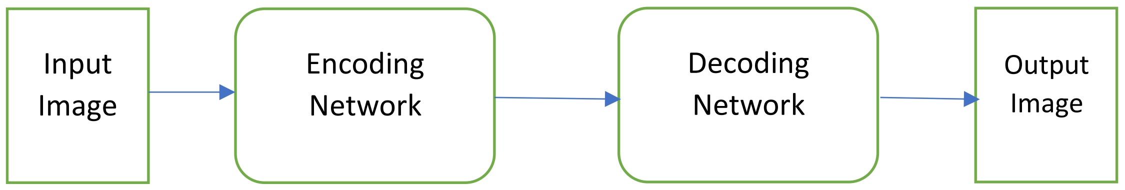
FOLLOW ME PROJECT REPORT

Network Architecture

The project uses Fully Convolutional Network (FCN) architecture. FCN model retains spatial information and therefore, is able to perform semantic segmentation for object recognition. Semantic segmentation is the task of assigning meaning to part of an object in the image.

FCN mainly consists of two blocks. Encoder network and Decoder network.



Below I will explain function and architecture of each of these blocks. Diagrams are provided later in the topic 'Final FCN Model Used in the Project'.

Note: For any reference to functions, refer model_training.ipynb notebook.

Encoding Network:

The main function of encoders is to extract features from an image. Encoders followed by a fully connected layer can be used for object recognition. The more the number of encoders, the better the network can perform at the cost of training speed. However, after a certain point, as the parameters increase lot more than required for the network, the network performance degrades.

I used five layers of encoding. Each encoding layer consists of separable convolution, RELU activation function, max pooling and batch normalization. Function of each of these layers is explained below:

Separable Convolutions –

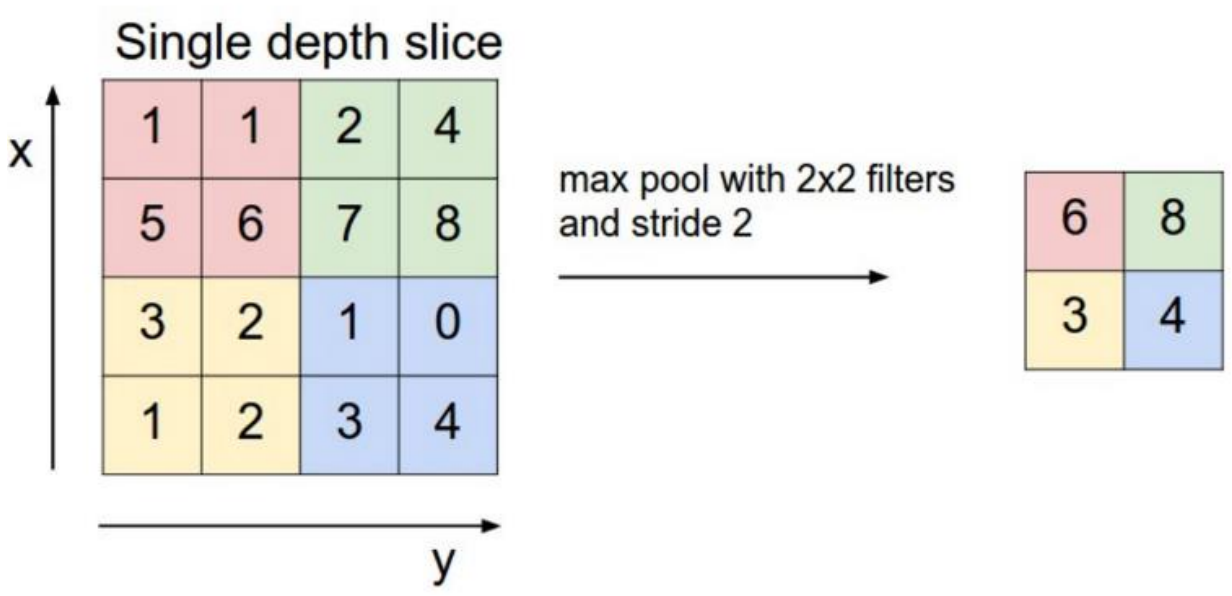
The advantage of using separable convolution is the parameter reduction, and consequently increased efficiency of encoding network. Furthermore, the reduced number of parameters are also benefiting in preventing overfitting to an extent.

