# Machine Learning Capstone Project[1]

Project: Real time person and vehicle detection of UAV images on TX2
Jun Zhang
November 18th, 2017

## I. Definition

### Project Overview

In recent years, autonomous UAVs have been widely deployed for inspection, surveillance, search-and-rescue, traffic monitoring and infrastructure inspection[1-3]. As UAV application becomes widespread, a higher level of autonomy is required to ensure the safety and operational efficiency. To achieve this, the real-time visual object detector is necessary for UAV systems. However, like many other mobile systems, the computation capability usually of UAVs are typically limited. In this capstone project, we will focus on object detection in a UAV system with NVIDIA TX2 and we are only focusing two objects: person and vehicle. We are going to train and evaluate the designed detector with both Pascal VOC and UAV123 datasets.

### Problem Statement

Visual object detection is a classic problem in the computer vision. However, it is even more challenging to perform the detection task in UAV systems due to the top-down view, distortion due to UAV motion and real-time requirement with limited hardware resource. On the other hand, the objects interested in UAV system are different compared to images from cameras on the ground. There are a few other work either focusing on a single object detection such as pedestrian detection [7] or vehicle detection  [10]. Those two kinds of objects are particularly interesting in UAV applications. Take an example of the company Aeryon Labs where I works. We design and manufacture small UAVs for video surveillance and inspection purposes for both military and public safety sector. And from the feedback of our customers, they uses our UAVs to locate and track Person and Vehicles a lot. We have use designated object tracking and moving object tracking features, but do not automatically detect what kind of objects in the video. Therefore, a automatic object detection for those two objects will be very convenient for

[1] Project code is hosted here: https://github.com/darknight1900/MobileDet

the customers using our UAVs. After an object of interest being detected, we can just use our current object tracker to track this object over time. Therefore a detector which can offer good detection accuracy and can also run real-time on a powerful embedded platform such as TX2 will be quite interesting for us.

## Metrics

In the computer vision field, an object detector is typically evaluated with metric mean average precision (mAP)[6]. However, since there are only two objects of interest, we will simply evaluate precision and recall for each object as our evaluation metrics. In addition, the UAV system typically is equipped with limited hardware, detection accuracy will be not our solo target. We will also evaluate the speed, hardware consumption of the detector to have a more thorough evaluation.

# II. Analysis

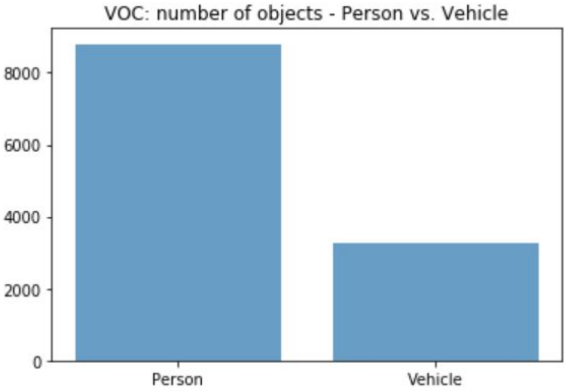
## Data Exploration and Exploratory Visualization

There are two datasets used in this project: Pascal Visual Object Classes (VOC)[6] and UAV123[11].
The goal of this project is to design a Person and Vehicle object detector for UAV images which can run real-time on a TX2 platform. However, most public object detection or classification datasets are captured from handheld device such is quite different from images captured by cameras mounted on UAVs. The closest dataset we found is UAV123 dataset. This is a very new dataset for object tracking released in 2016. The original dataset contains a total of 123 video sequences and more than 110k frames. This dataset has 123 annotated 1280x720 video sequences captured from a low-altitude aerial UAV. We have to do some manually filtering so that we can use this dataset to train the designed Person and Vehicle object detector. After the filtering, we only have around 3000 images left, which is relatively small. Therefore, we first train our model with VOC dataset and use the transfer learning technique to retrain the model with filtered UAV123 dataset.

### VOC Dataset

VOC (2007 & 2012) dataset contains around 20,000 images and most of the images are captured by handheld devices. A few randomly selected images from VOC dataset are shown as below.

There are 20 objects in the original VOC datasets. As we are only interested in ‘Person’ and ‘Vehicle’ objects, all the images which do not contain those two objects have been removed and also the labels for ‘bus’, ‘train’ and ‘car’ have been changed to be ‘Vehicle’. After this filtering process, there are about 8000 samples of person objects and 3000 samples of vehicle objects, as shown in below figure.

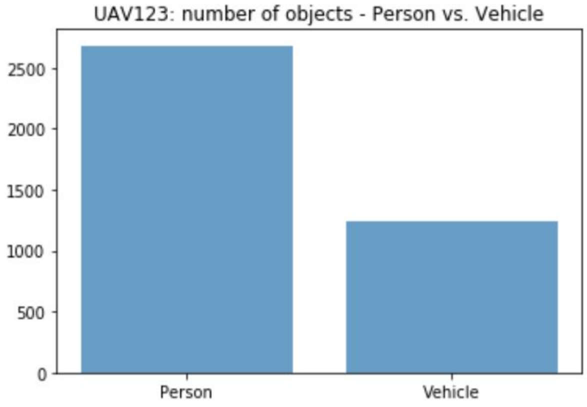


## UAV123 Dataset

Each UAV123 video clip is stored as a sequence of jpeg images. It has tracking labels for various kinds of objects such as ‘bike’, ‘bird’, ‘boat’, ‘car’, ‘building’, ‘trunk’ etc. A few randomly selected objects are shown as below.

For this project, we only keep the video with ‘person’ and ‘car’ and change the label ‘car’ to ‘vehicle’. One problem with this dataset is not all the objects are labeled. To use this dataset for object detection purpose, we only keep the images which all the ‘Person’ and ‘Vehicle’ objects are labeled. Moreover, each for the object under tracking the bounding box sometimes are lost due to occlusion and we will also have to remove those images.

