Grad Cert Intelligent Vision System
Continuous Assessment – Object Detection

Object Detection for Esports Analytics

Computer Graphics Detection using Mobilenet v1 SSD

Team Members:

Chan Kan Hei (A0198512Y)
Guo Xiang (A0198533U)
Li Jing Meng (A0198484J)

Background

In this project, we will train an image object detection model to detect computer graphic objects in popular esports computer games.

We are interested in this topic as object detection has been used in many real life competitive sports already such as soccer. Using object detection as input, sports teams and betting companies can analyse the match better. For example, one can track the distribution of players on pitch, movements of balls and changes in formations. However, such techniques have not been used as much in Esports.

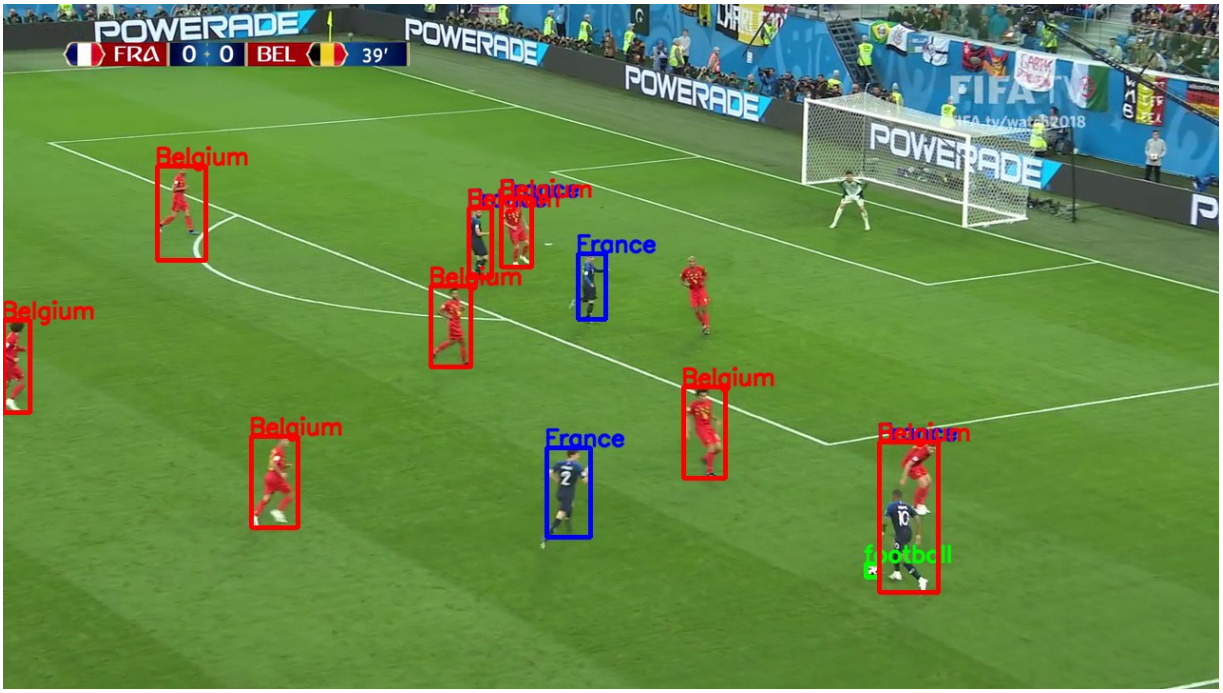


Fig1.Object detection in football game

Esports are getting more and more popular in recent years. Competitive games like League of Legends, PUBG, CS Go and Dota attract millions of players and audience all over the world. We believe that this type of vision systems will be useful in esports. It will help game analyst and online streaming channels(youtube, twitch) provide real time analytics. It can also help team to understand pro-player's playing style and other teams' strategy. We believe such system will actually be easier to train and have better detection performance because in-game graphics are generated by computer drawing. Thus, unlike real world pictures, they should be

more systematic in color and style, for example all houses in the game look similar as most likely they are copied and pasted. In-game real time analytics will also be more reliable as camera angles of the game are usually limited and fixed.