Rectified Linear Unit (ReLU) activation –

ReLU is used to add non-linearity to the neural network. ReLU is defined as max(0, output). Therefore, it only allows to pass the positive outputs and clips off negative outputs to zero.

Max Pooling –

It is used to reduce the size of input image so that network can focus only on the most essential elements of the image. Max pooling does so by retaining only the maximum value for each filter area. Refer to figure below for the illustration.



Batch normalization –

Batch normalization normalizes the input to each layer within the network. For this, the values in current mini-batch are used. The concept here is that, the input to any layer is in fact the output of previous layer. Therefore, each layer can be considered as a neural network. Its advantages are as given below:

1. Networks train faster
   - Although each training iteration is slower due to additional computations, the network converges faster and overall training time is reduced.
2. Allows higher learning rate
   - As the network gets deeper, their gradients get smaller during back propagation and hence more iterations are required. However, batch normalization allows usage of higher learning rate increasing the network training speed.
3. Simplifies the creation of deeper networks
   - Due to increased training speed, deeper networks can be built.
4. Provides some regularization
   - By adding noise to the network, batch normalization acts as dropout.

Implementation in Keras as used in the notebook –

Convolution, ReLU activation and Max Pooling:

```
SeparableConv2DKeras(filters=filters,kernel_size=3, strides=strides, padding='same',
                          activation='relu')(input_layer)
```

<u>Batch normalization:</u>

```
layers.BatchNormalization()(output_layer)
```

Refer to function “encoding_block()” to see the implementation of encoder layer.

I started with just one layer, and then kept on adding more and more layers. With the addition of each layer, training loss reduced and consequently the final score IOU increased. This was because addition of each layer increased number of parameters and therefore the network could draw more features from an image. However, for low number of epochs (10-25), four layers deep network performed better than five layers deep network.

The encoding network ultimately reduces to a deeper 1x1 convolutional layer, which is explained below.

1x1 Convolutional Layer:

It is normal convolution with stride equal to one and is the last layer of encoding network. It serves three main purposes:

1. The input size of an image need not be specific
2. Spatial information is preserved (for semantic segmentation)
3. Depth of the network is increased with only small increase in computations

This layer is *implemented in Keras* using function *conv2D_batchnorm()* with both kernel size and stride equal to one.

<u>Decoding Network:</u>

Decoders are used for semantic segmentation. It allows to recognize where in the image the detected object is present. A decoder uses transposed convolution to upsample the image. It differs from convolution in that the forward and backward passes are swapped. Therefore, the property of differentiability is retained.

Implementation of a decoder has three main parts:

1. **Upsampling -**
   This is implemented by function BilinearUpSampling2D((2,2))(input_layer)
   The tuple (2,2) implies that the input image will be upsampled by factor of two across both width and height
2. **Skip connections -**
   These are implemented here using function layers.concatenate().
   Upsampling causes some loss in resolution. Adding skip connection from corresponding encoding layer improves the final output image resolution. Consequently, the network is able to make more precise segmentation decisions. For skip connections, concatenation is advantageous over

addition as it is computationally inexpensive and is also independent of the filter size of decoding layer and corresponding encoding layer.

3. **Convolutional layers -**
   These layers help in extracting some more spatial information from prior image. I have used two layers to improve final scores. These are implemented using function 'separable_conv2d_batchnorm()'.

Decoding layers are responsible for locating each detected object in the original image, i.e., classifying each pixel into one of the object pixel.