In addition, images from a video clip have a large amount of spatial redundancy. To avoid overfitting, we randomly select around 30% of the selected images as training data and randomly select another 30% from the remaining for validation and testing purpose. After this preprocessing stage, we generated an HDF5 file named “UAV123.hdf5” and below figure shows the number of each object inside train and validation dataset of this HDF5 file.



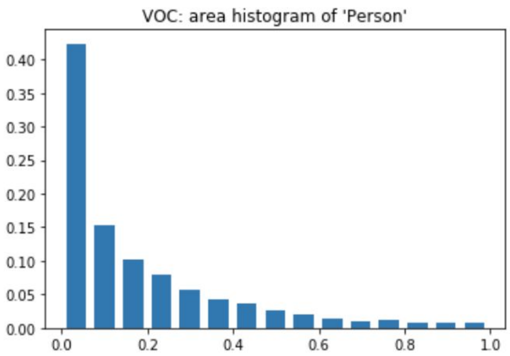
## VOC vs. UAV123

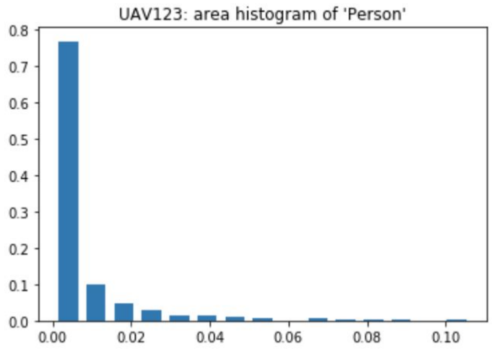
One of the most challenging parts of object detection from UAV imagery is the objects are typically way smaller than the objects from ground cameras. Below is a table comparing the mean normalize area size of each object. We can tell from the table that objects in the UAV123 dataset are way smaller compared to VOC dataset.

| | **Mean area of ‘Person’** | **Mean area of ‘Vehicle’** |
|---|---|---|
| **VOC** | 0.179 | 0.226 |
| **UAV123** | 0.008 | 0.011 |

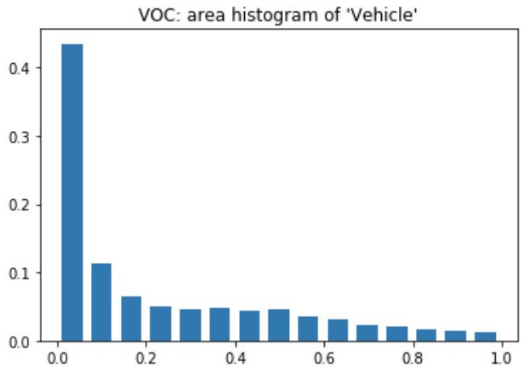
Below is the ‘Person’ and ‘Vehicle’ area histogram of VOC and UAV123 datasets. X-coordinates denotes normalized object area size and y-coordinates represents the percentage of the number of objects in each bin.

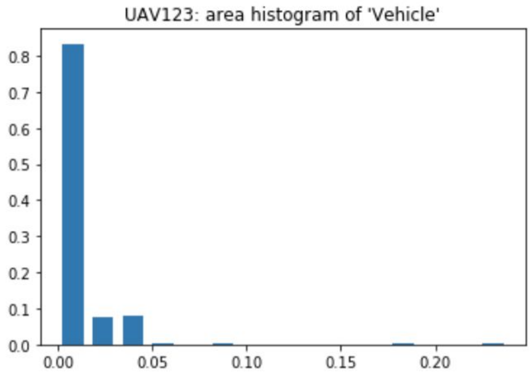
It is interesting to see that for the UAV123 dataset, over 75% of ‘Person’ objects are around 1% of entire image size, and 99% of ‘Person’ objects are smaller than 10% of the entire image. Whereas VOC dataset, only 40% of ‘Person’ objects are smaller than 10% of image size.





We observe the similar difference with ‘vehicle’ objects between VOC and UAV123 dataset.





## Algorithms and Techniques

In this project, we designed our object detector based on YOLOv2 (You Only Look Once) method. YOLOv2 is a deep learning based method to perform object detection.

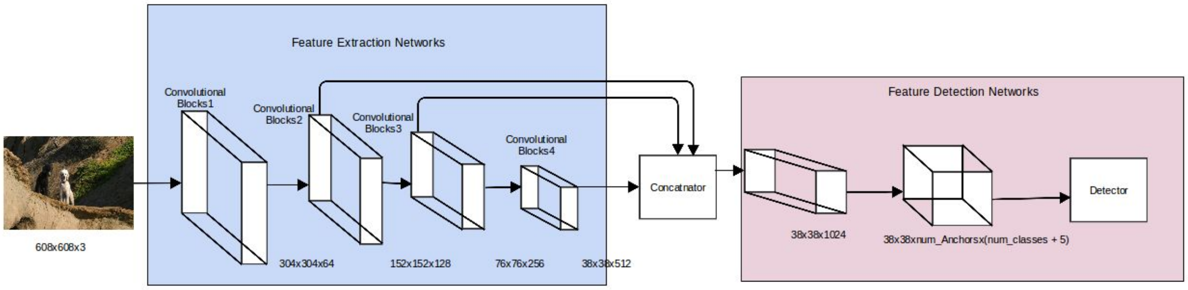
Deep learning has proven to be a powerful tool for image classification. For the ImageNet image classification competition, the ResNet 2015 has achieved remarkable 3.57% error which has outperforms human accuracy.

The traditional method of object detection will typically re-purpose image classifiers for detection. To detect a certain object, those methods will train a classifier for the objects and perform classification at various locations and scales of input images. For every single image, we need to test on thousands of locations to evaluate whether there is an object. As you can imagine, this type of method is very computation costly for large images and usually is unable to offer real-time performance on embedded platforms.

YOLOv2 is a method which only feeds the images to a convolutional networks which we call it feature extraction networks once and all the detection will be performed on the output of feature extraction networks. This method basically converts the object detection problem to a single regression problem, straight from image pixels to bound boxes coordinates and class/label probabilities. Compared to the previous method, this kind method is very efficient and offers real-time performance on desktop GPU.