Fig2. Playerunknown's Battlegrounds

Originally, we planned to train an object detection model using League of Legends Game Play, see an annotation example below.

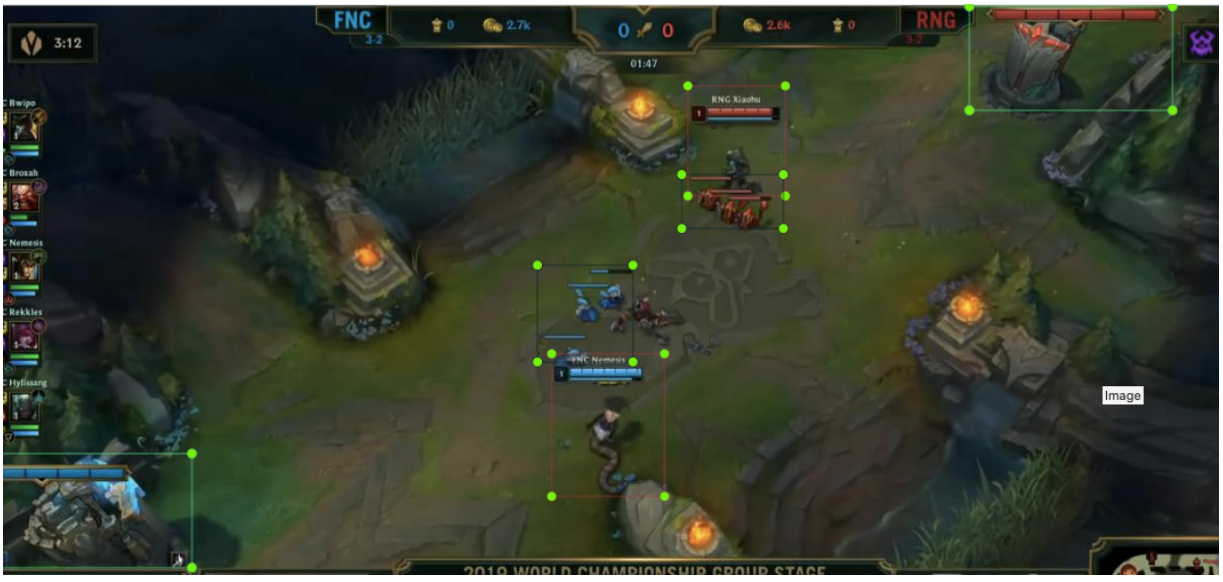


Fig3. League of Legends

Unfortunately, the training result is not good. Lots of the predicted boxes are actually empty. This is probably because of our dataset is too small (only ~500 images). We suspect that this is also because the original mobilenet model was trained on PASCAL VOC dataset. PASCAL VOC are real life pictures with bigger objects on the image, for example, aeroplane, dog and bus etc. Considering the limited time, we are not able to annotate large enough dataset or retrain the mobilenet. Thus, we decided to look for a game that is more similar to the original trained dataset. This maximises the learnings from original model and the feature extraction layers will be more relevant to our problem. We believe that will give us better detection performance.