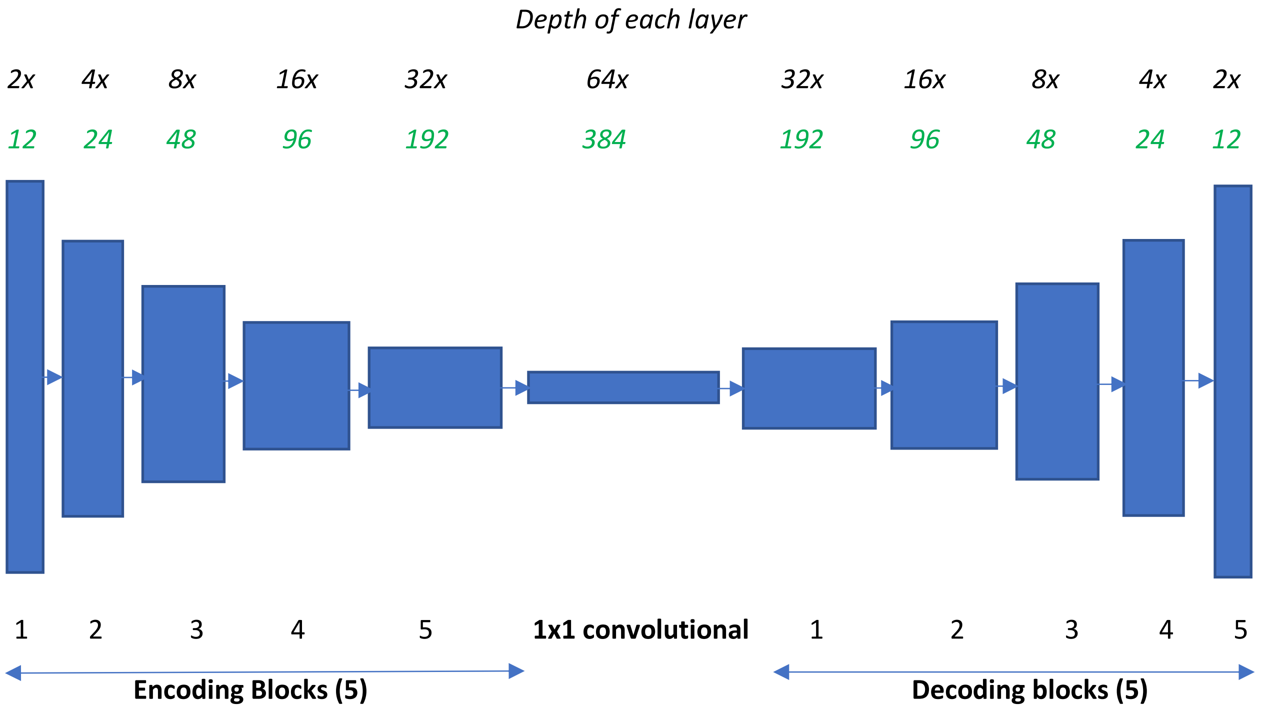
Final FCN Model used in the Project:

I started with one encoding layer, and then kept on adding more and more layers. With the addition of each layer, training loss reduced and consequently the final score IOU increased. This was because addition of each layer increased number of parameters and therefore the network could draw more features from an image. However, for low number of epochs (10-25), four-layer deep network performed better than five-layer deep network. Later on, when I increased number of epochs to more than 50, five-layer deep network performed better.

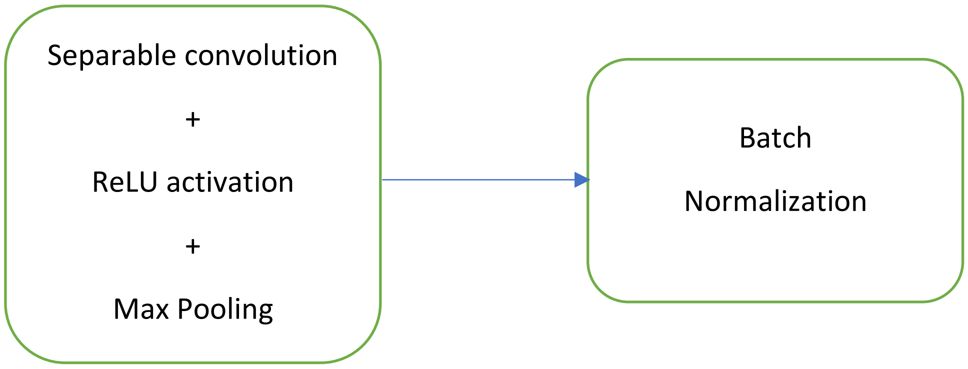
I followed symmetry in depth of each layer on encoding side and decoding side (Refer figure below). The optimum 'filters' parameter value (in function fcn_model()) was found to be six. Values other than six degraded the network performance.

At the end of encoding blocks 1x1 convolutional layer was used followed by decoding blocks. The decoding blocks are equal in number to that of encoding blocks.

