# Deep Learning Vehicle Detection with Raspberry Pi

# Object Detection

When it comes to deep learning-based object detection there are three primary object detection methods that you'll likely encounter:

- Faster R-CNNs - most popular method but much slower(7FPS) because of region proposal algorithm
- You Only Look Once (YOLO) - much faster but leaves behind the desired accuracy
- Single Shot Detectors (SSDs) - faster than Faster R-CNNS and much more accurate than YOLO

| System | VOC2007 test *mAP* | FPS (Titan X) | Number of Boxes | Input resolution |
|---|---|---|---|---|
| Faster R-CNN (VGG16) | 73.2 | 7 | ~6000 | ~1000 x 600 |
| YOLO (customized) | 63.4 | 45 | 98 | 448 x 448 |
| SSD300* (VGG16) | 77.2 | 46 | 8732 | 300 x 300 |
| SSD512* (VGG16) | **79.8** | 19 | 24564 | 512 x 512 |

# MODEL

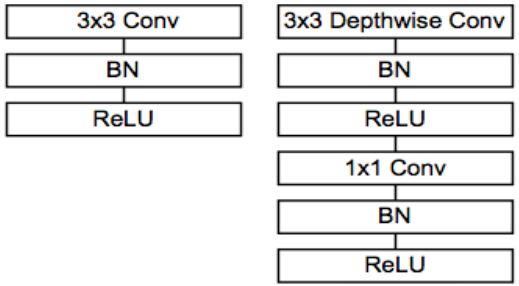
# SSD

The SSD object detection composes of 2 parts:

- Extract feature maps - MobileNet
- Apply convolution filters to detect objects.

# MobileNet

- When building object detection networks we normally use an existing network architecture, such as VGG or ResNet, and then use it inside the object detection pipeline. The problem is that these network architectures can be very large in the order of 200-500MB.
- Instead, we can use MobileNet,  paper by Google researchers. We call these networks “MobileNets” because they are designed for resource constrained devices such as your smartphone. They use the *depthwise separable convolution*(which reduces to almost half computation)




*(Left)* Standard convolutional layer with batch normalization and ReLU. *(Right)* Depthwise separable convolution with depthwise and pointwise layers followed by batch normalization and ReLU

# Depthwise Separable Convolution

- The Depthwise Separable Convolution is so named because it deals not just with the spatial dimensions, but with the depth dimension.
- The general idea behind depthwise separable convolution is to split convolution into two stages:

1. A *3×3* depthwise convolution.
2. Followed by a *1×1* pointwise convolution.

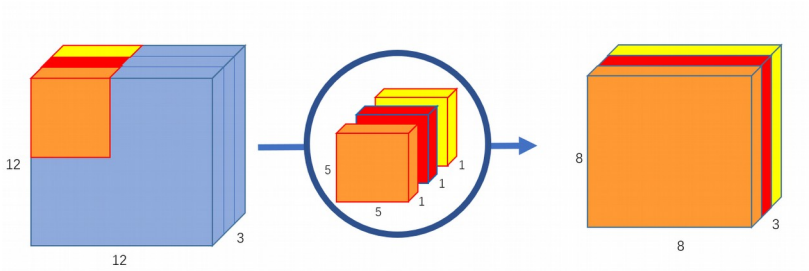



Image 1: Depthwise convolution, uses 3 kernels to transform a 12x12x3 image to a 8x8x3 image

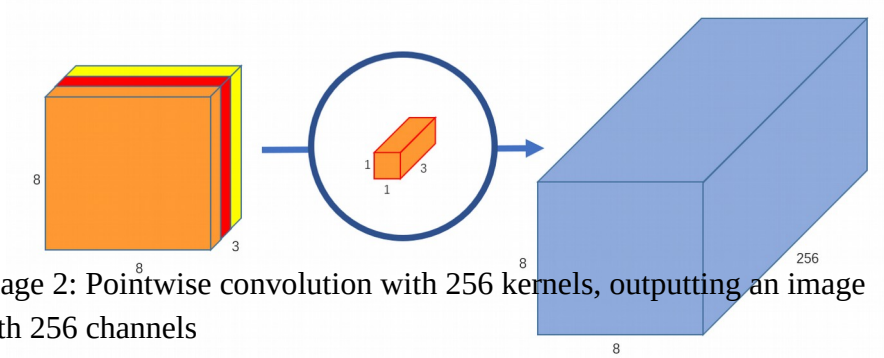



Image 2: Pointwise convolution with 256 kernels, outputting an image with 256 channels

# MobileNet Architec

- First layer is full convolution layer
- All layers are followed by a batchnorm and ReLU nonlinearity with the exception of the final fully connected layer which has no nonlinearity and feeds into a softmax layer for classification
- Down sampling is handled with strided convolution in the depthwise convolutions as well as in the first layer
- A final average pooling reduces the spatial resolution to 1 before the fully connected layer
- Counting depthwise and pointwise convolutions as separate layers, MobileNet has 28 layers
- MobileNet spends 95% of it's computation time in 1 × 1 convolutions which also has 75% of the parameters
- MobileNet models were trained in TensorFlow using RMSprop with asynchronous gradient descent similar to Inception V3
- However, contrary to training large models we use less regularization and data augmentation techniques because small models have less trouble with overfitting

