi *et al.* [1], and trained on Microsoft's COCO 17 dataset (Lin *et al.* [2]). Its checkpoints are publicly available through the Tensorflow Model Garden, developed by Yu *et al.* [9].
At first, will be introduced the steps for the generation of a custom dataset for training, by applying data augmentation, also generating negative samples. This is done to reduce the bias due to the fact that it contains only images of the same person. This dataset will then be used to train through transfer learning (as per shown by Zhuang *et al.* [10]) a checkpoint of the SSD and discuss about training statistics and experimental results.
The hereby proposed input interface could see applications as an instrument for learning (molecules, human body models or mechanical models) and as a mean of interaction in museums or during large-scale presentations.

## 2. Dataset

The first step amounts to the generation of the custom dataset, justified by the fact that no current dataset exists containing all of the gestures required for this specific scope. All the following steps are also described in depth in the Python notebook DatasetGeneration.ipynb as an additional support for understanding.
The first step is responsible for the definition of the set of gestures and the set up of automated sample capture. After defining the number of images to collect for each object class, the frames are captured, then are automatically saved in separate directories according to their label of reference. For the sake of simplicity, it is recommended to produce for each label the same amount of samples and to insert a single gesture per frame. The effect of multiple, possibly different labels present in the same frame could be further investigated in future work.
The amount of data at the network's disposal for training greatly impacts its ability to learn: for this very reason, data augmentation has been exploited in order to increase the number of available samples for the model's training, whilst trying to reduce the implicit bias due to the fact that the dataset contains only samples of the same individual. The data augmentation transformations used are horizontal/vertical stretching and zooming. Other data augmentation techniques could have caused shape overlaps between gestures and have thus been avoided. Negative samples have been introduced to reduce the amount of false positives registered by the network.
Totally, 12 classes were defined, plus an extra unlabeled

one for negative samples. For each class, 100 samples were produced.
The main variance between sets of images for the same label lie in different spatial positions of the hand with respect to the frame and rotations of the wrist.
It's important to note that a wider dataset for each gesure will increase variance, possibly increasing accuracy and the generalization capabilities of the final trained model.
The labeling process has been done with *LabelImg*, a Python based graphic tool developed by Tzutalin [8] which allows for manual selection of one or multiple ROIs within each frame and label it according to a specific class. This process produces, for each frame, an output *.xml* file that embeds the coordinates of the region of interest and label of a specific frame. At the end of this process, it will have produced exactly one *.xml* file for each sample in the dataset. The negative samples will have and empty *.xml* file, generated by pressing *Verify Image* on LabelImg's GUI.

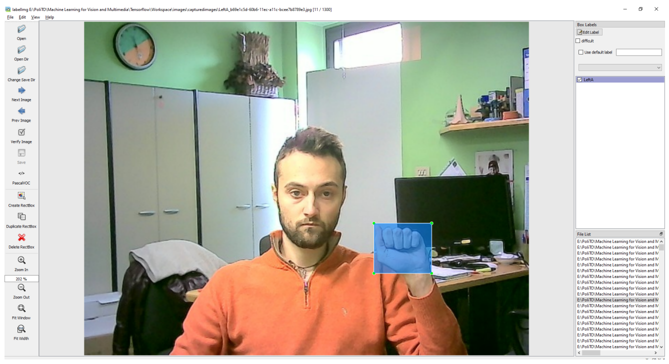


Figure 1. The LabelImg interface after drawing a bounding box around a gesture.

The final dataset thus contains 1300 images representing the same person, on the same background, doing a total of 12 different gestures. It has been split into two partitions, with 25% for validation and 75% for training.

## 3. Methods

The approach originally considered for the creation of an input interface for Blender was to build and train a RNN able to recognize actions from a video input. This however proved to be extremely inadequate in terms of usability, since repeating an action multiple times was the only way to produce incremental transformations through the interface. Moreover, the task of generating and labeling a custom dataset for such approach would have had a quite significant impact on the time available for the project's development. For this reason this approach was ultimately discarded.
The second approach, which was later adopted, was to train an Object Detection network able to map continuous static input gestures to transformations in the 3D scene of Blender. The choice of using the SSD MobileNet V2 FPNlite network arised from its inherent state of the art characteristics: a very fast mean input processing time of 22 ms, which suits perfectly the need to process video at 30 FPS from a webcam, which generates an image every 33.3 ms, and its very high mAP score of 22.2 over the COCO 17 dataset on which it was initially trained, as per the work of Yu *et al.* [9] (at *models/research/object_detection/g3doc/tf2_detection_zoo.md*).
This approach moreover allowed for the exploitation of transfer learning to retrain a pre-existing model, improving considerably the time required for training.

