OBJECT DETECTION

GR5291 Advanced Data Analysis

2019/12/09

Prepared by:

Jinrong Cao(jc4515), Huaqiu Chen(hc3042)
Xudong Guo(xg2305), Hang Hu(hh2718)
Yuki Kitayama(yk2797), Sixuan Li(sl4410)
Xinyue Li(xl2823), Yicheng Li(yl4104)
Greg Lu(bl2738), Chen Wang(cw3105)
Jingwen Wang(jw3667)

Table of Contents

1. Introduction

With the advancements in machine learning technologies, many novel industries have emerged. Among them, autonomous driving is without a doubt one of the most intriguing and profitable. A fundamental technology supporting the realization of autonomous driving is Object Detection (extendedly Object Recognition) where 3-D objects surrounding the vehicles are identified and tracked in even the most complex environmental settings to ensure the safety of the driver and passengers.

With inspirations from the industrialized 3-D object detection technology, our team aims to explore and develop our own rudimentary version of it. The ultimate goal of Object Detection is to locate and assign objects to their corresponding class labels. There are two types of common approaches in Object Detection: Machine-learning-based approaches and Deep-learning-based approaches. Different from machine-learning approaches where features need to be pre-determined, deep-learning-based approaches based on Convolutional-Neural-Network (CNN) are more flexible in tackling end-to-end problems. Deep learning techniques have witnessed a rapid development with tremendous success in its applications within the past decades. Object Detection, being one of the building blocks of visual recognition in computer vision, has been widely studied and enhanced based on deep learning image classification approach. Existing literatures and application projects provide us with rich research resources. A walk-through on Object Detection projects has been described in 'ImageNet Large Scale Visual Recognition Challenge' (Olga, et al., 2015), where three key steps are identified in general: (1) Image Classification; (2) Object Localization; (3) Object Recognition. Image classification takes images as input and predicts the type or classes of an object. Object Localization is the process where the object is located with a

proper bounding box. Object Recognition contains instance segmentation along with the localization.

2. Objective

Inspired by its great application in autonomous driving, the objective of this project is to turn fascinating ideas into reality by building an Object Detection algorithm based on deep learning approach. By reviewing related work with technical details and carefully considering the feasibility of implementation, we decide to apply the Object Detection technique on 2-D Images. By reviewing related work literature (Wu, 2019) and running a preliminary test, we realize that it is very difficult to perform the instance segmentation right after object localization with high accuracy. Therefore, our primary objective of this project is to perform the Object Detection with a precise Bounding Box. Bounding Box with object detected is formally defined as Object Localization. The objective of this project contains four major parts which will be specified in below.

2.1 Image Classification

The goal of image classification is to build a CNN model and to adjust the architecture for better performance. During the model building process, the modeling team will justify the feasibility and efficiency of the chosen training data (CIFAR-10). Moreover, we will keep exploring other available resources in enhancing the model performance. The pre-trained image classification model result will be passed onto the object localization algorithm. (Section 4.1)

2.2 Detection Algorithm Development

The objective of Detection Algorithm Development is to write the object localization algorithm. By the preliminary studies, we decided to develop the algorithm of sliding windows and image

pyramid. At this step, our goal is to draw a proper bounding box around the interested object in the image. (Section 4.2)

2.3 Methodology Review and Further Study

Object detection is one of the major challenges in Computer Vision, thus, the topic is very difficult to be covered thoroughly within this project scope. Suggested by Professor Alemayehu, we are going to research on industrial leading models and methodologies and compare them with our model so that we would find out our model limitation as well as potentials of improvement.

3. Data Description

3.1 Public Images Training and Validation Data

The CIFAR-10 Dataset (Canadian Institute for Advanced Research) (Krizhevsky, 2009) is utilized for building our image classification models. It is a collection of images which contains 60,000 32-by-32 color images in 10 different classes. The 10 different classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The CIFAR-10 Dataset is a well-known public image dataset used for machine learning and computer vision algorithms. This data set has several benefits in training the image classification model: (1) the classes are completely mutually exclusive to each other, which lowers the noise in training and validation image inputs; (2) for each class, there are 6,000 images can be used to trained the CNN model, which is a relatively big sample; (3) all images are sized 32-by-32 so that we can use a standardized sliding window size in the detection algorithm.

3.2 Testing Images Collection

The CIFAR-10 training images contains one object per image, and the object occupy most of the area of each training image, which is suitable as training dataset. However, this CIFAR-10 images are not suitable for object detection purpose. The testing images should be something like our

target object exist somewhere inside the test image and we are trying to find it. This task would be set depending on the different level of difficulty in background and object conditions. Hence our testing images are searched from Google, and individually picked up for object detection purpose. It helps us develop detection algorithm which can tackle with the real-world scenario.

4. Methodology

In this section, we will introduce the model planning and methodologies in detail. The project includes two major steps: (1) Image Classification (Section 4.1); (2) Object Localization (Section 4.2).

4.1 Image Classification

This section contains our model plan in performing image classification. The baseline model is trained by CNN. We consider transfer learning technique and available models to further enhance the classification accuracy.

4.1.1 Image Classification Model

With CIFAR-10 dataset, we built 10 class neural network image classification models for object detection. In total, we built 4 models, each of which was applied for object detection. The following table summarizes the result of our models, followed by detailed discussion of each model.

(For more details, please refer to the appendix.)

Table 1. Model Summary

Model name	ResNet model	MobileNet model	Model 2	Model 1
Training accuracy	99%	66%	83%	71%
Validation accuracy	89%	51%	76%	66%
# of training data	45,000	50,000	47,500	40,000
# of validation data	5,000	10,000	2,500	10,000
Optimizer	Adam lr = 0.001	Adam lr = 0.001	Adam lr = 0.0001	RMSProp
Epoch	10	4	20	5
Batch size	256	32	64	500
Time to train	About 10 min	About 9.5 min	About 7 min	About 8 min

Model 1:

