
Lyft Perception Challenge
BROUGHT TO YOU BY
UDACITY
lyft

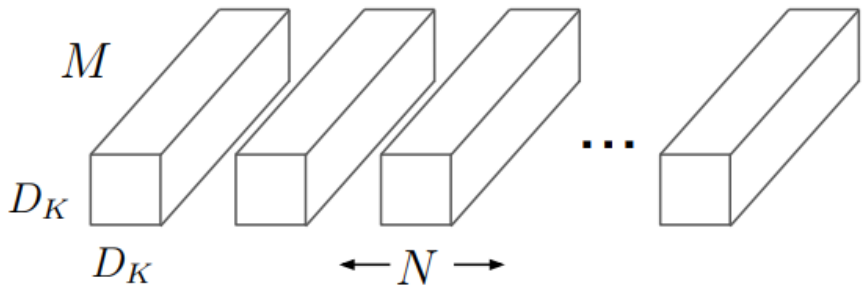
Lyft perception challenge

Overview

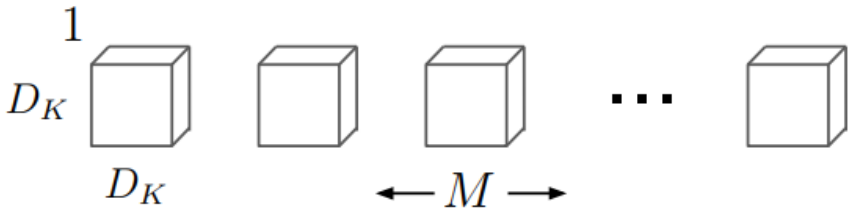
This project was conducted for Lyft perception challenge. It seems that there is a similar project at term 3 of Udacity self-driving car nanodegree program, but I am still at term 2 of the nanodegree, and I have no idea what that project is at term 3. I only found out that VGG16 pretrained model is recommended in term 3 project 2. However, I have used MobileNet instead of VGG since MobileNet is nearly as accurate as VGG16 while being 32 times smaller and 27 times less compute intensive [1].

MobileNet is a neural network architecture that strives to minimize latency of smaller scale networks. Basically, it makes neural network much lighter and portable for mobile and embedded applications using depth-wise separable convolution. MobileNet uses depthwise separable convolution in its 28 layer architecture. For MobileNet the depthwise convolution applies a single filter to each input channel. Then, the pointwise convolution applies a 1x1 convolution to combine the outputs of the depthwise convolution.

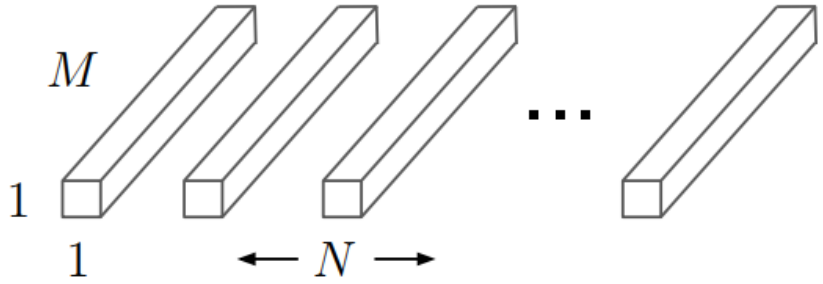
Simply speaking, regular convolution attempts to capture two things simultaneously: the relationship inside a channel, and the relationship between channels. But depthwise separable convolution aims to break this into two parts, First is depthwise convolution which only capture the relationship inside one particular channel, and the second is pointwise convolution which finally find the relationship between channels. Figure 1 below illustrates this difference.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



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

Figure 1. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

Figure 1-a shows the regular convolution filters which has the size of D_K x D_K x M and N is the number of output channel (output depth). The standard convolutional layer is parameterized by convolution kernel K of size D_K x D_K x M ×N, and it has the computational cost of D_K x D_K x M x N x D_F x D_F where D_F is the spatial dimension of the input assumed to be square.

Figure 1-b depicts depthwise convolution filter aiming to examine each channel independently. The filter size here in this part is D_K x D_K x 1. The computational cost of this filtering part is D_K x D_K x 1 x D_F x D_F x M.

Additionally, Figure 1-c which shows the second part of depthwise separable convolution layer has the filter size of 1 x 1 x M. The computational cost of this equals to M x N x D_F x D_F.

Finally, the total computational cost of depthwise separable convolution is the summation of first and second parts' costs which equals to D_K x D_K x 1 x D_F x D_F x M + M x N x D_F x D_F.

Now, it's time to compare costs of regular convolution versus depthwise separable convolution. By expressing convolution as a two-step process of filtering and combining we get a reduction in computation of:

$$\frac{\mathrm{DK\ x\ DK\ x\ 1\ x\ DF\ x\ DF\ x\ M\ +\ M\ x\ N\ x\ DF\ x\ DF}}{\mathrm{DK\ x\ DK\ x\ M\ x\ N\ x\ DF\ x\ DF}}$$

$$= \frac{1}{\mathrm{N}} + \frac{1}{\mathrm{DK}^2}$$

It simply shows that if you use 3 x 3 depthwise separable convolutions which I used for this challenge, the computing power decreases between 8 to 9 times than standard convolution for one single convolution layer which is amazing.