### 3.1. Architecture

The chosen model was originally developed by Ghiasi *et al.* [1] and aimed at improving over current hand-crafted CNNs, defining a better architecture of feature pyramid networks for object detection, called NAS-FPN. The model's architecture has been later improved in the work of Sandler *et al.* [7], which will be summarized in the following subsection. The main changes which have been applied to such model are the removal of the *random_horizontal_flip* data augmentation layer and the reduction of total training steps, justified by the fact that the dataset used is much smaller with respect to COCO 17 and thus boasts less features to be learnt.
The basic building block introduced in Sandler *et al.*'s paper is a bottleneck depth-separable convolution layer with residuals, whose structure is shown in Figure 2.

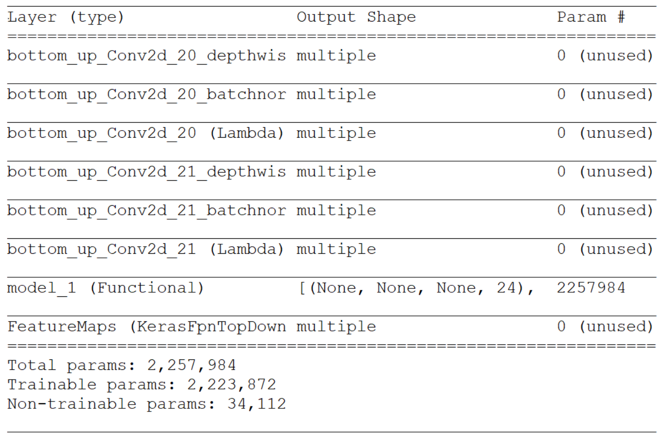
| Input | Operator | Output |
|---|---|---|
| $h \times w \times k$ | 1x1 conv2d , ReLU6 | $h \times w \times (tk)$ |
| $h \times w \times tk$ | 3x3 dwise s=$s$, ReLU6 | $\frac{h}{s} \times \frac{w}{s} \times (tk)$ |
| $\frac{h}{s} \times \frac{w}{s} \times tk$ | linear 1x1 conv2d | $\frac{h}{s} \times \frac{w}{s} \times k'$ |

Figure 2. Bottleneck residual block transforming from k to k' channels, with stride s, and expansion factor t.

The general architecture of the model contains an initial fully convolutional layer with 32 filters, followed by 19 residual bottleneck layers, as shown in Figure 3. The kernel size is $3 \times 3$, and dropout and batch normalization layers are used during training.
This architecture is then wrapped between six *bottom_up_conv_2d* layers and a *FeatureMap* layer, as shown in Figure 4. As it can be seen, this version of the model features $2,223,872$ trainable parameters and $34,112$ non-trainable ones.

| Input | Operator | $t$ | $c$ | $n$ | $s$ |
|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | |

Figure 3. The full model's architecture. Each line represents n repetitions of the same layer. All layers feature the same number of channels c. The first sequence layer has stride s, whilst the others have stride 1.


```
Layer (type)                 Output Shape              Param #
=================================================================
bottom_up_Conv2d_20_depthwis multiple                  0 (unused)

bottom_up_Conv2d_20_batchnor multiple                  0 (unused)

bottom_up_Conv2d_20 (Lambda) multiple                  0 (unused)

bottom_up_Conv2d_21_depthwis multiple                  0 (unused)

bottom_up_Conv2d_21_batchnor multiple                  0 (unused)

bottom_up_Conv2d_21 (Lambda) multiple                  0 (unused)

model_1 (Functional)         [(None, None, None, 24),  2257984

FeatureMaps (KerasFpnTopDown multiple                  0 (unused)
=================================================================
Total params: 2,257,984
Trainable params: 2,223,872
Non-trainable params: 34,112
```

Figure 4. The wrapped model's architecture, complete with parameters.

### 3.2. Hyperparameters

The loss used for training is split into four elements: classification, localization, regularization and total loss. Classification loss refers to the model's ability to correctly predict an object's class and uses as loss function a weighted sigmoid focal, with gamma equal to 2 and alpha equal to 0.25. Localization loss refers to the ability to find the position of the bounding box of an object inside the input image and uses as loss function a weighted smooth L1 function. Regularization loss is an additional loss generated by the L2 regularization function, which gets added to the network's loss and used to optimize over the sum of the other two. This method is used to help the optimization algorithm to generalize better over the dataset. The total loss represents the sum of these three previous losses and gives a top-down view of the model's training performance.
The model's live performance has then been evaluated through the ModelTest.ipynb notebook, checking the predicted labels superimposed over the input frames.
To boost the model's prediction accuracy, HSV skin tone thresholding has been applied to remove most false positives and stabilizing the model's performance in the wild. The results were extremely encouraging and thus this pre-processing step has been mantained in the final evaluation script.