Fig4. Object detection in League of Legends

Mobilenetv1 SSD Inner Workings

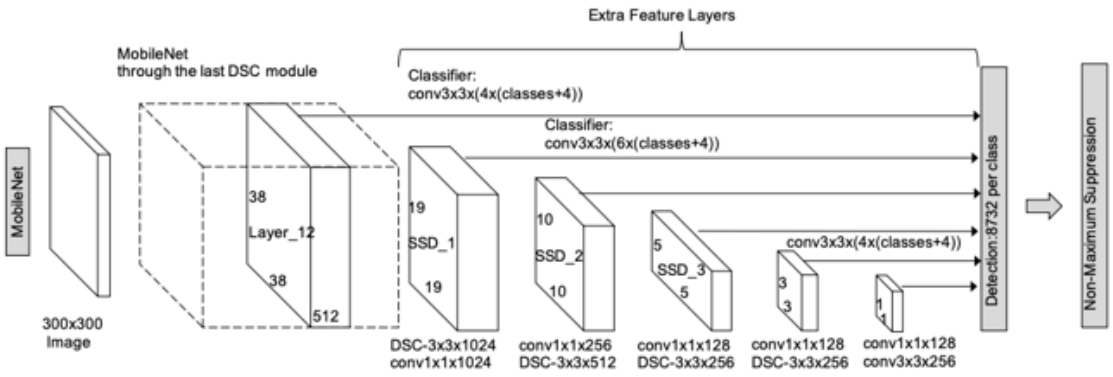


Fig5. The architecture of mobilenet ssd

In a nutshell, Mobilenet SSD uses MobileNet as feature extractor then put tons of predicting boxes on the features maps and finally predict the objects using convolutions. The reason for using mobilenet instead of VGG/inception net is because of its fast computations with depthwise and point-wise convolutions. SSD performs object localization and classification are done at the same time. So both architectures connect together to give a fast object detection model. It trades accuracy for speed.

Mobilenet

MobileNet is based on a streamlined architecture that uses depthwise separable convolutions to build a lightweight deep neural network.

The depthwise separable convolution is consist of the depthwise convolution and the pointwise convolution. The very first one applies a single filter to each input channel then the pointwise convolution applies a 1×1 convolution to combine the outputs the depthwise convolution. In the Mobilenet, we utilise depthwise separable convolution in the Mobilenet as below,

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig6. The layers of mobilenet ssd

The depthwise convolution applies a single filter to each input channel then the pointwise convolution applies a 1×1 convolution to combine the outputs the depthwise convolution. To build very small, low latency models that can be easily matched to the design requirements for mobile and embedded vision applications, reducing the computational cost is necessary.

Standard convolutions have the computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

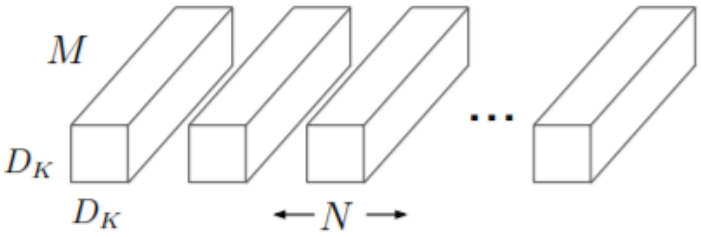
Depthwise separable convolution computational cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

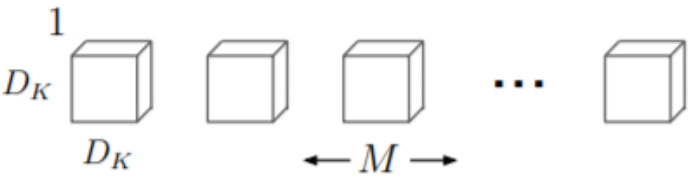
Reduction in computation of

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

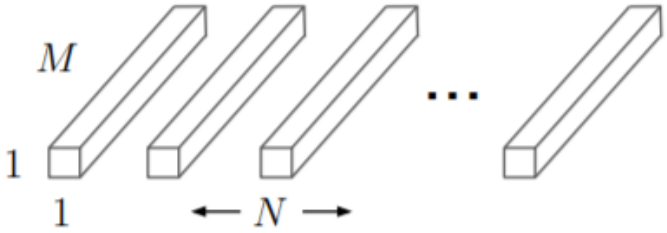
$$= \frac{1}{N} + \frac{1}{D_K^2}$$



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Fig7. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter

In summary, depthwise separable convolution needs a little computation comparison with standard convolution.

However, some applications require the model to be smaller and faster for many times. Therefore, to reduce the computational cost further, the author introduces the first parameter α which is called width multiplier to make a network uniformly thin each layer. The computational cost after introducing the parameter α :

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

Where α ∈ (0, 1].