However, using the original YOLOv2 might not work very well for images from UAV systems. There are five max-pooling layers interleaved with multiple convolutional layers in the original YOLOv2. After each max-pooling, the output size will shrink to half of the input. Therefore, the output feature from the last convolutional layer is only 1/32 ( $1/2^5$ ) of the original input image size. As we have seen in the previous sections, the size of objects of UAV datasets is way smaller compared to objects on images taken by ground cameras. Below is an image from the UAV123 dataset, the object size is around 0.4% of the image size. With five layers of max-pooling, the object might shrink to be a single dot or not even exists on the final feature map due to pooling operation.

To improve the the detection accuracy for UAV objects, we are going to make some necessary modifications to the original YOLOv2. The modified model structure is illustrated below.



First, we borrow some idea from SSD method [9]  and will add one extra scale of feature map from the earlier net layer for detection. Therefore, total three scales of features will be concatenated together and will be used for later detection. This extra scale of feature could help boost the detection accuracy for small objects with adding too much network complexity, as we are not adding new convolutional layers.

Second, we tailor down the feature extraction networks to only contain 4 max-pooling layers.
In the original YOLOv2, there are 5 convolutional network blocks, where each block contains a a few convolutional layers, batch normalization layers and relu layers. In the end of each block, there will be a max-pooling layer, which will shrink the image or the feature map size by half. To avoid shrink the small objects of UAV images too much, we removed the last convolutional block including its max-pooling layer. With this change, hopefully a small object might be present or even have multiple points on the features of the last convolutional block. This change

also helps hardware usage reduction as all the network layers between the last two max-pooling layers have been removed. However, the detection accuracy for large objects might be reduced.

Last, to further decrease the hardware usage, we will not use the original Darknet19 as backbone feature extractor. Instead, we will use MobileNet[14] as the feature extraction network. MobileNet uses the idea of depthwise convolution and could reduce the number of training parameters significantly without too much accuracy loss.

We will first train our detector with VOC dataset for around 150 epochs and then we will use the transfer learning idea to retrain the detector with the UAV123 dataset for another 30 epochs and we will evaluate the accuracy, speed and hardware usage on a TX2 platform.

## Benchmark

The benchmark model we are using the original YOLOv2 model with input dimension width, height and number of channels as 608x608x3. Our implementation is using a mix of Keras and Tensorflow. Keras model summary functionary will calculate how many trainable parameters for a deep learning mode. The original  YOLOv2 608x608x3 can offer more than 80% recall and precision accuracy (details will be discussed in latest sections) for both Person and Vehicle detection, however, it contains 50.6 million trainable parameters, which is quite a lot even for a powerful mobile platform such as TX2. We have tested the speed on original YOLOv2 on TX2 and it can only run about 1.8 frames per second, which is too slow the the real time application.

# III. Methodology

## Data Preprocessing

### VOC dataset

We used one script named ‘voc_to_hdf.py’ to combine VOC 2007 and 2012 detection datasets into one single HDF5 file. As we are only interested in ‘Person’ and ‘Vehicle’ objects in this project, we also removed all the images without those two objects and re-labeled ‘car’, ‘bus’ and ‘train’ to be ‘vehicle’. Details can be found in the mentioned script. To recreate the dataset, the VOC dataset has to be download and unzipped. And the path has to be provided to the script. The processed dataset in HDF5 format has also been uploaded to this google drive location.

### UAV123 dataset

For UAV123 dataset, we created one script named ‘uav123_to_hd5.py’ to create an HDF5 file to train our detector. The UAV123 dataset has 123 video sequences and was created originally for object tracking purpose. To use this dataset for ‘Person’ and ‘Vehicle’ object detection, we have to do a few necessary preprocessing. The selected folders and images information is stored as a few text files inside this tarball UAV123_selected.

First, we only keep the video sequences which labeled ‘Person’ and ‘Vehicle’ objects. Some video sequences have multiple persons and vehicle objects but this dataset only labeled a single of them. To avoid confusing the training process, we have to manually select images with single objects.

Second, there are lots of redundancies inside video clips and also some clips are longer than others. To avoid overfitting on a certain object, we only allow a maximum number of samples from each video sequence by randomly sampling and use the selected images and their corresponding bound boxes information as training/validation samples and use the remaining images as testing samples.

The processed dataset in HDF5 format has also been uploaded to this google drive location.

## Prior Anchor Boxes Generation

In the YOLOv2 model, a couple of prior anchor boxes with different aspect ratio and size will be provided to the model. On each cell of the final features for detection, the model will try to predict the location of an object, its label and the confidence inside each of those prior anchor boxes. Therefore, the anchor boxes have to be carefully selected.

Following the same idea from the YOLOv2 paper, we will use the k-means clustering algorithm to select 5 anchor boxes from each dataset and use those selected boxes for training and prediction. We are using a script named anchor_boxes.py to generate anchor boxes for each dataset. The input to this script is the HDF5 dataset from the previous section and the number of anchor boxes we want to have.

With configuration of model input 608x608, output feature map size 19x19 (with 5 max-pooling layer), 5 anchor boxes, the (width , height) of anchor boxes for each dataset are as below. The anchor box size is with respect to the final feature map 19x19, as the final detection will be based on feature map directly.

VOC dataset:

(1.3221, 1.73145)
(3.19275, 4.00944)
(5.05587, 8.09892)
(9.47112, 4.84053)
(11.2364, 10.0071)

UAV123 dataset:

(0.658624, 2.666625)
(2.444950, 7.431975)
(1.272862, 4.598098)
(1.902102, 2.050706)
(0.428755, 1.522414)

When training a model with 4 max-pooling layer (output feature map size 38x38), the anchor boxes size will also have to be multiplied by 2.