## 4. Experiments

All the models have been trained onto a machine equipped with an *Intel i5-11600K* CPU, 32 GB of DDR4 3600 MHz RAM and accelerated through an *NVidia GeForce RTX 2060 6GB* GPU.

### 4.1. Training

The first training has been done over 20000 steps, with a batch size of 8, totaling 164 epochs, and required over two hours to complete: this produced a model with a final Total Loss of 0.12568491, still too high to provide stable predictions. Testing this checkpoint in fact proved to be unfeasible, as it was not able to recognize almost anything at all during the evaluation phase. The training has then been resumed for another 20000 steps, requiring another two hours and a half. The final Total Loss came out to be 0.09876089, which is a great improvement over previous iterations. However, testing the model live immediately gave rise to bounding box detection issues and general inaccuracy of the predictions, without much generalization capabilities. Moreover, since the network during training internally performed horizontal flips for data augmentation, the left and right hand gestures got often swapped or overlapped. To boost the accuracy of the network, its input gets thresholded using an HSV profile, customizable to better suit each user. The hand swapping issue, has been solved by retraining the model from scratch whilst removing the horizontal flip data augmentation layer, this time with a batch size of 16, totaling 656 epochs. This last training required only four hours and a half to complete, suggesting that the GPU's hardware characteristics were not well exploited previously. The training statistics for this last model are shown in Figure 5.

### 4.2. HSV thresholding

The role of the HSV thresholding preprocessing step plays an impactful role on the model's ability to process the input frames. Despite the fact that the dataset contains images with hands in different spatial positions over a complex background, the model sometimes produces FP,

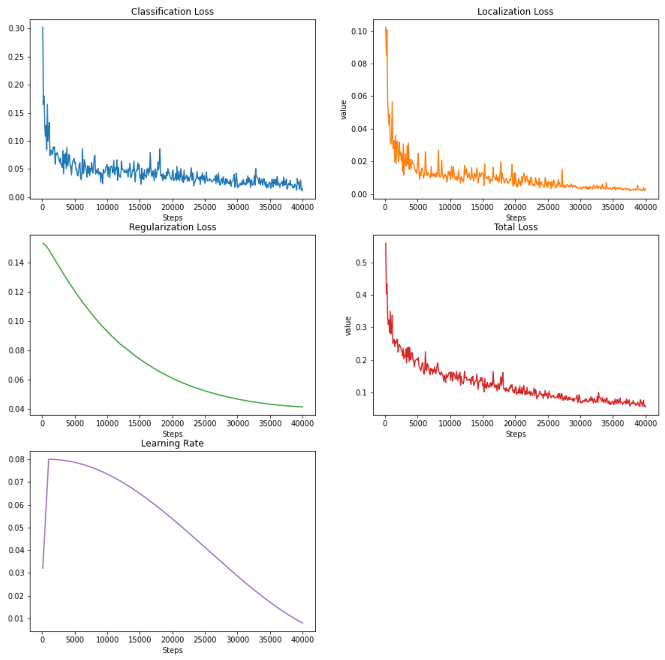


Figure 5. All losses and learning rate statistics for the 40000 steps model trained without the horizontal flip data augmentation layer. From top left to bottom right: Classification Loss, Localization Loss, Regularization Loss, Total Loss and Learning Rate.

detecting a gesture where there is none. This may be due to the fact that the whole dataset contains the same static background, or that some of the features learnt by the network are edge bound and some objects happen to fit partially those feature requirements. However, after a thresholding and opening transformation, most of the background contains no data, as shown in Figure 6, and thus gives to the model less room for error in evaluating gestures. To prove its beneficial effects, an ablation study has been conducted, whose results can be seen in Table 1. Some classes benefit very little from the thresholding procedure, like *LeftA* and *RIndexFront*. Some obtain a great improvement, like *RThumbBack* and *RIndexRight*, while most of the classes show a 2-5% improvement on average. The *LeftO*, *LeftV* and *RIndexDown* get an average 2% worsening, which however gets outweighted by the improvements in detecting the other nine classes. As such, the preprocessing step has been kept as part of the model.

### 4.3. Results

All the performance metrics hereby presented are based on the work of Padilla *et al*. [6].
The precision to recall curves for the 12 classes have been computed for two different values of Intersection Over Union percentages: 50% and 80%. This has been done as a way to provide insight over the model's stability in correctly localizing a gesture.