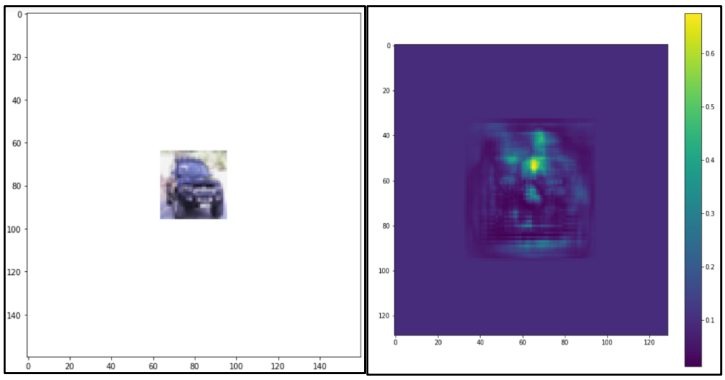
Figure 1: Model 1 -Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 7200)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3686912
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,702,186
Trainable params: 3,702,186
Non-trainable params: 0
_________________________________________________________________
```

Discussion

Model 1 is our baseline model. Our ultimate goal is not only to build a classification model, but to successfully detect objects by applying classification models. Therefore, the objective of our first model is to quickly build a moderately acceptable image classification model and make a pipeline to object detection tasks. Since our input is image, we used convolutional layers and max pooling layers to capture spatial information. We also intended to make shallow layers for this preliminary model, because deeper layers would potentially slower the process of prediction, which is object detection. To test the performance of object detection, we used the following image and obtained the result on the right.

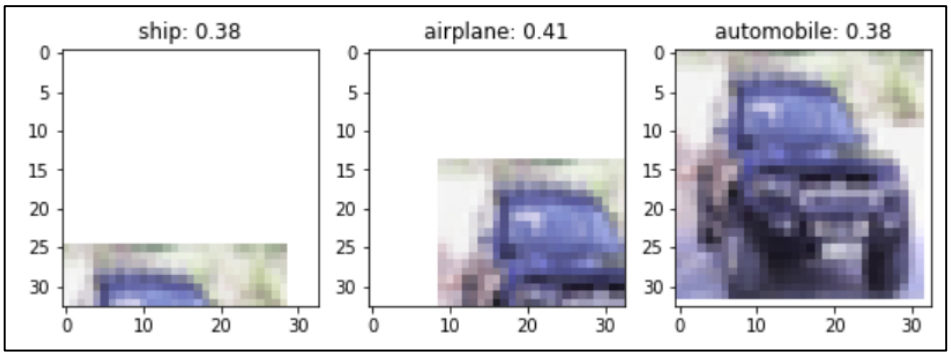
Figure 2: Model 1 – Detection Result



The right image is a matrix which contains maximum probability for each pixel point, with dark color representing low probability. We notice that the white space which has no objects produces low probability of classification, whereas in the middle, where our image is located at, we observe many yellow/greenish spots which represent higher probabilities. Hence, we could conclude that Model 1, along with sliding windows, is able to capture an object by producing higher probabilities for those pixel points.

However, as shown below, Model 1 produces many false positives, a potential issue that could lead to problematic consequences. For example, when we apply Model 1 to real pictures having complex backgrounds, the model would predict there is an object in the background even though that is not the object we hope to detect. Another limitation of Model 1 is that, even though we are able to detect objects, we cannot label them, e.g. an automobile in this image.

Figure 3: Potential Problems in Model 1



Model 2:

Figure 4: Model 2 - Architecture

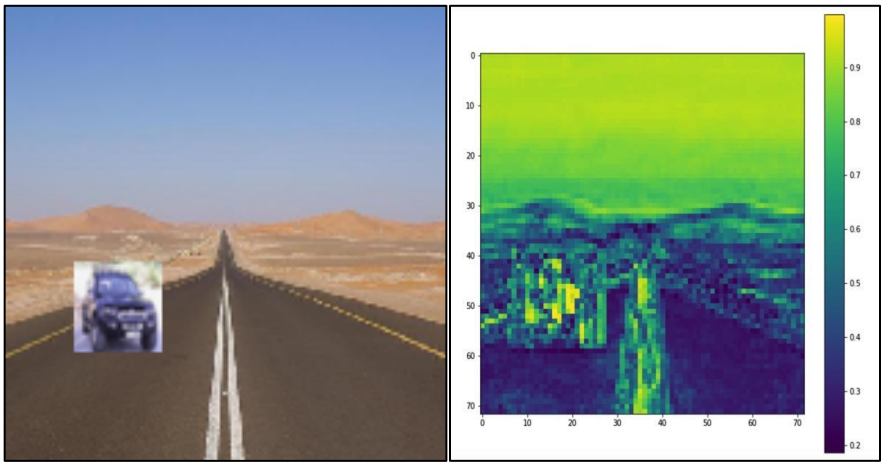
Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 32, 32, 32)	896
activation_38 (Activation)	(None, 32, 32, 32)	0
conv2d_23 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_v1_14 (B	(None, 32, 32, 32)	128
activation_39 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_11 (MaxPooling	(None, 16, 16, 32)	0
conv2d_24 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_v1_15 (B	(None, 16, 16, 64)	256
activation_40 (Activation)	(None, 16, 16, 64)	0
conv2d_25 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_v1_16 (B	(None, 16, 16, 64)	256
activation_41 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_12 (MaxPooling	(None, 8, 8, 64)	0
flatten_6 (Flatten)	(None, 4096)	0
dense_17 (Dense)	(None, 1024)	4195328
activation_42 (Activation)	(None, 1024)	0
dropout_11 (Dropout)	(None, 1024)	0

```
_________________________________________________________________
dense_18 (Dense)             (None, 512)               524800
_________________________________________________________________
activation_43 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                5130
_________________________________________________________________
activation_44 (Activation)   (None, 10)                0
=================================================================
Total params: 4,791,466
Trainable params: 4,791,146
Non-trainable params: 320
_________________________________________________________________
```

Discussion

The goal of Model 2 is to make more accurate predictions on image classification. To achieve this goal, we added more convolutional layers and batch normalization layers to maintain performance and training speed since Model 2 has much deeper layers. We also increased the number of nodes in densely connected layers. As a result, we successfully increased the model accuracy. To examine the performance of object detection, we manually created the following image, which contains the original automobile object and a simple background.

Figure 5: Model 2 – Detection Result



As we can observe from the probability matrix on the right, Model 2 produces higher probabilities in the automobile area of the image and lower probabilities for road and mountain which are not

the object we intend to detect. However, we notice that Model 2 also produces high probabilities in the sky area which is essentially empty. This mistake prevents us from successfully making bounding boxes which will be discussed in the later sections. We further confirmed that the sky is predicted to be airplanes, which leads us to suppose that Model 2 is not trained by the shape of airplanes, but rather the color of sky behind each airplane. We assume that to overcome this difficulty, we might need to conduct data augmentation, such as changing the color of airplanes in the training data. However, we still believe we have the potential to improve prediction accuracy, so we move on to build a transfer learning model and apply it for object detection.

4.1.2 Transfer Learning

In order to improve the image classification accuracy, the model team further enhances the model performance by adopting transfer learning techniques. Intuitively speaking, "Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned." (Olivas, 2010) Many high-performance image classification models with superior architecture and training of CNN have been developed on the ImageNet Large Scale Visual Recognition Challenge. By adopting models with transfer learning, the feature extraction and computational efficiency should be improved comparing to CNN. The top-performing pre-trained models are accessible from Keras. Before we select the transfer learning methods, the research team systematically reviews current available models in terms of architecture, accuracy, computational efficiency and feasibility of implementation.

Model Comparison and Selection

The models that we initially consider include VGGNet (VGG16, VGG19), InceptionV3, ResNet, MobileNet, DenseNet and NASNet.

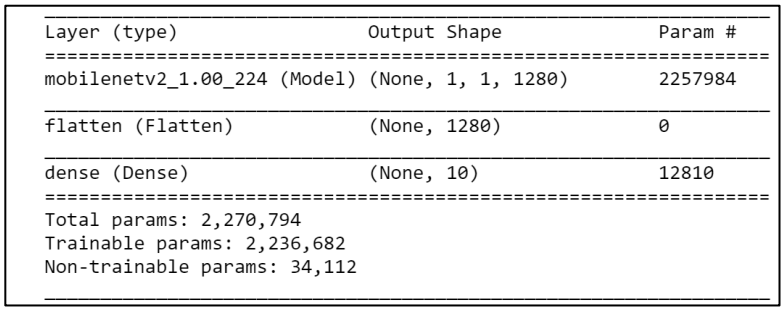
- **VGG networks**: [3]： is characterized by its simplicity in the ideology, which uses only 3-by-3 convolutional layers stacked on top of each other to increase the depth with max-pooling and activation layers before some fully connected classification layers. Two major disadvantages include: (1) slow running time; (2) heavy architecture weights themselves.
- **InceptionV3** (Koustubh, 2017)： specialized in widening the nets. The inception module family computes multiple different transformations over the same input parallelly and then concatenating the final results into a single output. Although InceptionV3 is able to reach to relatively higher accuracy, sometimes the sparse structure may lead to overfitting.
- **ResNet** (He, 2016)**:** The core idea of ResNet is introducing a so-called "identity shortcut connection". Skipping effectively simplifies the network with fewer layers in the initial training stages. This facilitates learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through.
- **MobileNet** (Howard, et al., 2019): based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks.
- **DenseNet**: each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Recent study (Tsang, 2019) shows that ResNet out perms DenseNet on CIFAR-10 dataset.
- **NASNet** (Tsang, 2019): the model's blocks/cells are searched by reinforcement learning search method. This method tends to resemble RNN. Studies show that the performance on CIFAR-10 dataset is competitive to ResNet method, but with greater complexity.

With carefully considering the computational efficiency and performance on CIFAR-10 dataset, we chose to use ResNet and MobileNet to enhance the classification model accuracy.

Compared to convolutional network, MoblieNets, a smaller and efficient neural network architecture, is proposed. It can reduce latency and remain accuracy on a computationally limited platform such as robotics, self-driving cars and augmented reality. (Refer to Appendix for details of each model) Thus, we have chosen MobileNet to compare with our existing models to see if the MobileNet structure could result in any improvement of the efficiency while keeping an approximate accuracy.

MobileNet model:

Figure 6: MobileNet Model Architecture



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 1, 1, 1280)        2257984
_________________________________________________________________
flatten (Flatten)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 2,236,682
Non-trainable params: 34,112
_________________________________________________________________
```

Discussion

Here we only discuss the architecture the model and result. The reason why we chose this existing model is discussed in the other sections.