### Data preprocessing during training

During the training process, we will first resize all the images to be 608x608x3 and normalize them images into the range of [0, 1] so make the model converge faster. When resizing the images, their corresponding bound boxes will also need to be resized and normalized according. YOLOv2 method also requires the bounding box information to be converted from (xmin, ymin, xmax, ymax, class_label) to (x_center, y_center, box_width, box_height, class_label).

## Implementation

In our customized YOLOv2 detector, the input images will go through 4 convolutional network blocks and each block will output a couple of channels of feature maps. We are going to use 3 scales of feature maps (152x152, 78x78 and 38x38) for the detection purpose. Assume the number of feature maps for each scale are (n1, n2 and n3)

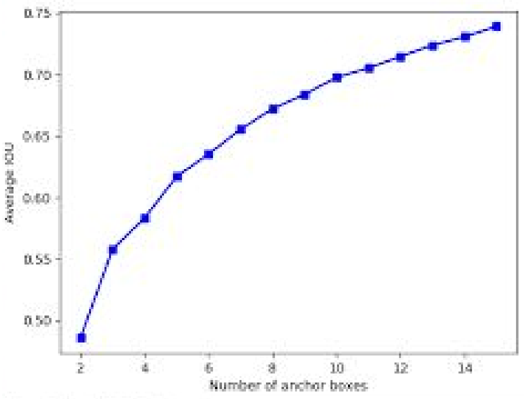
Those feature maps will be concatenated together to perform detection. To make the concatenation works, we will use the Tensorflow space_to_depth function. For n1 of 152x152 feature maps, we will convert them into 16*n1 of 38x38 feature maps. For n2 of 78x78 feature maps, we will convert them into 4 *n2 of 38x38 feature maps. Therefore, the final detection task will be entirely based on (16*n1 + 4*n2 + n3) of 38x38 features.

Those (16*n1 + 4*n2 + n3) of features will go through another layer of convolutional networks to be converted into a tensor with shape$38\,x\,38\,x\,B\,x\,(C\,+\,5)$. We can interpret the YOLOv2 method as a regression problem. Each image is divided into a 38x38 cell (or 19x19 if using the original YOLOv2 with 5 max-pooling), and within each cell YOLOv2 directly regresses to find B bounding boxes. For each of B bounding boxes, there are $(C+5)$ parameters, C is the probability of each classes, which is 2 in this project and 5 is the bounding box coordinates and its confidence probability, which is with in the range of [0, 1].
To fairly treat the large and small bounding boxes, the YOLOv2 method will not try to predict the absolute location of bounding box coordinates $(b_{x,}b_{y,}\ b_w,\ b_h)$ directly. Instead, the predictions are relative to the grid position and prior anchor boxes size for better performance. To map the predictions to the original image dimensions, below equations are used.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Where ( $c_x , c_y$ ) are the left corner coordinates of the cell, ( $p_h, \ p_w$ ) are the width and height of each prior anchor boxes and the $\sigma(x)$ is the sigmoid function.
During the training process, the ground truth bounding boxes coordinates in the image dimensions will be mapped to 38x38 feature map scale based above equations.

The prior anchor boxes are generated by using k-means clustering algorithm for each training dataset. Using direct Euler distance metric for k-means minizers errors for larger bounding boxes but not for smaller boxes. Therefore, in YOLOv2 method, intersection over union (IOU) is used as a distance metric. The IOU calculations are made assuming all the bounding boxes are located at one point. The detailed implementation of this is inside ‘anchor_boxes.py’ script. In theory, more prior anchors boxes will lead to better detection performance and the average IOU between ground truth bounding boxes will be also higher.
Figure below presents average IOU vs number of anchor boxes. We can see that when number of anchor boxes is 5 for VOC dataset, we already get the average IOU to be 0.61. A larger number of prior anchor boxes will result in large network size and complexity and will also increased training and interference time. Therefore, in this project, we use 5 anchor boxes, which is the same number as the original YOLOv2 implementation.



## YOLOv2 Loss Function

The YOLOv2 method uses one single loss function for both bounding box location and the classification of the object. The loss function is consist of three parts ( $loss_{coord}$ , $loss_{confidence}$, $loss_{classes}$ ), each of which is defined as following.

$$loss_{coord} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 \ + \ (y_i - \hat{y}_i)^2 \ + \ (w_i - \hat{w}_i)^2 \ + \ (h_i - \hat{h}_i)^2 \ ]$$

Where ( $x_i, y_i, \ w_i, \ h_i$ ) are the ground truth bounding boxes coordinates which has been remapped based on feature map dimension and the prior anchor boxes which has largest IOU with it. The mapping follows the equations we defined just now in previous subsection. (

$\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$ ) are the predicted bounding box coordinates and during the actual prediction stage, we will have to map it back to the original image space coordinates.

$$loss_{confidence} = \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

And finally, the classification error:

$$loss_{class} = \lambda_{class} \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c)^2)$$

The final loss function is the sum of ( $loss_{coord}$ , $loss_{confidence}$, $loss_{classes}$ ). There are 4 hyperparameters ( $\lambda_{coord}$ , $\lambda_{obj}$ , $\lambda_{noobj}$ , $\lambda_{class}$ ) we can tune to control the contribution of each type of loss. In this project, we set those hyperparameters as:

( $\lambda_{coord}$ , $\lambda_{obj}$ , $\lambda_{noobj}$ , $\lambda_{class}$ ) = (1, 5, 1, 1)

Which is the same based with the original YOLOv2 implementation.