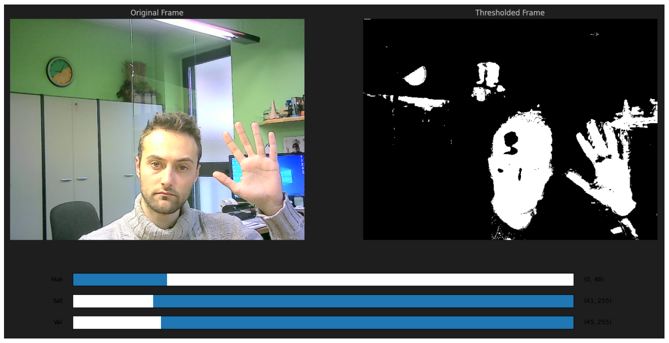


Figure 6. The effect of thresholding, as seen thorugh the *CreateHSVProfile.py* script.

In the $IOU = 50$ curves, shown in Figure 7, it's clear to see that the model is able to yield exceptional precision over all the classes with respect to the recall, with the *LeftA* and *RIndexRight* boasting a AP score of $1.0$ and most of the other classes following closely to the same result. The two worst performing classes are *RIndexUp* and *RThumbBack*, respectively with an AP score of $0.9217$ and $0.9242$, which are still extremely good results. All the *PASCAL* AP scores are listed in Table 1.

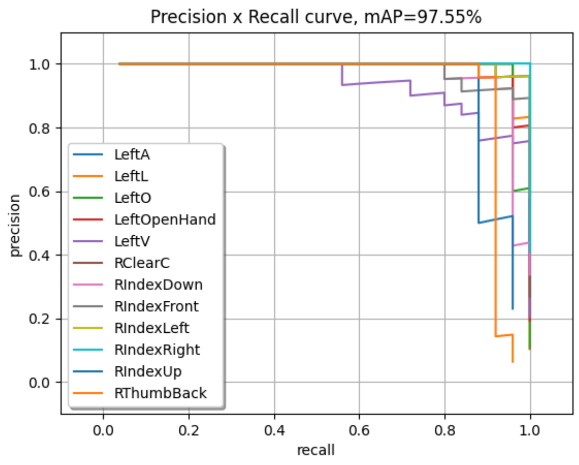


Figure 7. The precision to recall curves for all the 12 proposed gesture classes, with IOU = 50 and HSV thresholding.

The steeper requirement of an $IOU = 80$ value is reflected onto the precision to recall curves shown on Figure 8. The best performing class is *LeftOpenHand*, which yields an encouraging AP value of $0.8517$, suggesting for it to be the most stable class. However, most of the other classes make the model fall short of the expectations, with only six other classes boasting an AP score sufficiently close or greater than $0.4$. The two worst performing classes are *RIndexRight* and *RThumbBack*, both with a AP score close to $0.0240$, suggesting that these two classes tend to be harder to correctly localize and detect for the model.

| AP scores | | | | |
|---|---|---|---|---|
| Classes | $AP_{50}$ with HSV | $AP_{80}$ with HSV | $AP_{50}$ raw | $AP_{80}$ raw |
| LeftA | **1.0** | 0.4105 | **1.0** | **1.0** |
| LeftL | **0.9933** | 0.5018 | 0.8775 | 0.7950 |
| LeftO | 0.9843 | 0.3903 | **1.0** | 0.7963 |
| LeftOpenHand | **0.9890** | 0.8517 | 0.9600 | 0.4324 |
| LeftV | 0.9453 | 0.1717 | **0.9855** | 0.6535 |
| RClearC | **0.9969** | 0.2654 | 0.9892 | 0.6420 |
| RIndexDown | 0.9694 | 0.6183 | **0.9938** | 0.9087 |
| RIndexFront | **0.9846** | 0.1852 | 0.9823 | 0.5765 |
| RIndexLeft | **0.9969** | 0.4494 | 0.9699 | 0.7648 |
| RIndexRight | **1.0** | 0.0242 | 0.6334 | 0.1389 |
| RIndexUp | **0.9217** | 0.5382 | 0.8147 | 0.4875 |
| RThumbBack | **0.9242** | 0.0240 | 0.3184 | 0.2426 |

Table 1. The AP scores among the 12 classes and both IOU values. The highest AP scores for each class are in bold.