As shown from the results, we see a decrease in the accuracy of the model from 76% to 51%. This might be due to a mismatch between the MobileNet model structure and our dataset. In addition, the MobileNet does not include any regularization terms so it is not very good with generalizing. However, it is surprising to see the training time had also increased to around 9.5 mins. Thus, it is clear that our model structure outperforms the MobileNet structure in both accuracy and efficiency.

ResNet model:

Figure 7: MobileNet Model Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 1, 1, 2048)        23587712
_________________________________________________________________
global_average_pooling2d_3 ( (None, 2048)              0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                5130
=================================================================
Total params: 24,641,930
Trainable params: 24,588,810
Non-trainable params: 53,120
_________________________________________________________________
```

Discussion

Our goal of using transfer learning from ResNet model is to build more accurate image classification model than our Model 2. Hence, on top of the ResNet layers, we added our own layers. We firstly added Global Average Pooling (GAP) layer, because it is believed that GAP layer not only mitigate overfitting but also popularly used for object localization in research. Additionally, we added densely connected layer with a large number of nodes 512, and added dropout layer to mitigate overfitting. The learning rate is tuned to be 0.001 while 0.01 gives unstable result and 0.0001 results in very slow convergence. It excellently resulted in achieving 89% accuracy in validation dataset. Since we used ResNet layer, the model in fact is very deep. The training time is extremely long compared to previous models. Therefore, we used GPU setting on Google Compute Platform for ResNet model development, Hence, the training time was improved to similar level with other models.

4.2 Detection/Location Algorithm

After the model and weights trained by CNN and the transfer learning method has been saved, the next step is to draw a proper bounding box to locate the object. This section focuses on introducing methods that we are planning to use to realize the object localization. The only use of OpenCV is just to draw bounding boxes in an image. To draw the boxes, we have to specify appropriate location information. We designed and developed an algorithm to extract this location information, so we did not use the built-in function of OpenCV. To develop detection algorithm, we used the following three pictures, each of which has different goals that we want to achieve.

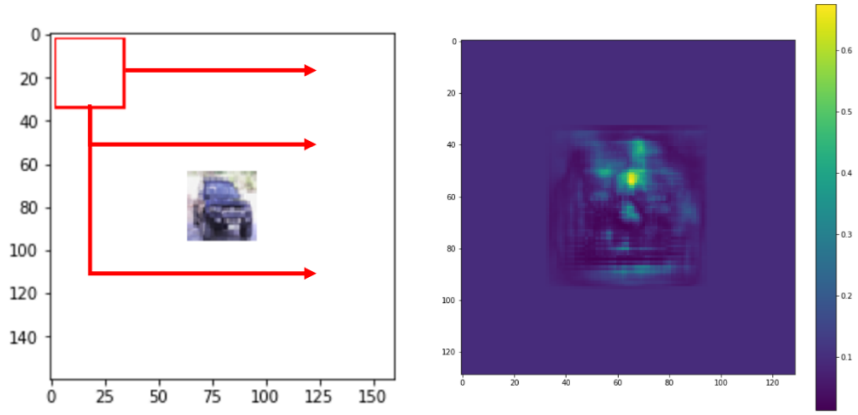
Figure 8: Detection Algorithm Input Images

The first picture has size 160 pixels height and width without any objects beside an object in the middle. Inside of the picture contains one automobile whose size is 32-by-32. We consider this is appropriate development case since the object size is the same as our image classification model input and we only have a single object. The second picture has the same size as the first image. However, it is more difficult than the first picture, since we do not have a particular object but we have a slightly simple background image. We want to test that challenges we would face with this setting. The third picture is also the same size with 160 by 160 pixels, but this time the object size is 64 by 64. Therefore, our image classification model input cannot contain an entire shape of the car. In a naïve way, we just predict the object based on a part of the information. However, to improve the predictive performance, we developed the following image pyramid approach.

4.2.1 Sliding Window Approach

Sliding Window (Rosebrock, 2015) is a common approach in Object Detection Process. Intuitively speaking, a sliding window is a rectangular region of fixed dimensions (height and width) which "slides" across an image from left to right, top to bottom. The window size is usually determined by resolution of the image, which is 32-by-32 in our case. For each window, we take the region defined by it and apply the pre-trained image classifier to determine how likely the window region contains an interested object. One potential issue of using fixed size window is that we may have a window where only a small portion of object is captured, or we may have a partially overlapped object. Image Pyramid technique is going to be used to tackle this challenge and further improve the detection algorithm. Python code of sliding windows is in the appendix.

Figure 9: Sliding Window Approach – Image Demostration

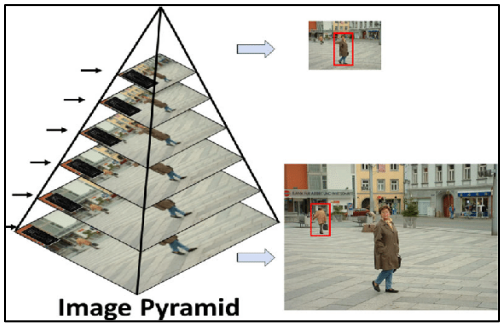


4.2.2 Image Pyramid and Resize

Image pyramid is a major technique used for object detection, which is necessary when a classification model input cannot contain an entire image of an object. The idea of image pyramid is explained with the following example. For example, each image of CIFAR-10 dataset has 32 height and 32 width so that our input for prediction is also limited to this 32 by 32. However, if an object in a test image has size more than 32 by 32, then the model has to make a prediction based

on a part of image information. Hence, the detection algorithm needs to reduce the size of a test image into a smaller size where a test object is fully contained by the input size of the classification model. Since the detection algorithm does not know in advance how much smaller it needs to resize, a variety of resizing should be conducted; the original size, a half of the original size, one quarter, and so on. With these different sizes, we assume that one of the resized images has an object size which can be detected. When these images are accumulated from the bottom with the original size to the top with smaller resized images, it looks like a pyramid. Thus, this methodology is called image pyramid.

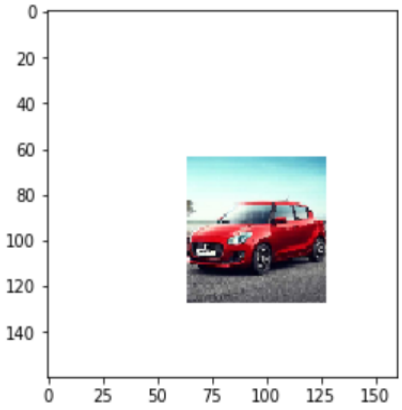
Figure 10: Image Pyramid Demo[1]



We developed this image pyramid python codes and confirmed that it works for object detection and labeling. To be concrete, we describe our approach with the following image.

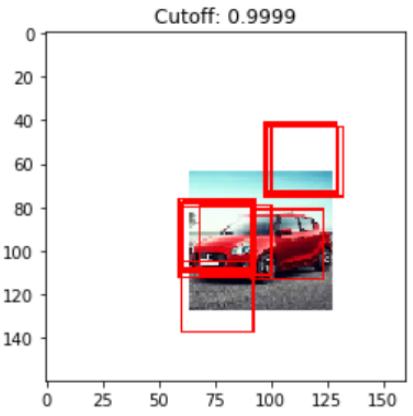
[1] Source: https://www.researchgate.net/figure/mage-pyramid-with-multiple-scales-Processing-all-scales-with-the-same-SVM-template-can_fig1_305510342

Figure 11: Image Pyramid – Input Image



This image has 160 height and 160 width with an automobile of 64 height and 64 width. Since our classification model input is 32 by 32, we need to use image pyramid for this object detection. In the original size, we can make 129 predictions (160 width – 32 input size + 1 original location) in column-wise. Likewise, there are 129 predictions in row-wise. In total, there are 16,641 predictions (129×129). We will have different level of argmax probabilities over 10 classes and different predicted labels so that we set cutoff to extract the confident predictions. With cutoff 0.9999, we can extract 14 predicted labels and locations out of 16,641. 9 out of 14 is predicted to be automobiles and 5 are predicted to be ships (1: automobile index, 8: ship index) as show below:

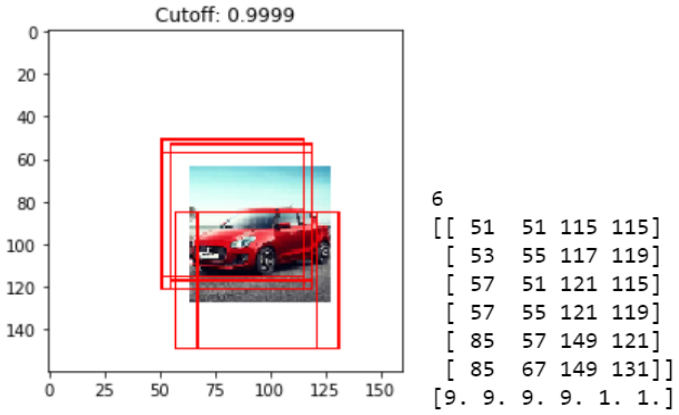
Figure 12: Image Pyramid – Result 1