## Feature Extraction Networks

The original YOLOv2 method used a custom convolutional networks named Darknet19 to extract features. However, DarkNet19 is way too complicated to run on an embedded platform such as TX2. Below is a quick summary of complexity and accuracy of some popular feature extraction networks.

| Model | Size | Top-1 Accuracy | Top-5 Accuracy | Number of Parameters |
|---|---|---|---|---|
| VGG16[2] | 528MB | 0.715 | 0.901 | 138M |
| ResNet50 | 99MB | 0.759 | 0.929 | 25M |
| InceptionV3 | 92MB | 0.788 | 0.944 | 24M |
| MobileNet | 17MB | 0.665 | 0.871 | 4.2M |
| DarkNet19[3] | 80MB | 0.729 | 0.912 | 19M |
| SqueezeNet[15] | 4.8MB | 0.575 | 0.803 | ~1.2M |

As we can see from above tablet. MobileNet has the very small footprint but still has relatively good accuracy. SqueezeNet has even smaller footprint compared to MobileNet and would be also a good candidate to be used as a feature extraction network. However, in this project we

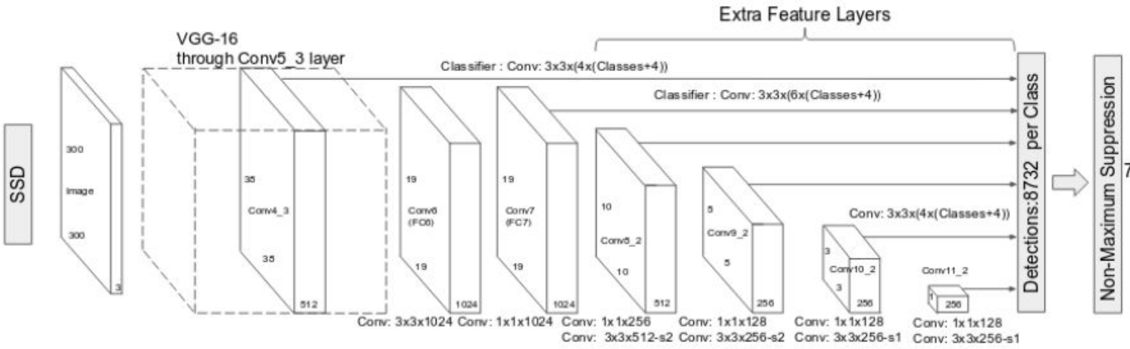
[2] Results for VGG16, ResNet50, InceptionV3 and MobileNet are from https://keras.io/applications/

[3] Results for DarkNet19 is got from this link: https://pjreddie.com/darknet/imagenet/

will use MobileNet as accuracy is also quite important to us and MobileNet has a good tradeoff between complexity and accuracy.
All the implementation code is hosted on github: https://github.com/darknight1900/MobileDet

## Refinement

In the very beginning, we designed our object detector by purely replacing the feature extraction networks with MobileNet and remove the last max-pooling blocks so that the final feature map size is 38x38 instead of 19x19. With such change, the detector can run real-time, however, both the recall and precision is very low.
After reading the SSD method and we found the one of the fundamental difference between YOLOv2 and SSD is how many scales of feature map used during the detection. Below is a figure from the original SSD paper, and we can see that multiple scales of features are used for the final detection. Whereas in the YOLOv2 method, only two scales of features are used for the final detection. For a 608x608x3 input, the feature map scales used for detection are 19x19 (the final output) and 38x38 (the feature map right before the 5th max-pooling).

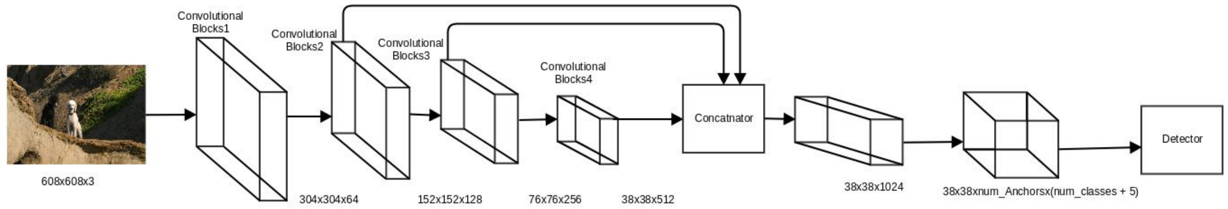


[4]

The benefit of using multiple scales of feature maps is to allow the detector to detect objects in multiple scales better. Especially, in our case, we are trying to detect from aerial images and the objects in those kinds of image are typically small. Using the large scale of feature map will increase our detection performance for small objects. Therefore, we borrow this idea from SSD method and added another scale of feature map for detection. For a 608x608x3 input networks with 4 max-pooling layer, we are using 38x38, 76x76 and 152x152 scales for the detection tasks. We will compare the results of using 2 scales of features and 3 scales of features in the next sections.

The final model we are using is shown as below graph:

[4] The graph is taken from https://arxiv.org/pdf/1512.02325



All the images feeding to the networks will first be scale to 608x608x3 and input images will go through a MobileNet with 4 convolutional blocks and in the end of each convolutional blocks there is a max-pooling layer, therefore the image or feature dimension will shrink progressively.
We concatenate 3 scales of feature maps with dimension (152x152, 76x76 and 38x38) to allow the model to detect smaller objects better. The 3 scales of features will be concatenated together and go through a few more convolutional layers before used for detection.