Please refer figure below –

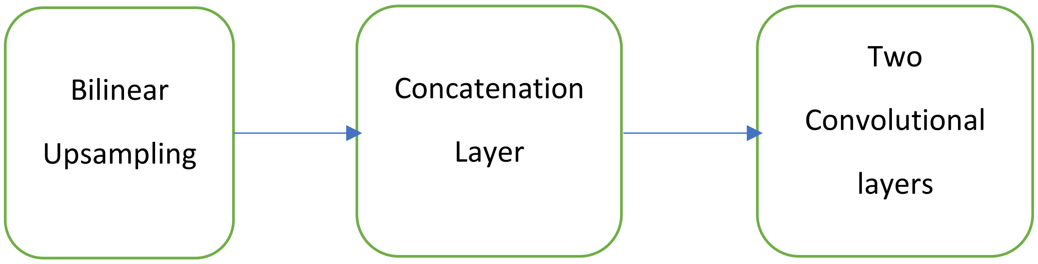


Components of each encoding block:



I have used convolution filters of size 3x3 and max pooling with stride 2.

Components of each Decoding block:



Parameter choosing for Neural Network

The parameters to choose were mainly epochs, learning rate, batch size, number of steps per epoch for training and validation data. Below, I will discuss one parameter at a time and then how it affects the output score. I will also explain how I reached to my final values of parameters.

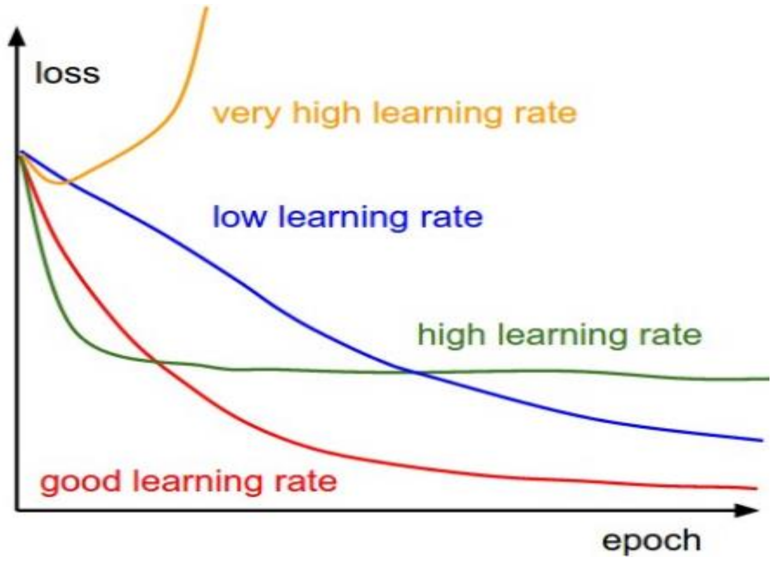
Epochs:

This number determines how many times the correction is made. As the number of epochs increases, the network learns better and better until one point, after which the network starts to overfit, i.e., it performs really well on the training data set but performs poorly on any other data set than training data set. Determining when overfitting starts was challenging, as the network performance does not gradually increase but fluctuates even before the network reaches the saturation point.

Graph of no of epochs vs training and validation loss-

Learning rate:

Learning rate determines how fast and how accurately the network can learn. Higher learning rate means that with each epoch, the weight corrections made by backpropagation are large in magnitudes. Consequently, the network initially learns faster. However, due to large values, the network accuracy is quite low. On the other hand, smaller learning rate means that network learns slower, i.e., it takes more number of epochs to improve accuracy significantly. However, small learning rate allows to decrease the loss further than that achieved with a large learning rate. This is illustrated in graphs below.



Therefore, one needs to trade-off between speed vs efficiency and choose the best suited learning rate.

Batch size:

Batch size determines the number of images that are parallelly processed. For example, consider a set of 90 images and a batch size of 30 images. This means that within a single epoch, 3 iterations each on 30 images from the set will be required to process the entire data in the set. The main reason for using batch size is that generally hardware required to run the network on all the images at once is not available or is too expensive. I observed that a larger batch size leads to a faster output, however a smaller batch size results in significant improvement in network accuracy.

Steps per epoch:

This is the number of steps with the given batch size to be performed in an epoch. For example, a set of 100 images, a batch size of 20 and 3 steps per epoch means that 20*3=60 images will be processed in one epoch. This should generally be equal to the number of unique samples of the dataset divided by the batch size. Therefore, sometimes smaller steps per epoch might be preferable. For example, consider the previous example. If steps per epoch are chosen to be 5, all images will be covered. However, if the images are repetitive, this will lead to overfitting and consequently, a poorer performance. In such a case, smaller steps per epoch such as 3 or 4 (depending on the data) might be advantageous.