In MobileNet, all layers are depthwise separable convolutions except for the first layer which is a full convolution. Also, all layers are followed by a batchnorm and ReLU with the exception of the final fully connected layer which has no linearity and feeds into a softmax layer for classification. The difference of my algorithm with the standard convolution is that in MobileNet each filtering layer (depthwise and pointwise) is followed by batchnorm and ReLU. Figure 2 below illustrates this difference.

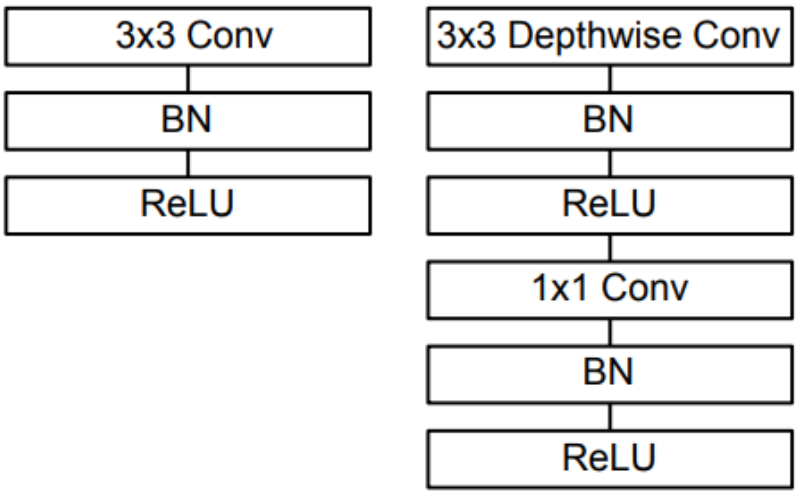


Figure 2. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Attempting to reduce computational cost absolutely leads to less accuracy, but what is amazing about depthwise separable convolution is that its accuracy decreases one percent at the worst case scenario. As an example, MobileNet with the fully connected layers network has 29.3 million parameters while it reduces to 4.2 million parameters using MobileNet with depthwise separable convolution layers, while the accuracy on ImageNet drops by only 1 percent. Moreover, the number of multiplications and additions which is a direct measure of computation has also significantly decreased for depthwise separable convolution MobileNets (from 4866 million to 569 million). In the paper [1], authors also defined other hyper-parameters for reducing the computational cost of the model which I have not used those methods in the challenge. So far, I hope that I could show why I used depthwise separable convolution layers instead of standard convolution layers.

Now, I want to show a comparison between MobileNet and VGG 16 which shows again why I decided to make use of MobileNet instead of VGG 16. For instance, by using MobileNet and VGG 16 on ImageNet, the number of parameters falls from 138 to 4.2 million, and the number of multiplications and additions significantly decreases from 15300 million to 569 million which is extremely amazing for computing power. On the other hand, the accuracy drops by only 1 percent [1].

In my algorithm, I made use of dilated depthwise convolutions [2] for all stride-16 depthwise convolutions similar to the paper [2]. Basically, dense prediction problems such as semantic segmentation calls for multiscale contextual reasoning in combination with full-resolution output. In paper [2], authors develop a convolutional network module that aggregates multi-scale contextual information without losing resolution or analyzing rescaled images. The module can be plugged into existing architectures at any resolution. Unlike pyramid-shaped architectures carried over from image classification, the presented context module is designed specifically for dense prediction. It is a rectangular prism of convolutional layers, with no pooling or subsampling. The module is based on dilated convolutions which support exponential expansion of the receptive field without loss of resolution or coverage.

In my algorithm, I also removed the two final stride-32 layers. This is similar to what is done in paper [2]. Keras MobileNet implementation is utilized for the model and training is done with TensorFlow. Adam optimizer was used as a well-established optimizer. Hyper parameters were chosen by the try-and-error process. After many try-and-error process, I finally utilized these hyper parameters:

- Learning rate: 0.001
- Decay = learning rate/ (2*epochs)
- Epochs: 40
- Batch size: 8