```
14
[[ 41  97  73 129]
 [ 42  97  74 129]
 [ 43  97  75 129]
 [ 43  98  75 130]
 [ 43 100  75 132]
 [ 76  59 108  91]
 [ 76  61 108  93]
 [ 77  61 109  93]
 [ 78  60 110  92]
 [ 78  61 110  93]
 [ 79  68 111 100]
 [ 80  59 112  91]
 [ 81  91 113 123]
 [105  60 137  92]]
[8. 8. 8. 8. 8. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Counter({1.0: 9, 8.0: 5})
```

Although we have many automobile predictions, they are not predicted by an entire image of the automobile, as one can observe above that bounding boxes only contain a part of the object. The detection algorithm proceeds to resize the image by a half. Now, we have 80 height (=160/2) and 80 width, and the automobile size has 32 height and 32 width. The resized object can fully be contained by the mode input size. With this new size, there are 49 predictions (80 width – 32 input size + 1 original location) in column-wise, 49 predictions in row-wise, and 2,401 predictions in total (49×49). Applying the same 0.9999 cutoff, we extracted 6 confident predictions. 4 out of 6 are predicted to be trucks, and 2 are automobiles (9 is truck index). While drawing 32 by 32 bounding boxes in the first picture, we drew 64 by 64 boxes in the second picture. The original size of the car is 64 by 64, and we resized it into 32 by 32 so that we drew twice as large bounding boxes as the original size. The result is the following.

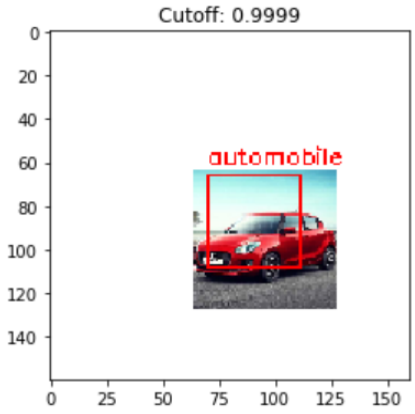
Figure 13: Image Pyramid – Result 2



We resized again into one quarter, which has 40 height (=160 / 4) and 40 width, there are 81 predictions made in total (40 – 32 + 1 with column and row). However, the maximum probability of predictions is 0.9973, so we do not extract predicted labels and location information with the same 0.9999 cutoff. In the end, we take the average of all the location coordinates that we extracted,

and take majority vote from the predicted label, we can make the following object detection. The two concepts are discussed in the further sections.

Figure 12: Image Pyramid – Final Result



4.2.3 Suppression of multiple boxes into single bounding box

We can draw bounding boxes in all locations of an image, however, the probability levels for distinctive locations may not be the same. We cannot arbitrarily pick bounding boxes since it is not an automated object detection algorithm. Hence, we set a cutoff value to extract bounding boxes in which the probabilities are above the cutoff. At the end of the box selection process, usually we end up with multiple boxes, therefore, we implemented the following method to combine them into a single box.

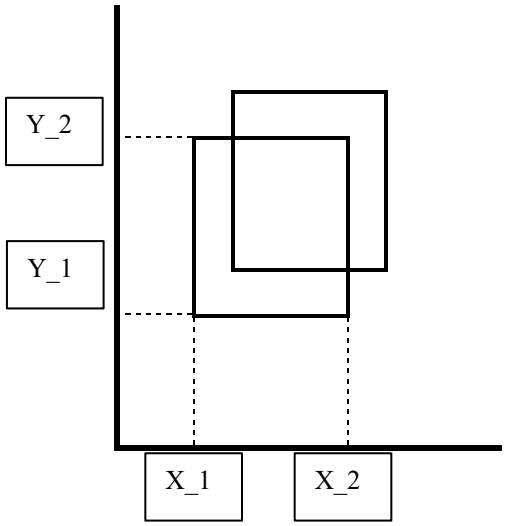
- **Min Max Method**

In order to draw a bounding box, we need to have 4 values; x_1, x_2, y_1, and y_2. They determine the coordinates of the four corners of a box as illustrated in the diagram below. We label (x_1, y_1) as the first coordinate, (x_2, y_1) as the second coordinate, (x_1, y_2) as the third coordinate and (x_2, y_2) as the fourth coordinate. In min max method, we aggregate multiple coordinate by taking minimum from x_1, maximum from x_2, minimum from y_1, and maximum from y_2 in

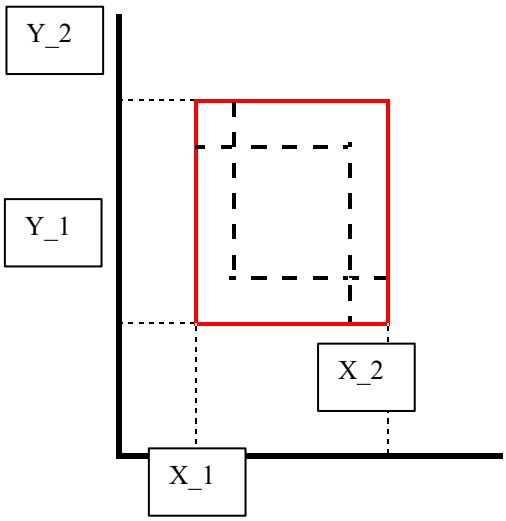
Graph 1. This procedure results in a single bounding box that covers all the areas of the multiple boxes we started with, as illustrated in Graph 2 in red.

The advantage of this method is that we end up with a bigger box so that it is more likely to contain the entirety of an object. A potential drawback of the method is that, in the case where two different objects are located very closely, the individual sets of bounding boxes of object A and B could connect with each other and thus making separate and accurate detections of the two objects extremely difficult. We might just end up with a large box that covers the entirety of the two objects altogether.

Graph 1



Graph 2



- **Taking average**

We are also using the same coordinate labeling system in this method as the previous one. The procedure starts by taking the average of each x_1, x_2, y_1, and y_2. With this approach, we will be able capture the overlapping area of multiple boxes. Hence, the resulting single bounding box tends to be smaller than the total area occupied by the multiple boxes from which it is drawn.

The advantage is that, even though we might have a small portion of bounding boxes from false positive detection, since we end up with a larger number of boxes around an object, we can mitigate the effect of the false positive detection and be able to draw a single box around the object. The disadvantage is that, because we are taking averages of the coordinates, we cannot assure that the resulting box contains the entire image of the object. The appendix has python code of taking average.

4.2.4 Object labeling

To label an object that we detect by sliding windows and image pyramid, we took the majority vote from the predicted classes. The reason why we have to take the majority vote is the following. In object detection, we perform image classifications over all the subareas in an image. In image classification, we can predict a label because we first predict probability of each class and then take argmax to decide which is our predicted label. Hence in object detection, we have multiple predicted classes and extract one prediction from them. For implementation, we simply counted how many predictions we have from each class. For example, we apply sliding windows, set cutoff as 0.99, and extracted 10 predictions. Each prediction is made by argmax. Suppose that we have predicted labels of 4 airplanes, and 6 automobiles. By taking majority vote, the label of the detected object is automobiles. Although we did not implement, we have potential issue. In applying sliding windows, we have multiple predicted classes from each floor of the image pyramid. However, the resolution of the objects in each floor is different. So maybe it is appropriate to set weights depending of which floor of the image pyramid the predictions are extracted. The appendix has majority vote python code.

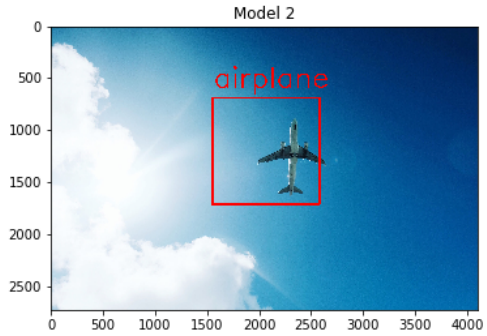
5. Model Performance: Results from Object Detection

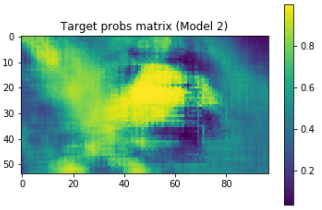
To obtain results from object detection, we tested it to 10 real-world setting pictures, among which 5 are successfully detected.

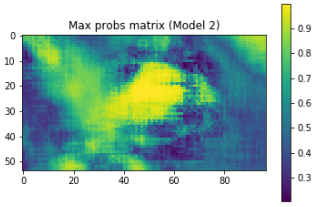
Test pictures	Result
Dog_01	Successful
Bird_01	False Positive
Cat_02	False Positive
Airplane_02	Successful
Automobile_truck_02	Multi-object
Frog_01	Successful
Deer_02	False Positive
Ship_01	Successful
Horse_01	Successful
Dog_03	False Positive

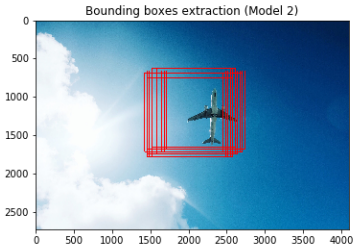
As we want to compare industry leading models with our unique model, we chose to implement ResNet model and Model 2. Since in practice, the image pyramid is computationally expensive, we chose to resize all test images to around 10,000 pixels each. According to our test results, we can detect the objects even with one cycle of the sliding window. Below are 3 detailed test examples (for all results, please refer to the appendix):

- airplane_02.jpg



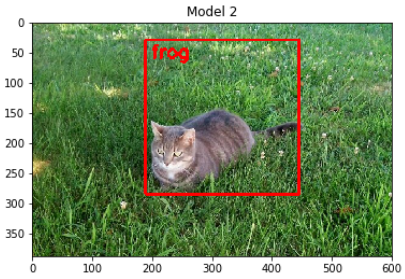


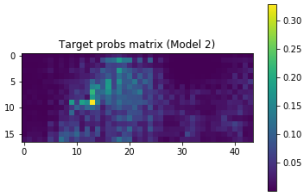


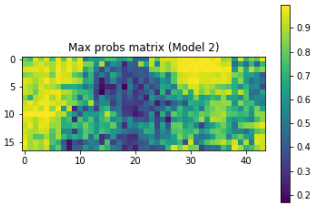


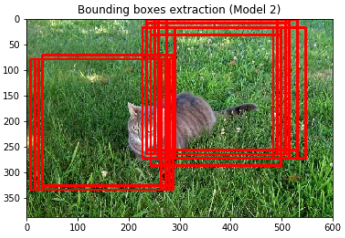
In this case, object detection by Model 2 is successful. The two probability matrixes also suggest that it does not face much false-positive in detection.

- cat_02.jpg

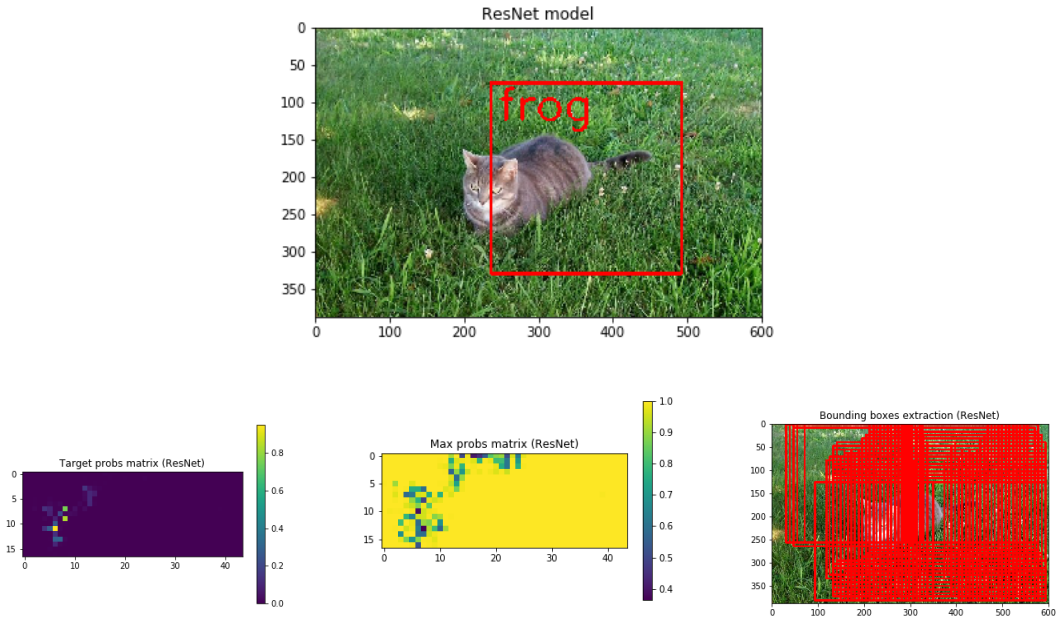








For this picture, although the bounding box contain the object, we failed to label it. Model 2 predicts few cats and the probability is around 30% as shown in the left image. The label with the highest probability is frog, rather than cat. Since we failed to detect the object, we applied ResNet model to test whether our detection would be improved, as shown below.



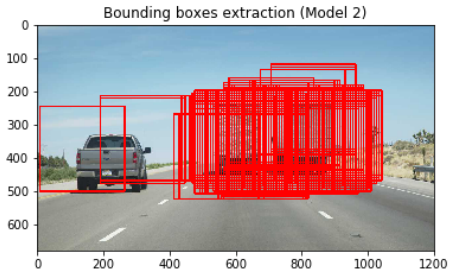
ResNet model did not make more cat predictions. Rather the probability of frog is intensified with probability 1. To conclude, there are too many false negatives so that we failed to detect object. Further data augmentation discussions will be shown in the conclusion part.

- automobile_truck_02.jpg