Validation steps:

This is same as training set per epoch. The only difference is that this number is used for validation data set.

Workers:

These are the number of processes to spin up. Even after doubling the value from 2 to 4, the speed did not improve significantly. Therefore, I am using the recommended value of 2.

Number of filters in each layer:

The implemented model in function 'fcn_model()' has a parameter called filters. This parameter determines the number of filters in each encoding layer. Furthermore, the depth (number of filters) is double in each next encoding layer. For example, number of filters in first layer is 2*filters, in second layer it is 4*filters and so on. It was observed that when this parameter 'filters' was increased from 2 to 6, the network output accuracy increased. Further increment of 'filters', however decreased the network accuracy. Therefore, 'filters' parameter is chosen to be equal to 6.

<u>How did I choose these parameters:</u>

Initially, I experimented with the original data only, to observe how change in each parameter affected network performance and to find trade-off values. I started off with parameter values which would result in high speed, and eventually compromised speed for efficiency as required. I varied parameters as below. The values of each parameter are in the order I tried –

Network depth = 1 layer, 2 layers, 3 layers

No. of epochs = 10 to 25

Batch size = 512, 256, 128, 64, 32

Learning rate = 0.01, 0.005, 0.003, 0.002, 0.001

Steps per epoch = (No. of images)/(batch size)

For example, number of images were 4131. When batch size was 256, steps per epoch would be int(4131/256)=16.

Validation steps = steps per epoch

From these experiments, the best accuracy was observed with

Batch size = 32

Learning rate = 0.002

Number of epochs = 25

I observed that batch size and learning rate had to be small, and number of epochs to be large. In these experiments, the best final IOU score observed was less than 0.30. Another noteworthy observation was the high correlation between final score and training loss. Therefore, I focused on minimizing training loss.

Meanwhile, I generated my own data set using the guidelines given in classrooms. I generated a total of around 2300 images of following types –

1. Patrolling mode and follow me mode in dense crowd
2. Capturing all angles of target in patrol mode, where two patrol points are set and target walks in a zig zag way between the patrol points

However, this additional data did not improve the final score.

Next, I focused on finding the saturation point of the network where the network starts to overfit. The accuracy increased to more than 0.35 with the originally provided data. First, I experimented with the network model in function 'fcn_model()'. I found that with high number of epochs, 5 layer deep network performed better than 4 layer deep network. Adding one more convolution layer in the decoding layer, i.e., in function 'decoding_block()', improved the network efficiency slightly. Also, optimum value for 'filters' parameter was found to be six. A value more than six resulted in degraded performance.

Therefore, the final model that I used was –

Number of convolutional layers in each decoding layer	2
Number of layers	5
Filters (in function 'fcn_model()')	6

Below I'll give a table for some of iterations with original data of 4131 images and corresponding final IOU score –

Sr no	Learning Rate	Batch Size	No of epochs	Steps per epoch	Final IOU score
1	0.002	32	26	200	0.35
2	0.001	32	26	200	0.36
3	0.001	32	26	129	0.31
4	0.001	16	26	400	0.36
5	0.001	32	40	50	0.34
6	0.001	32	40	100	0.37
7	0.001	32	70	100	0.396
8	0.002	32	70	100	0.36
9	0.001	32	60	120	0.394

Observations from the iterations –

1. **Final score was better with lower 'steps per epoch' and more 'number of epochs'**
   It was observed that when 'steps per epoch' was close to or exceeded the value of (number of images)/(batch size), the network saturated earlier.

2. **Lower learning rate resulted in higher final score**
   When learning rate was increased to 0.002, the network saturated earlier and performed poorer than with learning rate of 0.001. For example, compare the 7th and 8th row. With parameters in 7th row, network reached a highest accuracy of 0.396, after which it had started to overfit. Whereas same network with learning rate of 0.002 (8th row) peaked at 60 epochs with a score of 0.377 (not shown in table) and the with 70 epochs, in fact, the network performance dropped down to 0.36.
3. **Too small batch size did not improve the final score**
   Reducing batch size from 32 to 16 while varying steps per epoch (150, 200 and 220) for 6th, 7th and 8th row in the table did not improve the final score.