# IV. Results

## Model Evaluation and Validation

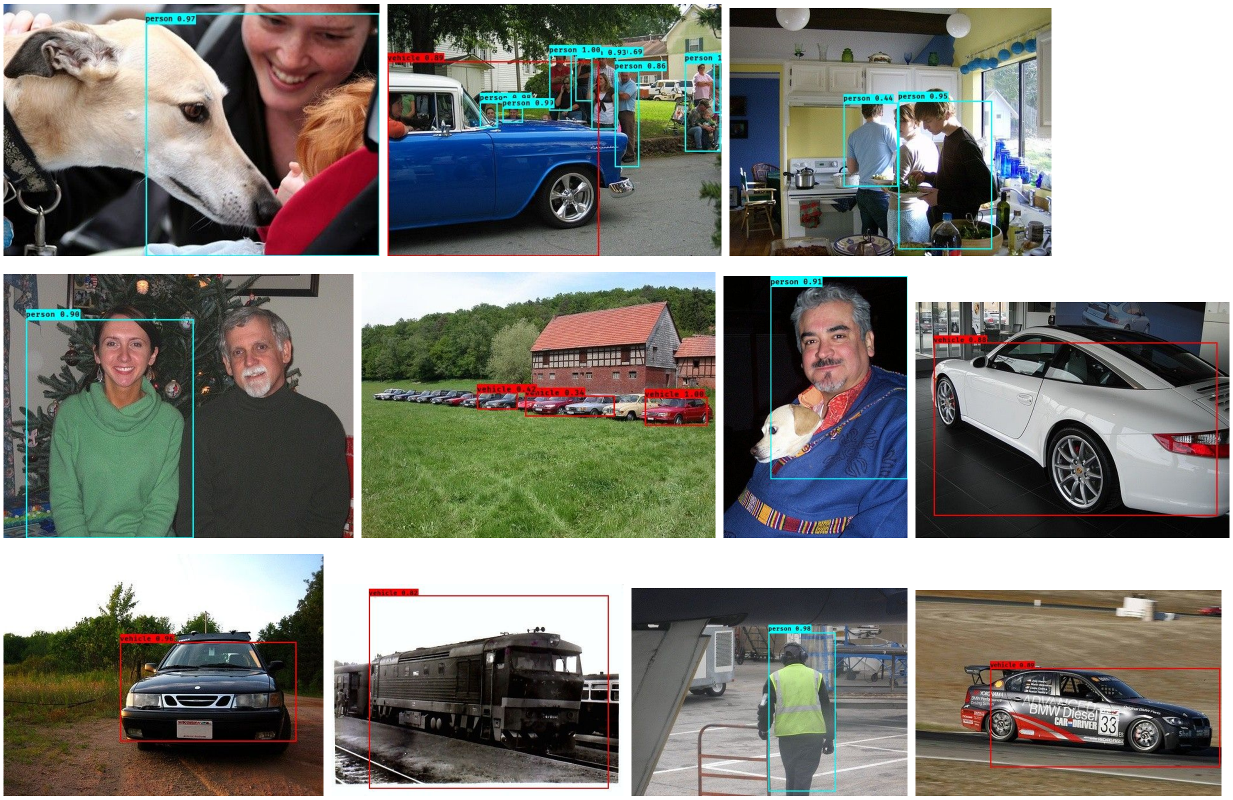
The final model we used is called YOLOv2-MobileNet-Shallow-3Scales. We use a shallow MobileNet as our feature extraction networks, where the last convolutional network blocks and its max-pooling layer have been removed. This is due to the observation that objects on UAV images are typically very small and we do not want shrink the objects too much otherwise they might not even on the final feature map used for detection. This change will also significantly reduce the number of parameters inside the feature detection networks. To further improve the detection for small objects, we also use an extra scale of feature map for detection, therefore total 3 scales of feature maps are used.

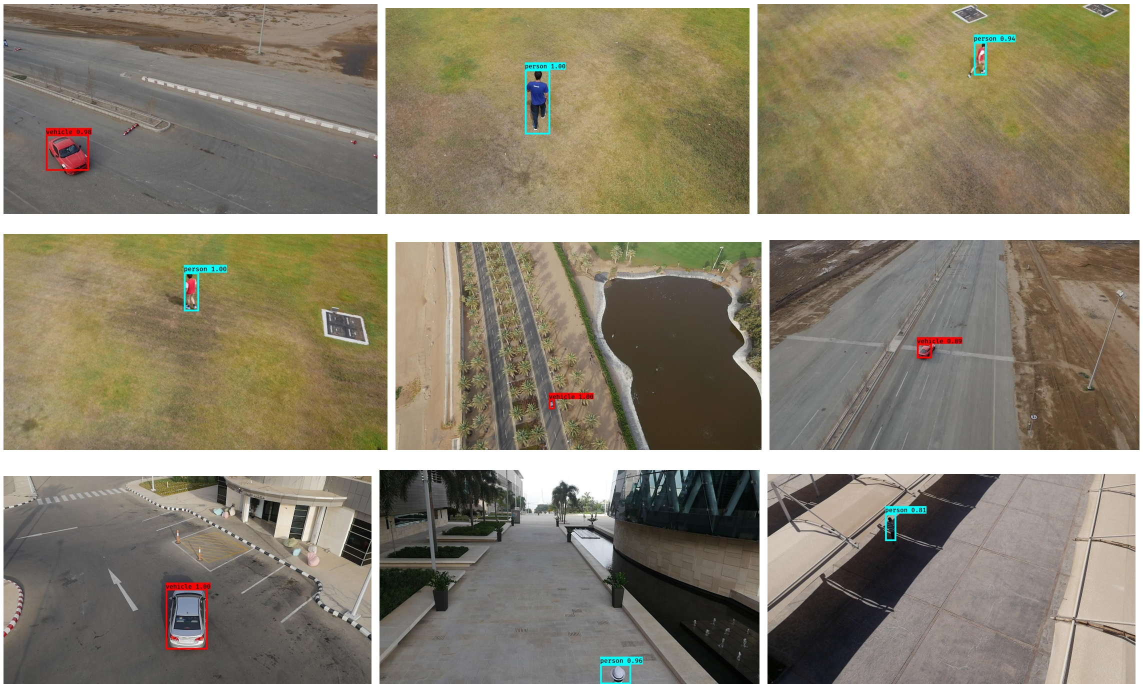
To verify the results, we test our all of the trained models with Test dataset which the model never saw during the training process. For both VOC and UAV123 dataset, the recall and precision with UAV123 test dataset are quite close to with training/validation dataset, therefore the model generalizes well with to the unseen data.