```
Counter({9.0: 66})
9
truck
```

In previous cases, we took the mean of confident locations to result in a single object detection. In this case however, by decreasing the cutoff level, we can still draw bounding boxes for the car at the left and the truck in the middle. In the above case, we set 0.999999 cutoff while the below bounding boxes are extracted with 0.999 cutoff. However, when we suppress the multiple

bounding boxes, we take average and it leads to single bounding box. Therefore, it is apparent that we need to develop a multi-object detection algorithm, which will be further discussed in the conclusion part.



6. Conclusions/Further Discussion

In this project, we successfully implemented the deep learning application for object detection. We first developed the multi-class image classification model with deep learning methodology. We achieved a high accuracy with our unique layers of neural network and compared our performance with the existing neural network models by transfer learning. Then we applied this model to a bigger test image which contains some object. Detection of this object is enabled by sliding windows, image pyramid, and bounding box that we developed. As we confirmed in the result section, our unique Model 2 successful detected object in test photos without transfer learning models. Using existing models such as ResNet model provided us a higher accuracy in validation data. However, we observed that it did not improve our performance of object detection, because it still suffers from false positive detection while the probabilities became higher.

Some test photos prevented us from detecting objects with both our models and transfer learning models. We consider the two reasons; the lack of data augmentation, and the quality of training dataset. There are several wrong predictions which we suppose are made from the color in the test

picture, not from the shape of the object. For example, the image classification model feeds in green area, and detects it as a frog with higher probabilities than the main object. We suppose it could be solved by adding training data which only changes color intensity, but keeps the shape of the objects with the same label class. Besides trying regularization method such as data augmentation, we consider that the resolution of the training data is not high enough for out object detection mission. Since our training picture size is extremely small, and the edge of some objects are not clearly identifiable, we think the model is mainly learning the color but not shape. With these potential approach and improvement, we prospect to decrease false positiveness and succeed more in object detection with custom original model as well as transfer learning model.

The remaining tasks for detection algorithm are the following; reducing the expensive computation of sliding windows and image pyramid, and multi objects detection. In the real photos, we faced different sizes of pictures so that sometimes if the model slides every pixel, it has to make too many predictions. Hence, we should have automated program that reads the size of the photos, and decides how many floors the model should go through in the pyramid, what the appropriate resize is, and what intervals the pixels should be skipped to mitigate computationally expensive predictions. In multi objects test case, because our main approach was to take average of confident locations, it ended up with single bounding boxes around the objects that the model is the most confident with. Research on this point is required for further attempts of object detection.

7. Bibliography

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications arXiv:1704.04861

He, K. (2016). Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., . . . Adam, H. (2019). Searching for MobileNetV3. *arXiv: Computer Vision and Pattern Recognition*. Retrieved 11 20, 2019, from https://arxiv.org/abs/1905.02244

Image Pyramids. (n.d.). Retrieved from OpenCV: https://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html

Koustubh. (2017, August). *ResNet, AlexNet, VGGNet, Inception: Understanding Various Architectures of Convolutional Networks*. Retrieved November 2019, from CV-Tricks: cv-tricks.com/cnn/understand-rsnet-alexnet-vgg-inception/

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images.* Retrieved 2019, from University of Toronto.

Olga, R., Deng, J., Su, H., Krause, J., Satheesh, S., & Sean Ma. (2015, January 30). *ImageNet Large Scale Visual Recognition Challenge.* Retrieved October 2019, from Cornell University.

Olivas, E. (2010). *Handbook of Research on Machine Learning Applications and Trends Algorithms, Methods, and Techniques.* Information Science Reference.

Rosebrock, A. (2015, March 23). *Sliding Windows for Object Detection with Python and OpenCV*. Retrieved from pyimages: https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

Tsang, S.-H. (2019, March 20). *Review: DenseNet – Dense Convolutional Network (Image Classification).* Retrieved November 2019, from Towards Data Science.

Tsang, S.-H. (2019, August 1). *Review: NASNet-Neural Architecture Search Network (Image Classification).* Retrieved November 2019, from Medium.

Wu, X. (2019). *Detection, Recent Advances in Deep Learning for Object.* Salesforce Research Asia.

8. Appendix

8.1 Codes for Models

Model 1

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding = 'same', activation = 'relu', input_shape = x_train.shape[1:]))
model.add(Conv2D(32, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Flatten())
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'sigmoid'))
```