Table 1. MobileNet Body Architecture

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

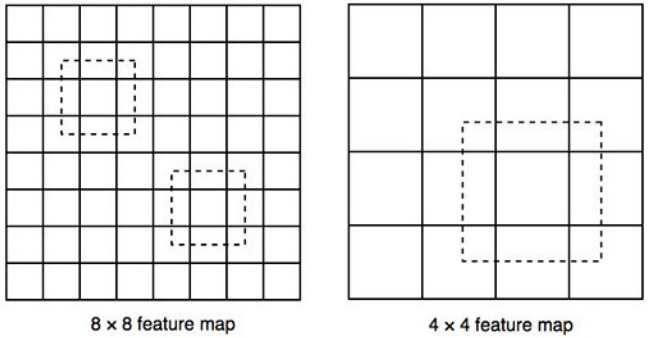
# Object detection using MobileNet

- The MobileNet SSD was first trained on the COCO dataset and was then fine-tuned on PASCAL VOC reaching 72.7% mAP (mean average precision).
- In Table, MobileNet is compared to VGG and Inception V2 under both Faster-RCNN and SSD framework .
- SSD is evaluated with 300 input resolution (SSD 300) and Faster-RCNN is compared with both 300 and 600 input resolution (Faster- RCNN 300, Faster-RCNN 600)
- So, from the table you can conclude that for lower resolution images(300) SSD provides better precision and less computation

| Framework Resolution | Model | mAP | Billion Mult-Adds | Million Parameters |
|---|---|---|---|---|
| SSD 300 | deeplab-VGG | 21.1% | 34.9 | 33.1 |
| | Inception V2 | 22.0% | 3.8 | 13.7 |
| | MobileNet | 19.3% | 1.2 | 6.8 |
| Faster-RCNN 300 | VGG | 22.9% | 64.3 | 138.5 |
| | Inception V2 | 15.4% | 118.2 | 13.3 |
| | MobileNet | 16.4% | 25.2 | 6.1 |
| Faster-RCNN 600 | VGG | 25.7% | 149.6 | 138.5 |
| | Inception V2 | 21.9% | 129.6 | 13.3 |
| | Mobilenet | 19.8% | 30.5 | 6.1 |

# Convolution Filter For Object detection

- SSD uses **MobileNet** to extract feature maps and Then it detects objects using the **Conv4_3** layer. (it is  38x38 spatially)
- For each cell (also called location), it makes 4 object predictions.
- Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and we pick the highest score as the class for the bounded object.
- Conv4_3 makes a total of 38 × 38 × 4 predictions: four predictions per cell regardless of the depth of the feature maps. As expected, many predictions contain no object. SSD reserves a class “0” to indicate it has no objects.
- It uses multiple layers (**multi-scale feature maps)** to detect objects independently. As CNN reduces the spatial dimension gradually, the resolution of the feature maps also decrease. SSD uses lower resolution layers to detect larger scale objects. For example, the 4× 4 feature maps are used for larger scale object.



# contd..

- SSD adds 6 more auxiliary convolution layers after the MobileNet. Five of them will be added for object detection. In three of those layers, we make 6 predictions instead of 4. In total, SSD makes 8732 predictions using 6 layers.

## Deep learning-based vehicle detection with OpenCV on Raspberry Pi (Model1)

- In this,  we will use the MobileNet SSD + deep neural network ( dnn ) module in OpenCV to build our object detector.
- We initialize our VideoStream using a USB camera as we are using the Raspberry Pi
- Load Model (is a Caffe version of the original TensorFlow implementation )
- Reading frame from video stream and then resizing it for our model.
- Preprocessing - Mean subtraction, scaling and optionally channel swapping
- Output of preprocessing block is taken as input of the network and then compute the forward pass for the input, storing the result.
- As  multiple objects can be detected in a single image so, we  also apply a check to the confidence (i.e., probability) associated with each detection. If the confidence is high enough (i.e. above the threshold), then we'll display the prediction in the terminal as well as draw the prediction on the image with text and a colored bounding box.
- Results we are obtaining ~0.9 frames per second throughput using this method and the Raspberry Pi compared to the 6-7 frames per second using our laptop/desktop we can see that the Raspberry Pi is *substantially* slower.

# Deep) learning-based vehicle detection with OpenCV on Raspberry Pi (Model2

- Using the example from the previous section we see that forward path  is a ***blocking operation*** — the rest of the code in is not allowed to complete until forward path returns the detections.
- But what we *can* do is create a separate process that is solely responsible for applying the deep learning object detector, thereby unblocking the main thread of execution and allow our rest of code to continue.
- Moving the predictions to separate process will give the *illusion* that our Raspberry Pi object detector is running faster than it actually is, when in reality the forward  computation is still taking a little over one second.
- The only problem here is that our output object detection predictions will lag behind what is currently being displayed on our screen. If you detecting *fast-moving objects* you may miss the detection entirely, or at the very least, the object will be out of the frame before you obtain your detections from the neural network..
- Therefore, this approach should only be used for *slow-moving objects* where we can tolerate lag
- Here you can see that our other part of code  is capable of processing 27 frames per second. However, this throughput rate is an illusion — the neural network running in the background is still only capable of processing 0.9 frames per second.
- Therefore, you should ***consider only using this approach when***:

1. **Objects are slow moving and the previous detections can be used as an approximation to the new location**.
2. **Displaying the actual frames themselves in real-time is paramount to user experience.**

# THANK YOU !!