Below is a table summarize the results of trained model with training/validation dataset (seen during training) against test dataset for both VOC and UAV123 dataset. As we can tell from this table, that the detection recall/precision is slightly better with training/validation dataset compared to test dataset, which is as expected. However, the difference is not that significant.

| **YOLOv2-MobileNet_Shallow_3Scales** | Precision (Person/Vehicle) | Recall (Person vs Vehicle) |
|---|---|---|
| VOC Training/Validation Dataset | 0.64 / 0.56 | 0.36/0.27 |
| UAV123 Training/Validation Dataset | 0.93/0.96 | 0.92/0.96 |
| VOC Test Dataset | 0.60 / 0.52 | 0.35 / 0.30 |
| UAV123 Test Dataset | 0.93/0.95 | 0.92/0.94 |

Below is some sample output by using the trained model to detect objects on VOC and UAV123 test dataset. As we can see the model can successfully recognizes the objects we have trained the model to detect, therefore we should be able to trust the results for the trained model. However, as expected we still observe false and miss detection with the model. As we have mentioned earlier detection accuracy is not our solo metric for this project, we are also interested in hardware consumption about our mode and we are going to show the results of running our model on a TX2 platform in the next subsection.





## Justification

In this subsection, we will compare the various performance of our final YOLOv2-MobileNet_Shallow_3Scales vs the original YOLOv2 which serves as our benchmark model.

Below two tablets are the detection results we obtained from VOC test dataset. We trained various version of our model with this dataset for 150 epochs.

In the first table, we compared the detection recall and precision of YOLOv2 and our MobileNet based model with VOC dataset. The input image width and height of the model is always 608x608x3. In the second table, we ran the model on TX2 board and evaluate the speed and GPU memory usage of designed models.

YOLOv2-MobileNet is the model, where the feature extraction network of YOLOv2 has been replaced by MobileNet from DarkNet19. With this change only, we did not lose too much recall and precision at all. However, the number of parameters is reduced from 50.6M to 6.8M, and the speed performance on TX2 is improved from only 1.8fps to 6fps.

YOLOv2-MobileNet-Shallow is the model based on YOLOv2-MobileNet where we removed the last convolution blocks and its max-pooling layers, therefore, there are 4 max-pooling layers inside the network and the final feature map size is increased from 19x19 to 38x38 (38 = 608 / $2^4$ ), with this change, the detection precision is slightly decreased and recall is also reduced from 0.46/0.47 to 0.35/0.33, however the number of parameters is further reduced to 2.6M.

| Model | Detection Feature Scales | Precision VOC (Person/Vehicle) | Recall VOC (Person/ Vehicle) |
|---|---|---|---|
| YOLOv2 (BenchMark) | 38x38, 19x19 | 0.60 / 0.56 | 0.49 / 0.48 |
| YOLOv2-MobileNet | 38x38, 19x19 | 0.58 / 0.59 | 0.46 / 0.47 |
| YOLOv2-MobileNet-Shallow | 76x76, 38x38 | 0.58 / 0.54 | 0.35 / 0.33 |
| YOLOv2-MobileNet-Shallow-3Scales | 152x152, 76x76, 38x38 | 0.60 / 0.55 | 0.35 / 0.33 |

| Model | Detection Feature Scales | Number of Parameters | TX2 FPS | TX2 GPU Memory Usage |
|---|---|---|---|---|
| YOLOv2 (BenchMark) | 38x38, 19x19 | 50.6M | 1.8fps | 264MB |
| YOLOv2-MobileNet | 38x38, 19x19 | 6.8M | 6fps | 112MB |
| YOLOv2-MobileNet-Shallow | 76x76, 38x38 | 2.6M | 6.7fps | 110MB |
| YOLOv2-MobileNet-Shallow-3Scales | 152x152, 76x76, 38x38 | 2.7M | 6.7fps | 110MB |

YOLOv2-MobileNet-Shallow-3Scales is our final model. It is based on YOLOv2-MobileNet-Shallow but adding one more scale of feature map to be used for detection. With this change, the number of parameters is only slightly increased from 2.6M to 2.7M but we will allow the more scales of feature map used for detection so that detection performance for small objects could be improved. With this change, both the detection precision and recall do not change much for VOC dataset. This is most likely due to objects in VOC dataset are all relatively large compared to the image size.

In the below table, we evaluate the detection accuracy of the benchmark YOLOv2 model and the designed YOLOv2-MobileNet-Shallow-3Scales with UAV123 datasets.
After training both of the models with VOC dataset for 150 epochs, we use the transfer learning method by training additional 30 epochs with UAV123 dataset. As we can see from below table, The detection precision and recall are almost the same. However, from the last subsection, the number of parameters is reduced from 50.6M to 2.7M and the speed is improved fro 1.8fps to 6.7fps and GPU memory usage is also reduced by half.

| Model | Detection Feature Scales | Precision UAV123 (Person/Vehicle) | Recall UAV123 (Person vs Vehicle) |
|---|---|---|---|
| YOLOv2 (BenchMark) | 38x38,<br>19x19 | 0.92/0.90 | 0.90/0.94 |
| YOLOv2-MobileNet-Shallow-3Scales | 152x152,<br>76x76,<br>38x38 | 0.93/0.95 | 0.92/0.94 |

# V. Conclusion

## Free-Form Visualization

We used the trained YOLOv2-MobileNet-Shallow-3Scales to detect objects from 4 video sequences which are never seen by the model during the training stage.

https://www.youtube.com/watch?v=KCiKe3zt0IQ&t=23s

https://www.youtube.com/watch?v=YrvsdHExoFo

https://www.youtube.com/watch?v=FVeL0ZZYNDo

https://www.youtube.com/watch?v=jmQabJ598D4&t=1s

As we can see from those videos, the trained model sometimes still miss detecting Person or Vehicle objects especially if the number of objects on the image is more than one. I think this might be caused by the training dataset UAV123. As this is a dataset created for visual object tracking purpose, and only one object is labeled in each object. During pre-processing stage, we already manually removed images which contain multiple Person or Vehicle and tried our best to create good training samples. However, use this kind of training dataset might train the model to only detect a single object from images. This issue could be resolved with a better labeled dataset if we are going to use our model in the real world.