Model 2

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))

model.add(Conv2D(32, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

MobileNet

```
model = Sequential()
model.add(MobileNet)
model.add(Flatten())
model.add(Dense(10, activation = 'sigmoid'))
```

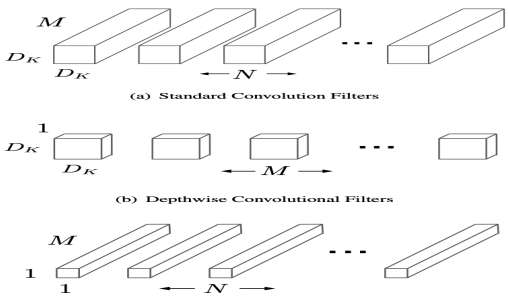
ResNet

```
model_transfer_ResNet50 = tf.keras.Sequential([
    ResNet,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(512, activation = 'relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation = 'softmax')
    ])
```

MobileNet Model:

Basic structures:

The MobileNet model is based on depthwise separable convolutions. It's a form that factorize a standard convolution into a depthwise convolution and a pointwise convolution.



It can substantially reduce the computational cost. For a standard convolution, its computational cost is DK · DK · M · N · DF · DF. However, after conducting the factorization, the computational cost can be separated into two parts and equals DK · DK · M · DF · DF + M · N · DF · DF. To compare the two results, we have

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

From above, the cost of a depthwise separable convolution decreases by N times than a standard convolution. This means the dramatic increase of cost efficiency and the decrease of latency.

Network structure and training:

The structure of MobileNet is as follows. The model spends 95% computational cost in dense 1x1 convolutions. In addition , 75% of the parameters are used in 1x1 convolutions. MobileNet models were trained in Tensorflow with RMSprop. It uses less regularization and data augmentation techniques.

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	$1 \times 1 \times 1000$

Width Multiplier:

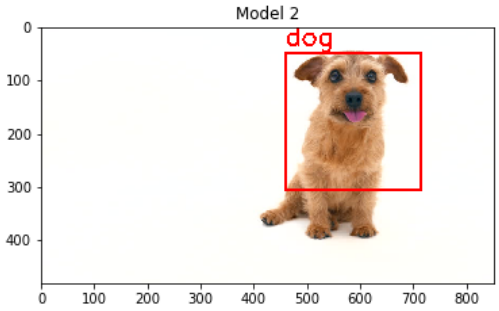
A simple hyper-parameter α, which is called width multiplier can adjust the thinness of layers. It thinks all layers in the network uniformly. It can help the model become smaller and faster when needed. With the parameter, the computational cost of a depthwise convolution changes to $D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$, where $\alpha \in (0, 1)$.

Resolution Multiplier:

Another hyper-parameter ϱ, which is called resolution multiplier, can adjust the resolution of the input and the internal representation of every layer. The computational cost of a depthwise convolution changes to $D_K \cdot D_K \cdot \alpha M \cdot \varrho D_F \cdot \varrho D_F + \alpha M \cdot \alpha N \cdot \varrho D_F \cdot \varrho D_F$, where $\alpha \in (0, 1)$ and $\varrho \in (0, 1)$.

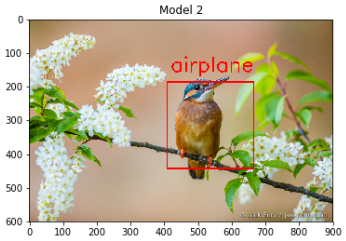
8.2 Test Results

- dog_01.jpg



Our Model 2 succeeded in detecting object and correctly labeled the object as a dog.