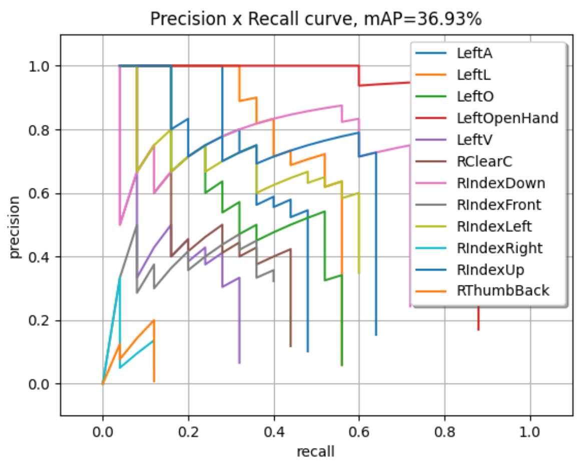


Figure 8. The precision to recall curves for all the 12 proposed gesture classes, with IOU = 80 and HSV thresholding.

Over all the classes, the model boasts the following *COCO* metrics. The $AP_{50}$ has a value of $0.9757$, in line with its relative *PASCAL* AP metric. The $AP_{75}$ has a value of $0.5704$, whilst the overall $AP$ has a slightly smaller value of $0.5645$. This skew between the *COCO* and *PASCAL* metrics suggests that the model's performance is promising and that the $IOU = 80$ value is a requirement too steep to be satisfied as per the current architecture of the network. Morever, the proposed set of gestures proved to contain common features between classes, as the number of FP explodes as soon as we try to consider lower confidence predictions.

## 5. Conclusions

The process of producing and understanding the model presented in this paper has been multi-faceted and long, however it also proved to be a very formative experience. The generation of the dataset has been a quite daunting task, requiring multiple days of work just for capturing and labeling. Moreover, the expansion of the dataset to include also negative samples to feed to the network resulted into a better generalization of the gesture features, reducing by a significant margin the false positives due to predicting gestures where there are none. In fact, one of the most common pitfalls in these kinds of applications is the false prediction of a hand gesture on the user's face, which in our case has been mitigated quite well.

Setting up the *pipeline.config* file proved to be a challenging task too, due to the required installation of the Tensorflow Object Detection API, which brought in many versioning issues deriving from Tensorflow 1 and 2 incompatible calls among the *Tensorflow/models* submodule's scripts. In the end, parts of those scripts had to be modified in order to account for updates in Tensorflow's API.

The analysis of the *pipeline.config* file also gave insight about the inner workings of the model, opening the possibility for tinkering around hardware specific definitions to improve training performance (like the batch size, which has been found to boast better performance at value 16) and removing components which were working against our needs.

The model's generalization capabilities over the single-person dataset proved to be extremely promising when live-tested onto different people. The model expects

a certain shape and orientation, thus incorrect gestures due to different individual's interpretation still get unrecognized, though gestures which try to resemble more the original dataset end up being recognized with no issue and a high confidence of around 90%. This requires some of effort from the user's side to work, however it's believable for it to be a quite encouraging result given the amount of data available. This aspect could be improved by generating a bigger dataset with higher user variance, especially in terms of hand morphology and skin tone, and consequently training for a higher number of steps.
It's important to note how a thorough study of a new gesture set could improve the model's performnace in future works, as some of the gestures shared morphological features which caused the model to produce more FP than expected. The thresholding external preprocessing step, originally implemented following Nikam *et al*. [5]'s work as a *YCbCr* thresholding, proved to boast low masking capabilities with respect to skin tone. It has later been changed to the current *HSV* thresholding script, as allows for more precise masking for *in-the-wild* environments.
In conclusion, the results of this paper are to be considered significant, in that they provide insight over the techniques and approaches to be used to produce gestural input interfaces through the exploitation of state of the art object detector neural networks.

## References

[1] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection, 2019.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[4] Ujjal Mavinkurve and Devansh Verma. Eg-easy: Design and testing of blender-based tool to teach projections in engineering drawing. In *2016 IEEE Eighth International Conference on Technology for Education (T4E)*, pages 250–251, 2016.

[5] Ashish S. Nikam and Aarti G. Ambekar. Sign language recognition using image based hand gesture recognition techniques. In *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, pages 1–5, 2016.

[6] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3), 2021.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[8] Tzutalin. Labelimg. Free Software: MIT License, 2015.

[9] Hongkun Yu, Chen Chen, Xianzhi Du, Yeqing Li, Abdullah Rashwan, Le Hou, Pengchong Jin, Fan Yang, Frederick Liu, Jaeyoun Kim, and Jing Li. TensorFlow Model Garden. `https://github.com/tensorflow/models`, 2020.

[10] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.