Then the second hyper parameter called resolution multiplier ρ. In practice we set ρ by setting the input resolution. Then the computational cost with α and ρ is as below,

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$$

Where ρ∈ (0, 1].

After introducing two hyperparameters in the neural network, this model faster than the standard neural network more, but the accuracy may be reduced if α < 1 or ρ < 1 because of the thinning model or reducing the resolution of images. The comparison as below,

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

SSD

SSD stands for Single Shot Multibox Detector. Unlike RCNN Resnet that uses different networks for Regional Proposal and Object Detections, it performs object localization and classification in one go. For Mobilenet SSD, it uses Layer 12 which is a 38*38*512 layer, and add 5 more convolution layers to it. Each of these layers is called a feature map. The highest resolution feature map is 38*38 and the resolution decreases to 1*1 in the last layer. The key idea is higher resolution layer is responsible for detecting smaller objects(because it keeps more details) and lower resolution layer is responsible for detecting larger objects.

Predefined predicting boxes of different size and aspect ratios are placed on each of the position on the feature map. Convolution are performed to output the predicted x coordinate of the object centre, y coordinate of the object centre, width, height and the classes confidences.

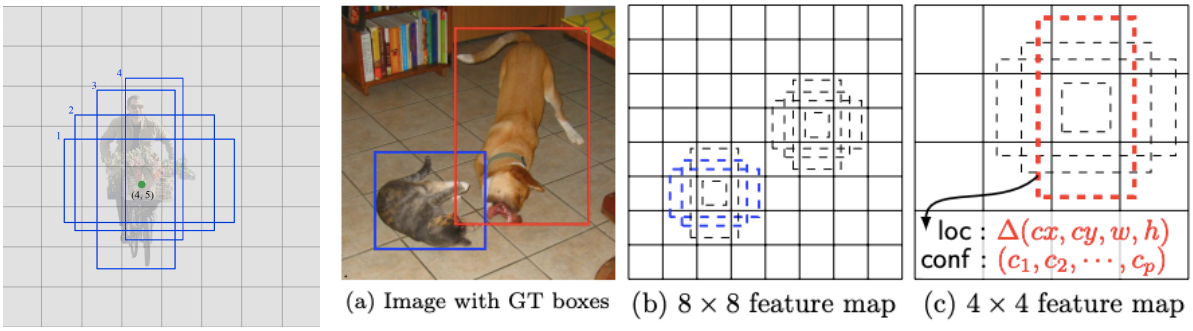


Fig8. (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8 × 8 and 4 × 4 in (b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories ((c1, c2, · · · , cp)). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1 [6]) and confidence loss (e.g. Softmax)

In the end, all predicting boxes' outputs are concatenated together. Predicting boxes of IoU with ground truth boxes lower than the threshold are removed. Label is given to the predicting box by choosing the class with highest confidence scores. Among the predicting boxes, those that are overlapping more than a certain IOU threshold are also removed(meaning that they are too close to one another, likely to be predicting the same object). If there is any ground truth box not matched with any predicting box, the highest IOU predicting box, even the IOU is lower than the threshold, is kept. Matched predicting boxes are considered positive and those not matched are considered negative. As there are far more negative(background) predicting boxes, we specify a ratio as hyperparameter to only keep part of the negative boxes output.

With all that processing done, on the training label we have the ground truth boxes centre x, centre y, width, height and the one-hot encoded class label; on the SSD output we have the predicting boxes centre x, centre y, width, height and the confidence scores. Backpropagation can kick in and the network learn to minimize between the two sets of values.

The loss function is as below, the first part is the classification loss and the second part is the localisation loss.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

Note that ground truth centres, widths and heights are rescaled to the same scale of that of the default predicting boxes for localisation loss calculations.