The important things to observe were three scores calculated at the end of the notebook –

1. Scores while the quad is following the target
2. Scores for images while the quad is on patrol and the target is not visible
3. The score which tells how well a neural network can identify the target from far away

The first type score for target was around 90%. The score of second type was excellent as number of pixels overlapping target were zero, which was the desired score as there was no target in this case. The third type of score was below 14%.

To improve these scores, I collected new data. In the new data, I apart from previously discussed points, I focused on collecting target images from far away. I collected both training and validation data and added it to the original data. I used the same network model and same parameters *except* 'steps per epoch'. Please refer the table below for some important iterations –

Sr no	Learning Rate	Batch Size	No of epochs	Steps per epoch	Final IOU score
1	0.001	32	70	100	0.385
2	0.001	32	70	150	0.36
3	0.001	32	70	120	0.4118

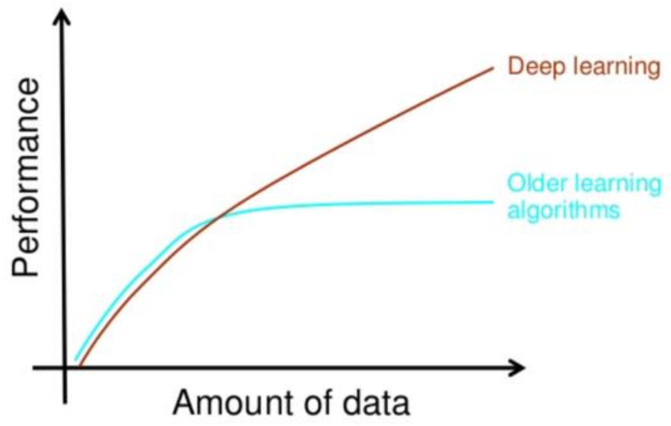
Upon addition of new data, the 'scores while the quad is following the target' increased to more than 92% and the score which tells how well a neural network can identify the target from far away increased to more than 18%. Consequently, the final IOU score reached to 41.18%.

Note: Although total number of epochs are 70, I have trained the network with epochs 50, 10 and 10. Therefore, in the notebook, it shows only the last ten epochs.

Future Enhancements:

The current neural network has a large scope of improvement. I will discuss some of the possible methods to improve the network performance.

1. **Get More Quality Data:**
   The network identifies object from far away with only 18% accuracy. Adding more data with target far away in it would help improve network performance because neural networks perform better with more data. Currently network is trained with about 7400 images which is a small number. Refer figure below

   Additionally, data collection should be specific. For example, as we are interested in accuracy of identifying target, more images with target in it should be collected. Additional data can be generated by random flipping and rotating images in current data set. This will add up to quality data, as every image in this data set will be different from image in original data set.

2. **Deeper Network:**
   With addition of more data as suggested in the previous point, network should be made deeper. This will allow the network to extract more features resulting in better performance.

3. **Different Model parameters:**
   Trying different activation functions like sigmoid, optimizers like RMSprop or regularization methods like dropout might further improve network performance.

Limitations of Neural Network

Neural networks are very data specific. Therefore, the same model with same parameters will not work for a new target, for example, to identify a target such as car, dog, cat etc. This is explained below.

1. **Same model and parameters cannot be used:**
   The current model classifies an image into three objects – target human being, other human beings (crowd) and surrounding. The target being a human itself, it takes longer for network to learn to identify the target and crowd correctly. In case the target was car in all training images, then with given model and parameters, the network might overfit. This is because the shape of car is much different from the crowd and the surrounding. Hence, in fact smaller number of epochs and less deep network might work better. The same logic applies in case of target object as dog or cat also.
2. **Trained weights cannot be used:**

As any of car, cat or dog are not anywhere close in feature to human target used in this project, the trained weights cannot be used. Therefore, the entire training process will have to be repeated for new objects. However, if the new target to be identified is very similar to the target person in this project, then it might be possible to use trained weights obtained in this project. Thus, only a few iterations for fine tuning will be required. For example, if the new target is same person with slightly different color of clothes, then previously trained weights can be fine-tuned for the new target.

Reference links:

http://iamaaditya.github.io/2016/03/one-by-one-convolution/

https://machinelearningmastery.com/improve-deep-learning-performance/

https://arxiv.org/pdf/1511.00561.pdf