On some of the video such as ‘bike1’, the same Person is correctly detected on one frame and might incorrectly detect as Vehicle in the next frame. And on the future frame, it is correctly detected to be Person again. This issue can be potentially resolved by introducing an object tracker such as optical flow into the pipeline to reject false detection or miss detection.

## Reflection

We used the trained YOLOv2-MobileNet-Shallow-3Scales to detect objects In this project, we are trying to design a real-time object detector for ‘Person’ and ‘Vehicle’ objects of a UAV system. We customized the YOLOv2 method by first replacing the feature detection networks DarkNet19 with a lighter network structure MobileNet. This modification will reduce the number of parameters by ⅕ which will allow us to run the whole detector real time on a TX2 system.

Moreover, in the original YOLOv2 method, there are 5 convolutional blocks and each contains a max-pooling layer. The image or feature map dimension will shrink progressively after each max-pooling layer.  Therefore, for an image with width 608 and height 608, the final feature map would be 19 x 19 x number of channels, which is only 1/32 of the original image size. The objects from the UAV imagery are typically very small compared to the image size. If use max-pooling like the original YOLOv2, the object size on the final feature map might only show as one point. To improve the detection accuracy, we removed the last convolutional network blocks including its max-pooling layers, with this change we both reduce the network complexity and also make it easier for us to train the network to learn how to detect small objects.

It is also interesting to notice that the network structures of SSD and YOLOv2 method are quite similar. For SSD method, multiple scales of feature maps are used for the object detection. As for YOLOv2, only the last two scale of features are used. To improve the detection accuracy, we borrow some idea from SSD method and bring in one extra scale of feature map for detection. For a 608x608 image with 4 max-pooling, we are using feature map with size 38x38, 76x76 and 152x152 for detection. This change also improved a detection accuracy a bit.

One of the difficult parts of this project is to find a suitable dataset to train our network. The most popular dataset used for object detection and classification research are taken from ground cameras and the object size is typically very large compared to the image size. Whereas, the objects from the camera mounted on UAV can be very small. In this project, we are using UAV123 dataset, which is a visual object tracking dataset. The dataset has 123 video sequences and not all the ‘Person’ and ‘Vehicle’ objects are labeled. To use this dataset for object detection purpose, we have to manually remove all the images which not all of those two objects are labeled. Also, length of each video sequence varies a lot, to avoid overfitting, we limit the number of images from each sequence to be used for training. After that processing, we got around 2500 relatively good training images which are still small. We deal with this small training sample issue, we first train our model with VOC dataset which has around 5000 images contain ‘Person’ and ‘Vehicle’ objects. After training the detector with VOC for 150 epochs. We then use transfer learning to train the detector for another 50 epochs with the UAV123 dataset. Also, to make sure the detector is also generic as possible, we designed a couple of ways of data augmentation during the training stage to avoid overfitting issue.

The precision and recall of final trained detector are not as good as the original YOLOv2 method. However, we can run our customized model at around 6.7 frame per second which is almost real time on a TX2 whereas the original  YOLOv2 can only run with 1.8fps, which is definitely not fast enough.  As our ultimate goal is to detect and track those objects from UAV video, we can use the optical flow method to reduce the false and missing detection issue.

# Improvement

With the customized YOLOv2-MobileNet-Shallow-3Scales method in this project, we can detect ‘Person’ and ‘Vehicle’ at almost 7 frames per second. Many traditional computer vision object detection algorithm will only use luma channel of images. We could also potentially slightly tune the detector to only use luma channel and this could reduce the number of parameters by  half for 16bit YUV422 images or by ⅓ for 12bit YUV 420 images.

Although we still have false and missing detection issues with the UAV123 dataset, we could use some other computer vision algorithm to mitigate the issue. As essentially, we are trying to detect objects from video and as long as the object has been detected once, we could use the method such as optical flow [16] to reduce the false and missing detections. The heaviest part of our object detector is caused by the feature extraction networks. As we have mentioned it in previous sections. In this project, we replace the DarkNet19 in the original YOLOv2 method by using MobileNet and the number of parameters is reduced almost by 5 times. There is a new network structure named SquezzeNet which has even smaller parameters and we could consider using that to further improve the detection performance.

Also our current model is biased toward only detecting a single object from images. This issue could be potentially resolved by using better training datasets.

# References

[1] M. Bhaskaranand and J. D. Gibson, “Low-complexity video encoding for uav reconnaissance and surveillance,” in Military Communications Conference (MILCOM). IEEE, 2011, pp. 1633–1638.

[2] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair,I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, “Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue,” IEEE robotics & automation magazine, vol. 19, no. 3, pp.46–56, 2012.

[3] Andriluka, M.; Schnitzspan, P.; Meyer, J.; Kohlbrecher, S.; Petersen, K.; Von Stryk, O.; Roth, S.; Schiele, B. Vision based victim detection from unmanned aerial vehicles. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 18–22 October 2010;pp. 1740–1747.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640, 2015.

[5] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242, 2016

[6] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C., Winn, J., and Zisserman, A. The Pascal visual object classes challenge: A retrospective. IJCV, 111(1):98–136, 2015

[7] T. Huster and N. C. Gale. Deep learning for pedestrian detection in aerial imagery. In MSS Passive Sensors, 2016.

[8] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B.Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. CoRR, abs/1405.0312, 2014

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E.Reed, C. Fu, and A. C. Berg. SSD: single shot multi-box detector. CoRR, abs/1512.02325, 2015.

[10] J Carlet, B Abayowa, Fast Vehicle Detection in Aerial Imagery. arXiv preprint arXiv:1709.08666, 2017 - arxiv.org

[11] M. Mueller, N. Smith, and B. Ghanem, “A benchmark and simulator for uav tracking,” in Proc. of the European Conference on Computer Vision (ECCV), 2016.

[12]S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In NIPS, 2015.

[13] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan.Object detection with discriminatively trained part based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):1627–1645, 2010. 1, 4

[14] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T., Marco Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[15] Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. arXiv preprint arXiv:1602.07360 (2016) 4

[16]J. Y. Bouguet . Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. Intel Corporation, 5.