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^{w} = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^h}{d_i^h}\right)$$

Localization Loss

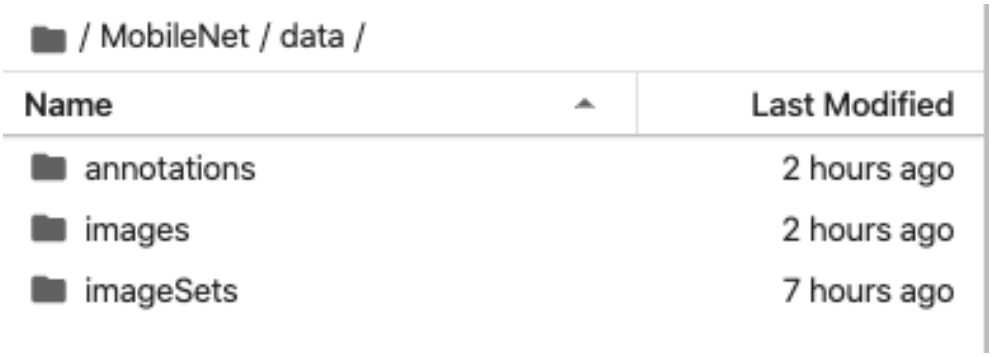
Design of APIs

Most of the codes are adapted and modified from https://github.com/ManishSoni1908/Mobilenet-ssd-keras During rewriting, ManishSoni replied us promptly on email for code clarifications. His work and help is highly appreciated.

Python file training API

Arguments in TrainTest notebook:

- img_dir_path: this path should contain 1 directory called “data”. Inside “data” there should be 3 sub-directories : annotations, images, imageSets. Folder annotations contains all the annotations xml. Folder images contains all the jpg images. Fold imageSets contains 2 files namely test.txt and trainval.txt. They contains the image file names for testing and training & validations respectively. Example of file structure below.



- weight_file: this path points to the pretrained mobilenet.h5 model you want to use. It can be downloaded from a lot of github repo. It is usually trained on PASCAL VOC or COCO datasets. We use the weights to load into the mobile net for feature extractions.

- checkpoint_directory: this path points to the directory you want to save your model and checkpoints

- epochs: the number of training epochs

SSD functional API

```
ssd_mobilenet.ssd_300(mode = 'training',
                      image_size=(img_height, img_width, img_channels),
                      n_classes=n_classes,
                      l2_regularization=0.0005,
                      scales=scales,
                      aspect_ratios_per_layer=aspect_ratios,
                      two_boxes_for_ar1=two_boxes_for_ar1,
                      steps=steps,
                      offsets=offsets,
                      limit_boxes=limit_boxes,
                      variances=variances,
                      coords=coords,
                      normalize_coords=normalize_coords,
                      subtract_mean=subtract_mean,
                      divide_by_stddev=None,
                      swap_channels=swap_channels)
```

mode: “training” or “inference”, in inference mode a simpler prediction decoding method is used, training mode can also be used during inference

image_size: the height, width, number of channels of input images, suggested to keep as original mobilenet as 300*300*3
n_classes: number of classes of objects to be detected (excluding background)
l2_regularization: degree of l2 regularization in all convolution layer
scales: list of scaling factors for anchor box size calculations, scaling factors should be increasing as layer number
aspect_ratios_per_layer: list of aspect ratios for anchor box size calculations
two_boxes_for_ar: True or False, True for adding 1 more special box for aspect ratio 1 as stated in original paper
steps: list of values to control the space between two adjacent anchor box center points for each predictor layer. This is to avoid overlapping at the beginning.
offsets: list of values of offsetting from edge of predictor layer .
limit_boxes: Whether or not you want to limit the anchor boxes to lie entirely within the image boundaries
variances : the variances by which the encoded target coordinates are scaled as in the original implementation
cords: format of the annotations of ground truth boxes, can be ‘centroids', 'corners', or 'minmax'
subtract_mean: the mean of each channel in the training dataset
divide_by_stddev: rescale the input data by stddev, default to be none
swap_channels: swap RGB to BGR which is the original color channel order in the SSD