Below you can see the model architecture I used in the challenge.

```
def reducedMobileNet(input_height, input_width):
    assert input_height // 16 * 16 == input_height
    assert input_width // 16 * 16 == input_width
    resolution_multiplier = 1 # □resolution_multiplier = 1 is the baseline
MobileNet and □ < 1 are reduced
                              # computation MobileNets.
                              # Resolution multiplier has the effect
                              # of reducing computational cost by □2.

    img_input = Input(shape=[input_height, input_width, 3],
name='image_input')
    # s / 2
    x = _conv_block(img_input, 32, alpha, strides=(2, 2))
    x_s2 = _depthwise_conv_block(x, 64, alpha, resolution_multiplier,
block_id=1)
    # s / 4
    x = _depthwise_conv_block(x_s2, 128, alpha, resolution_multiplier,
                              strides=(2, 2), block_id=2)
    x_s4 = _depthwise_conv_block(x, 128, alpha, resolution_multiplier,
block_id=3)
    # s /  8
    x = _depthwise_conv_block(x_s4, 256, alpha, resolution_multiplier,
                              strides=(2, 2), block_id=4)
    x_s8 = _depthwise_conv_block(x, 256, alpha, resolution_multiplier,
block_id=5)
    # s / 16
    # Based on the paper cited in the write-up,
    #all stride-16 depthwise convolutions are substitude with dilated
depthwise convolutions with rates =1,2, and 4.

    x = _depthwise_conv_block(x_s8, 512, alpha, resolution_multiplier,
                              strides=(2, 2), block_id=6)
    x = _depthwise_conv_block(
        x, 512, alpha, resolution_multiplier, dilation_rate=1, block_id=7)
    x = _depthwise_conv_block(
        x, 512, alpha, resolution_multiplier, dilation_rate=1, block_id=8)
    x = _depthwise_conv_block(
        x, 512, alpha, resolution_multiplier, dilation_rate=2, block_id=9)
    x = _depthwise_conv_block(
        x, 512, alpha, resolution_multiplier, dilation_rate=4, block_id=10)
    x_s16 = _depthwise_conv_block(
```

```
        x, 512, alpha, resolution_multiplier, dilation_rate=4, block_id=11)

    return img_input, x_s2, x_s4, x_s8, x_s16


def SegMobileNet(input_height, input_width, num_classes=21):
    assert input_height // 16 * 16 == input_height
    assert input_width // 16 * 16 == input_width
    img_input, x_s2, x_s4, x_s8, x_s16 = reducedMobileNet(input_height,
input_width)

    x_up8 = Conv2DTranspose(int(256*alpha), (3,3), strides=(2, 2),
data_format='channels_last', padding='same', name='deconvolution8')(x_s16)
    x_s8 = Add(name='add_s8')([x_up8, x_s8])

    x_up4 = Conv2DTranspose(int(128*alpha), (3,3), strides=(2, 2),
padding='same', name='deconvolution4')(x_s8)
    x_s4 = Add(name='add_s4')([x_up4, x_s4])

    x_up2 = Conv2DTranspose(int(64*alpha), (3,3), strides=(2, 2),
padding='same', name='deconvolution2')(x_s4)
    x_s2 = Add(name='add_s2')([x_up2, x_s2])

    x = Conv2DTranspose(num_classes, (3,3), strides=(2, 2), padding='same',
activation='softmax', name='deconvolution1')(x_s2)

    return Model(img_input, x, name='SegMobileNet')

def _conv_bn_pred(x, stride, num_classes=21):
    x = Conv2D(num_classes, 1, use_bias=False, padding='same',
               name='conv_pred_s%d' % stride)(x)
    x = BatchNormalization(name='conv_pred_s%d_bn' % stride)(x)
    return x


def _resize_bilinear(x, target_h, target_w):
    x = tf.image.resize_images(
        x,
        [target_h, target_w],
        method=tf.image.ResizeMethod.BILINEAR,
        align_corners=True)
    x.set_shape((None, target_h, target_w, None))
    return x


def _depthwise_conv_block(inputs, pointwise_conv_filters, alpha,
                          resolution_multiplier=1, strides=(1, 1),
block_id=1,
                          dilation_rate=1):
    # In MobileNet, all layers (both depthwise covolution layer and pointwise
convolution layer)
    # are followed by a batchnorm and ReLU. More explanation can be found in
the write-up.

    channel_axis = 1 if K.image_data_format() == 'channels_first' else -1
    pointwise_conv_filters = int(pointwise_conv_filters * alpha)
```

```
x = DepthwiseConv2D((3, 3),
                    padding='same',
                    resolution_multiplier=resolution_multiplier,
                    strides=strides,
                    use_bias=False,
                    name='conv_dw_%d' % block_id,
                    dilation_rate=dilation_rate)(inputs)
x = BatchNormalization(
    axis=channel_axis, name='conv_dw_%d_bn' % block_id)(x)
x = Activation(relu6, name='conv_dw_%d_relu' % block_id)(x)

x = Conv2D(pointwise_conv_filters, (1, 1),
           padding='same',
           use_bias=False,
           strides=(1, 1),
           name='conv_pw_%d' % block_id)(x)
x = BatchNormalization(
    axis=channel_axis, name='conv_pw_%d_bn' % block_id)(x)
return Activation(relu6, name='conv_pw_%d_relu' % block_id)(x)
```

I have to mention that since the dataset provided for the challenge was really small (1000 images), several augmentation methods were utilized, including rotation, flipping, blurring, change of illumination.

For training the model, I used GeForce GTX 1080 Ti.

At the end, I also have to mention that I inspired some parts of the code from https://github.com/see--/P12-Semantic-Segmentation.

References

1. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861. 2017 Apr 17.

2. Yu F, Koltun V. Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122. 2015 Nov 23.