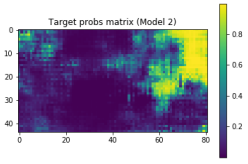
- bird_01.jpg

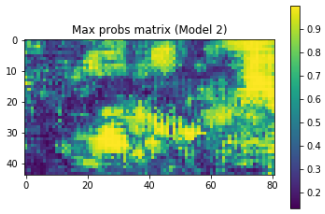


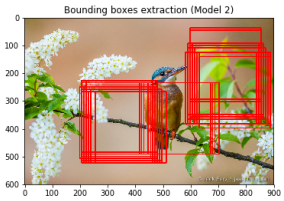
Object labeling

```
### Label
label_index = int(Counter(confident_id
label_pred = label_name[label_index]
print(Counter(confident_idx_x8))
print(label_index)
print(label_pred)
```

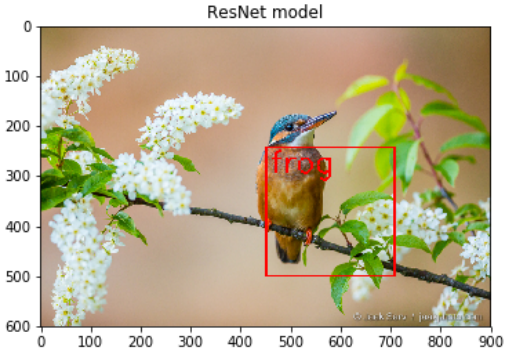
```
Counter({0.0: 19, 2.0: 18, 9.0: 4})
0
airplane
```

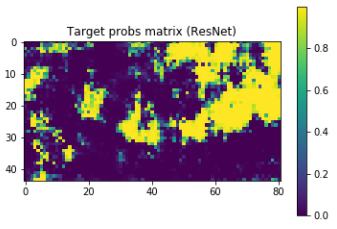


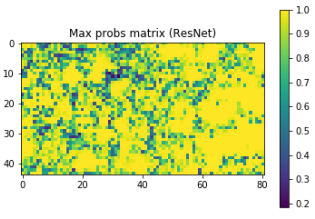


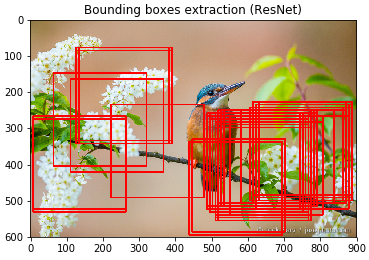


Object labels are 0: airplane, 2: bird, and 9: truck, so in this object detection, 19 out of 41 were airplane, and 18 out of 41 were actually bird. We would be successful with improved models. The following is the result from ResNet model.



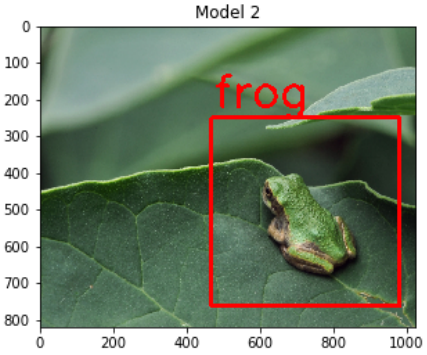






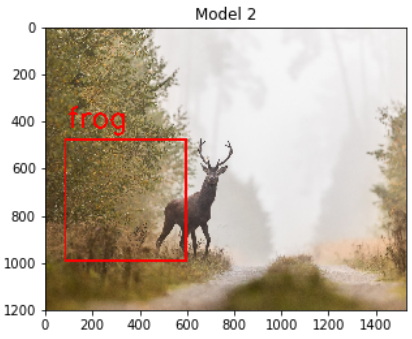
Since we failed to detection object, we applied ResNet transfer learning model. While we have many bird predictions with high probabilities, it produced probabilities which is higher than bird predictions. So due to false positive, ResNet model also is not successful in detection. Interestingly, ResNet model gave many predictions of frog with high probabilities.

- frog_01.jpg

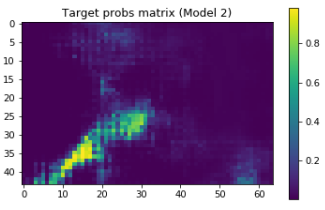


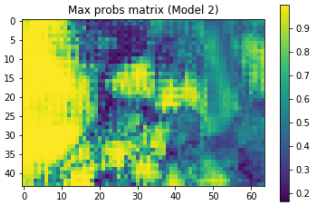
Our Model 2 succeeded in detecting object and correctly labeled the object as a frog.

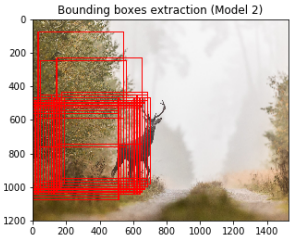
- deer_02.jpg



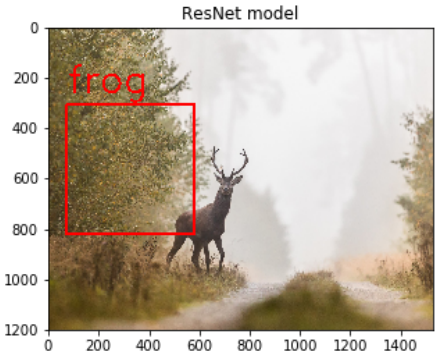

```
Counter({6.0: 52})
6
frog
```

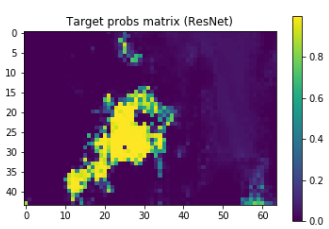


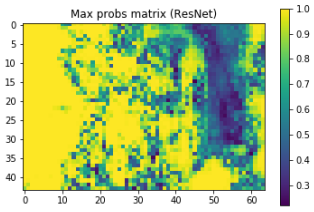


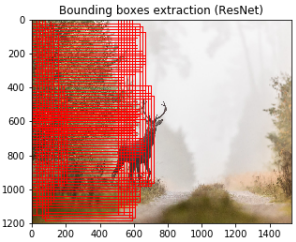


This is an unfortunate result because the detection suffers from much false negative. The image classification model captured the green parts at the left side of the picture and predicted them to be frogs. Probability predictions also suggests that we do not have high probability around the locations around the deer, shown at the left picture. But we have much many higher false positive as you can see in the middle. We assume maybe the deer on this picture is darker than our training data or because the deer looks at our direction but maybe training images contain the image from the side.



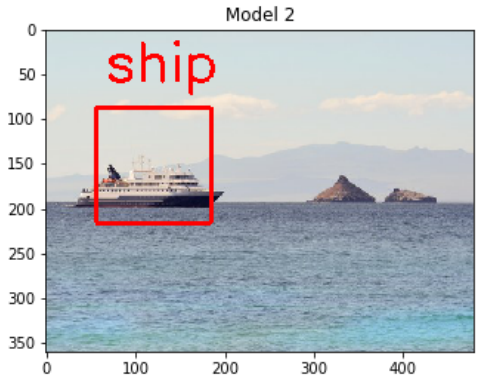






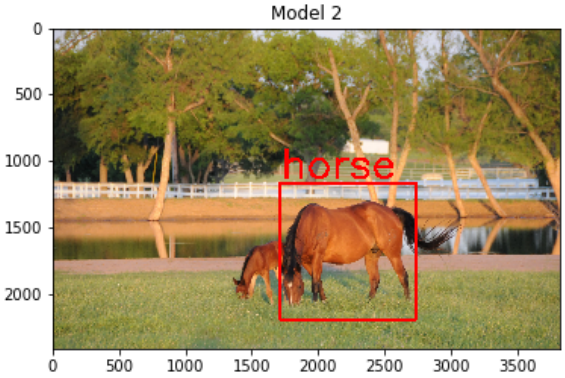
Since our Model 2 failed to detection the object, we applied ResNet model. As the probability picture at the left shows, ResNet model can capture the deer more accurately than Model 2. However, ResNet model also suffers from false positive and predicted the woods to be frog, and failed to detect the deer.