Examples of functional call:

```
img_height = 300
img_width = 300
img_channels = 3
subtract_mean = [123, 117, 104]
swap_channels = True
n_classes = 6
scales = [0.1, 0.2, 0.37, 0.54, 0.71, 0.88, 1.05]
```

```
aspect_ratios = [[1.0, 2.0, 0.5],
                 [1.0, 2.0, 0.5, 3.0, 1.0 / 3.0],
                 [1.0, 2.0, 0.5, 3.0, 1.0 / 3.0],
                 [1.0, 2.0, 0.5, 3.0, 1.0 / 3.0],
                 [1.0, 2.0, 0.5],
                 [1.0, 2.0, 0.5]]
two_boxes_for_ar1 = True
steps = [8, 16, 32, 64, 100, 300]
offsets = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
limit_boxes = False
variances = [0.1, 0.1, 0.2, 0.2]
coords = 'centroids'
normalize_coords = True

model = ssd_mobilenet.ssd_300(mode = 'training',
                 image_size=(img_height, img_width, img_channels),
                 n_classes=n_classes,
                 l2_regularization=0.0005,
                 scales=scales,
                 aspect_ratios_per_layer=aspect_ratios,
                 two_boxes_for_ar1=two_boxes_for_ar1,
                 steps=steps,
                 offsets=offsets,
                 limit_boxes=limit_boxes,
                 variances=variances,
                 coords=coords,
                 normalize_coords=normalize_coords,
                 subtract_mean=subtract_mean,
                 divide_by_stddev=None,
                 swap_channels=True)
```

Code Structure

The code includes two scripts:

1. ssd300.py

    class keras_layer_AnchorBoxes: to make the model self-sufficient at inference time.

    class keras_layer_DecodeDetectionsFast: to decode the raw SSD prediction output.

    class keras_layer_L2Normalization: Performs L2 normalization on the input tensor with a learnable scaling parameter.

    class keras_ssd_loss: SSD loss.

    class ssd_batch_generator: A batch generator for SSD model training and inference which can perform online data agumentation.

    class ssd_box_encode_decode_utils: to compute IoU similarity for axis-aligned, rectangular, 2D bounding boxes, perform greedy non-maximum suppression and decode raw SSD model output.

    class mobilenet_v1: to build the mobilenet neural network

    class ssd_mobilenet: the mobilenet ssd architecture.

2. TrainTest.ipynb

    train the mobilenet ssd model by inputting images and test the images

Fig9. Performance on Playerunknown's Battlegrounds

Fig10. Performance on Playerunknown's Battlegrounds

Steps to Improve Performance

Mean Average Precision(MAP)

We used Mean Average Precision as performance metrics. AP (Average precision) is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1. And Mean Average Precision calculates the mean of AP among all the classes.

Performance

As the original SSD was trained on real life images, we decided to use the below aspect ratios according to our observations and heuristics instead of those from the original SSD:

aspect_ratios = [[0.8, 1.5, 0.5],
[0.8, 0.5, 1, 2.5, 1.5],
[0.8, 0.5, 1, 2.5, 1.5],

[0.8, 0.5, 1, 2.5, 1.5],
[0.8, 1.5, 0.5],
[0.8, 1.5, 0.5]]

And the MAP we got is 0.076. Below is the AP of each class:

{Player: 0.05032527532134931, Gun: 0.006314432989690721, Vehicle: 0.04088931886211395, House: 0.024925595238095243, Tree: 0.0005134788189987163, Plane: 0.33316905524051443}

To improve the performance, we decided to use the aspect ratios from the original SSD300 as below:

aspect_ratios = [[1.0, 2.0, 0.5],
[1.0, 2.0, 0.5, 3.0, 1.0 / 3.0],
[1.0, 2.0, 0.5, 3.0, 1.0 / 3.0],
[1.0, 2.0, 0.5, 3.0, 1.0 / 3.0],
[1.0, 2.0, 0.5],
[1.0, 2.0, 0.5]]

And the MAP increased to 0.092. Below is the AP of each class:

{Player: 0.04821870773043822, Gun: 0.00030665996575087487, Vehicle: 0.058277983729790955, House: 0.002294136950958477, Tree: 0.003332601089190914, Plane: 0.4410769687621884}