- ship_01.jpg



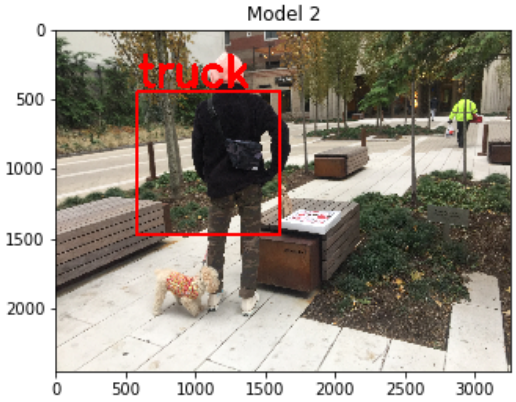
Our Model 2 succeeded in detecting object and correctly labeled the object as a ship.

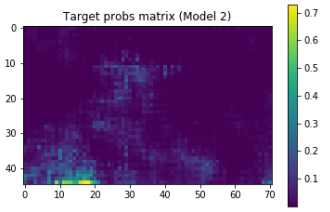
- hourse_01.jpg

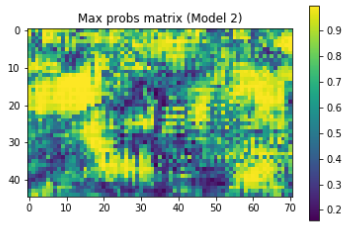


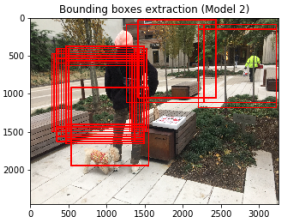
Our Model 2 succeeded in detecting object and correctly labeled the object as a horse.

- dog_03.jpg

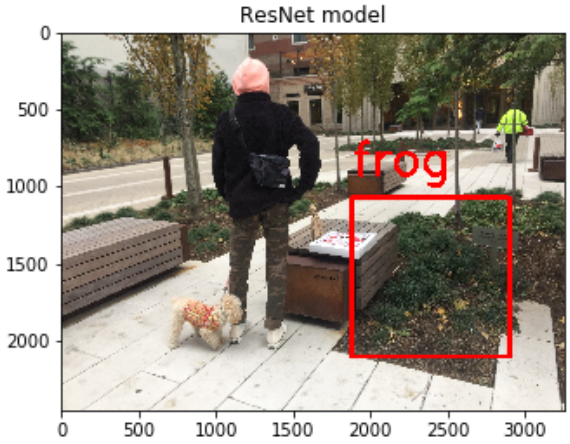


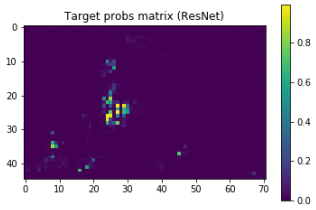


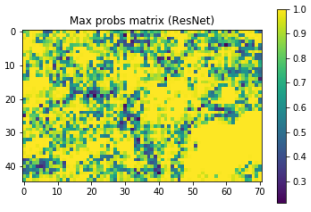


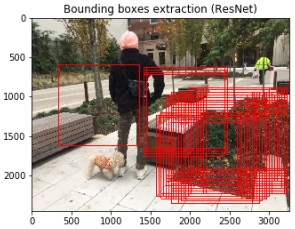


Object detection failed in this test photo. Our classification model does not have human class, but our detection has high probability around human object so it is false negative. The image in the above middle suggests that we in fact made prediction of dogs at the bottom left, but their probabilities are 73% at maximum. Hence, due to high false positive around many locations in the test photo as shown in the image in the above right, we failed to detection a dog. The next is the results from ResNet model.









It is difficult even for ResNet model to detect a dog. We think it is maybe because the color of dog is very similar to the color of the ground and the picture contains so many different objects. ResNet model captures the green leaf parts again in this picture, and predicted them to be frogs.

8.3 Image Sources

- dog_01.jpg
    - https://merryjane.com/culture/barking-up-the-right-tree-should-you-give-cbd-to-your-dog
- ship_01.jpg
    - https://www.cntraveler.com/story/what-happens-during-a-cruise-ship-rescue
- hourse_01.jpg
    - https://www.discoverdenton.com/what-to-do/attractions/horse-country/
- frog_01.jpg
    - https://karenrexrode.typepad.com/studiology/2016/06/gray-tree-frog.html
- automobile_truck_02.jpg
    - https://www.wmur.com/article/government-moves-toward-easing-drive-time-rules-for-truck-drivers/28702834#
- deer_02.jpg
    - https://pixnio.com/tag/deer
- airplane_02.jpg
    - https://milvarusso.com/blog/seven-days-at-katathani-phuket-beach-resort/
- bird_01.jpg
    - http://www.jaakphoto.com/en/nature/birds/kingfisher-lady-on-bird-cherry-tree/
- cat_02.jpg

- https://www.quartertothree.com/fp/2011/10/07/visit-a-magical-never-before-seen-wonderland-in-sims-3-pets/

- dog_03.jpg
  - Chen's personal picture

8.4 Detection Algorithm

- Sliding windows

```
### matrix which contains predictions

# maximum probability over 10 classes
prob_matrix = np.zeros((pred_height, pred_width))
# argmax predicted labels
idx_matrix = np.zeros((pred_height, pred_width))
# target probabilities
target_prob_matrix = np.zeros((pred_height, pred_width))

# set input
img_proc = img_1_x2_proc

for j in range(pred_height):
    for i in range(pred_width):
        # sliding windows
        tmp = img_proc[:, j:(input_size+j), i:(input_size+i), :]
        # prediction from neural network image classificaion models
        pred_tmp = loaded_model.predict(tmp)
        prob_max = np.max(pred_tmp)
        idx_max = np.argmax(pred_tmp)
        prob_target = pred_tmp[0][idx_target]

        # store values
        prob_matrix[j, i] = prob_max
        idx_matrix[j, i] = idx_max
        target_prob_matrix[j, i] = prob_target
```

- Image pyramid

```
### image pyramid

# resize parameter (for example, a half)
size = 2
location = []
prob_array = prob_matrix.flatten(order = "C")
idx_array= idx_matrix.flatten(order = "C")
n_prob_height = prob_matrix.shape[0]
n_prob_width = prob_matrix.shape[1]
ratio_height = (img_1.size[1] - input_size * size + 1) / n_prob_height
ratio_width = (img_1.size[0] - input_size * size + 1) / n_prob_width

# calculate location coordinates
for i in range(n_prob_height):
    for j in range(n_prob_width):
        tmp = [int(np.round(i * ratio_height, decimals = 0)),
               int(np.round(j * ratio_width, decimals = 0)),
               int(input_size * size + np.round(i * ratio_height, decimals = 0)),
               int(input_size * size + np.round(j * ratio_width, decimals = 0))]

        # store location infomation
        location.append(tmp)

# extract confident locations
confident_location = np.array(location)[prob_array > cutoff]
# extract confident labels
confident_idx = idx_array[prob_array > cutoff]
```

- Taking average for multiple bounding boxes suppression

```
### taking average to suppress multiple bounding boxes

n = len(confident_location)

loc_1 = []
for i in range(n):
    loc_1.append(confident_location[i][0])
loc_2 = []
for i in range(n):
    loc_2.append(confident_location[i][1])
loc_3 = []
for i in range(n):
    loc_3.append(confident_location[i][2])
loc_4 = []
for i in range(n):
    loc_4.append(confident_location[i][3])

### taking mean over coordinates
loc_all = []
loc_all.append(int(np.mean(loc_1)))
loc_all.append(int(np.mean(loc_2)))
loc_all.append(int(np.mean(loc_3)))
loc_all.append(int(np.mean(loc_4)))
```

- Majority vote object labeling

```
### majority vote object labeling

label_name = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer',
                       'dog', 'frog', 'horse', 'ship', 'truck'])
label_index = int(Counter(confident_idx).most_common(1)[0][0])
# predicted object label
label_pred = label_name[label_index]
```