Conversion to OpenCV

Unfortunately, after all the trial and error, in the end we are still not able to read our converted model in OpenCV using cv.dnn.readNetFromTensorflow API. We can see that there are many similar discussion on OpenCV Developer Forum, we believe that the support of OpenCV reading tensorflow network is still not very robust. From our research, most of the time it is caused by 1. Keras, tensorflow, OpenCV version compatibility issue), 2. Certain operations of tensorflow is not understood by OpenCV. We tried most of the suggested solutions online. We are regretted that we are not able to find a solution by the submission deadline However, we detailed our steps below as reference.

1. Keras Model Training

We trained our keras model as usual, in the end, we have a h5 model file.

2. Keras model to Tensorflow model conversion

OpenCV does not understand h5 directly. We need to generate the protobuffer and protobuffer text file for OpenCV. As keras is using tensorflow backend, we can leaverage tensorflow's python library (tensorflow.python.tools import freeze_graph) to obtain a frozen graph.

3. Optimize the frozen graph for inference

The generated graph contains many unnecessary nodes, using "tensorflow.python.tools import optimize_for_inference_lib" we can remove those. At the end we have a protobuffer file and a protobuffer text file like below. The file details the node name, operation and its relationship with other nodes.

```
node {
  name: "input_6"
  op: "Placeholder"
  attr {
    key: "dtype"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "shape"
    value {
      shape {
        dim {
          size: -1
        }
        dim {
          size: 300
        }
        dim {
          size: 300
        }
        dim {
          size: 3
        }
      }
    }
  }
}
node {
  name: "identity_layer_5/Identity"
  op: "Identity"
  input: "input_6"
  attr {
    key: "T"
    value {
      type: DT_FLOAT
```

You can use tensorboard to examine the network visually using the protobuffer file. That shows the pb and pbtxt file are valid for tensorflow.

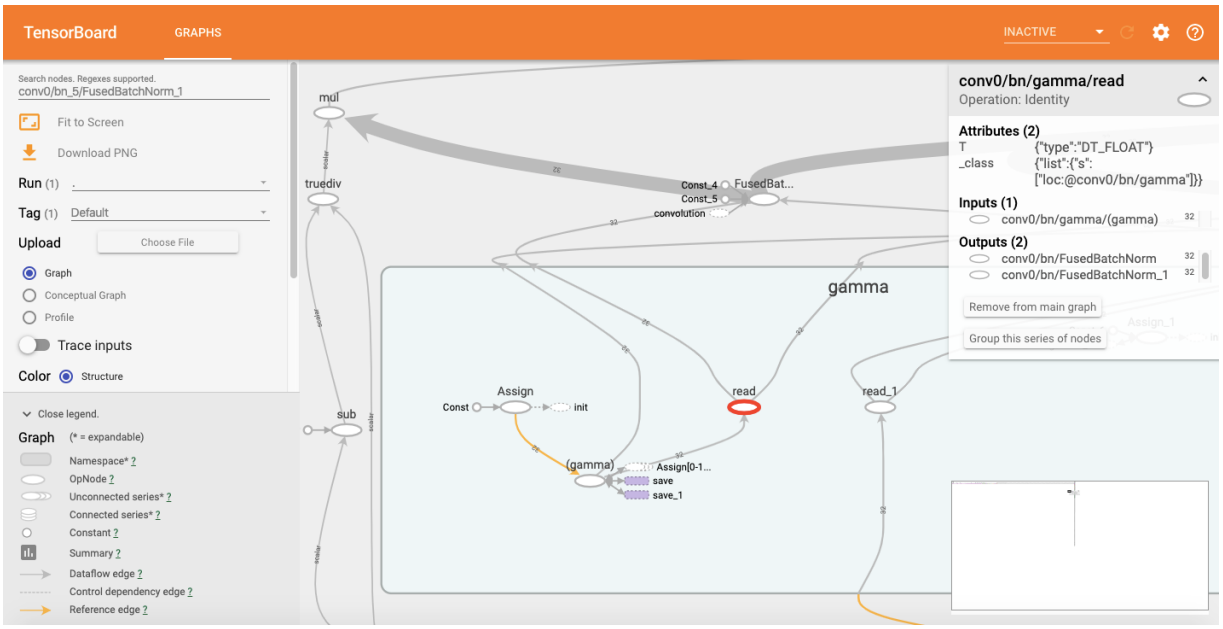


Fig11. Tenforflow graph

4.Loading into OpenCV

Finally we load the pb and pbtxt file into OpenCV.

net = cv.dnn.readNetFromTensorflow(MODEL_PATH+'/opt_test.pb',

MODEL_PATH+'/stripped_test.pbtxt')

We faced several types of errors, one is StridedSlice operation, it is reported that strided slice with step -1 is not supported in OpenCV but it is supported in tensorflow. To debug, we go indepth into the code of tf_importer.cpp and find the hightlighted line throwing the error.

```
else if (type == "StridedSlice")
{
    CV_Assert(layer.input_size() == 4);
    Mat begins = getTensorContent(getConstBlob(layer, value_id, 1));
    Mat ends = getTensorContent(getConstBlob(layer, value_id, 2));
    Mat strides = getTensorContent(getConstBlob(layer, value_id, 3));
    CV_CheckTypeEQ(begins.type(), CV_32SC1, "");
    CV_CheckTypeEQ(ends.type(), CV_32SC1, "");
    CV_CheckTypeEQ(strides.type(), CV_32SC1, "");
    const int num = begins.total();
    CV_Assert_N(num == ends.total(), num == strides.total());

    int end_mask = getLayerAttr(layer, "end_mask").i();
    for (int i = 0; i < num; ++i)
    {
        if (end_mask & (1 << i))
            ends.at<int>(i) = -1;
        if (strides.at<int>(i) != 1)
            CV_Error(Error::StsNotImplemented,
                     format("StridedSlice with stride %d", strides.at<int>(i)));
    }
```

Some developer suggested that we can rename the ops to Identity as workaround, while understanding that identity operation is not the same as stridedSlice, we gave it a try.

The replacement removed the strided slice error but we see another one below. After researching, some said FusedBatchNorm is not compatible in OpenCV while some said the graph definition is broken during pb creation. We are unable to move forward from there as these operations are not directly used in our keras network definitions. They are implicitly added by tensorflow during conversion and we have no idea how the conversion work and how OpenCV handle the operations.

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-29-496ef7a8c80d> in <module>
----> 1 net = cv.dnn.readNetFromTensorflow(MODEL_PATH+'/opt_test.pb', MODEL_PATH+'/stripped_test.pbtxt')

error: OpenCV(4.1.1) /io/opencv/modules/dnn/src/tensorflow/tf_importer.cpp:582: error: (-2:Unspecified error) In
put [conv0/bn_5/gamma] for node [conv0/bn_5/FusedBatchNorm_1] not found in function 'getConstBlob'
```

Fig12. Error when converting tensorflow to opencv

Findings and Conclusions

From the performances we can see that the model was slightly improved when using the original aspect ratios from SSD300. Also, we can see that the model perform poorly on these two classes: Gun and Tree. After investigating our dataset, we found out that for guns, they mostly appear in our dataset when they are carried by a player and therefore most of their parts are often blocked. For trees, they often appear in the background of an image, and is difficult to differentiate from the background.

To further improve the performance of our model, there are several things we need to do in the future.

Firstly, enlarging our dataset. Currently, the number of images in our whole dataset is approximately 730, which is not enough for the computer to learn.

Secondly, training the Mobilenet with our own dataset. The Mobilenet we used currently is trained on PASCAL VOC, which is a real life image dataset and is different from our dataset, which is from a video game.

Lastly, try more different parameters such as other sets of aspect ratios and scales, and train the model for more epochs. Due to time constraint, we only trained the model for 250 epochs and we did not try many different sets of parameters. To further improve our model, we definitely need to spend more time and effort on tuning the model.