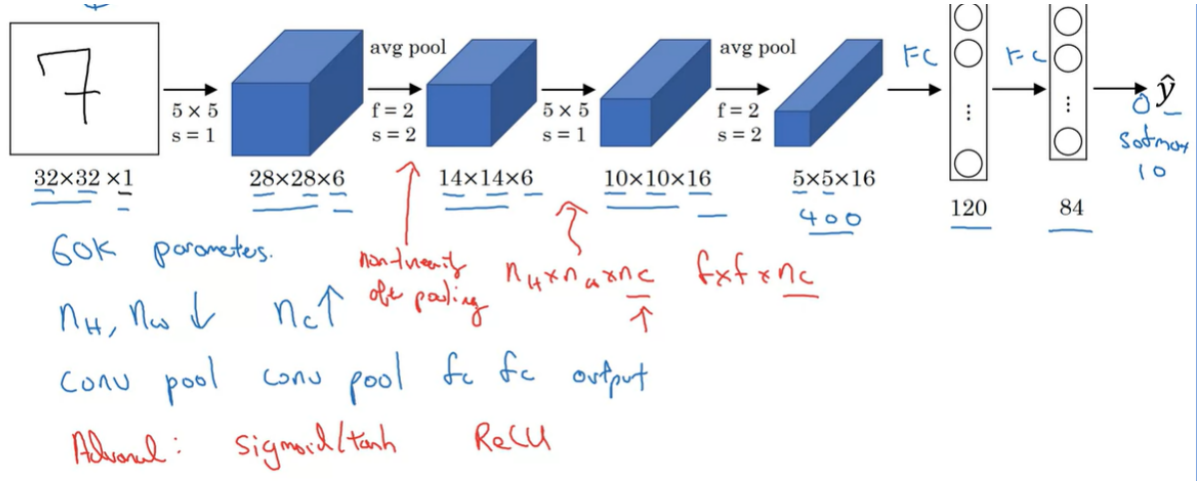
dl.ai_CASE STUDIES

Explore more about the difference between versions

- AlexNet achieved a top-5 error rate of around 17% on the ImageNet dataset
- Inception-v3 achieved top-5 error rates of around 3.46% on the ImageNet dataset
- ResNet-50 achieved a top-5 error rate of around 7.8% on the ImageNet dataset.
- ResNet-152, have demonstrated even better accuracy, achieving top-5 error rates below 5% as of that time.
- VGG-16 achieved a top-5 error rate of around 7% on the ImageNet dataset, while VGG-19 also achieved similarly impressive accuracy on this dataset.
- MobileNetV1 achieved around 10.5% top-5 error rate on ImageNet being highly optimized for deployment on mobile and edge devices.
- MobileNetV2 attained approximately around 9% top-5 error rate.

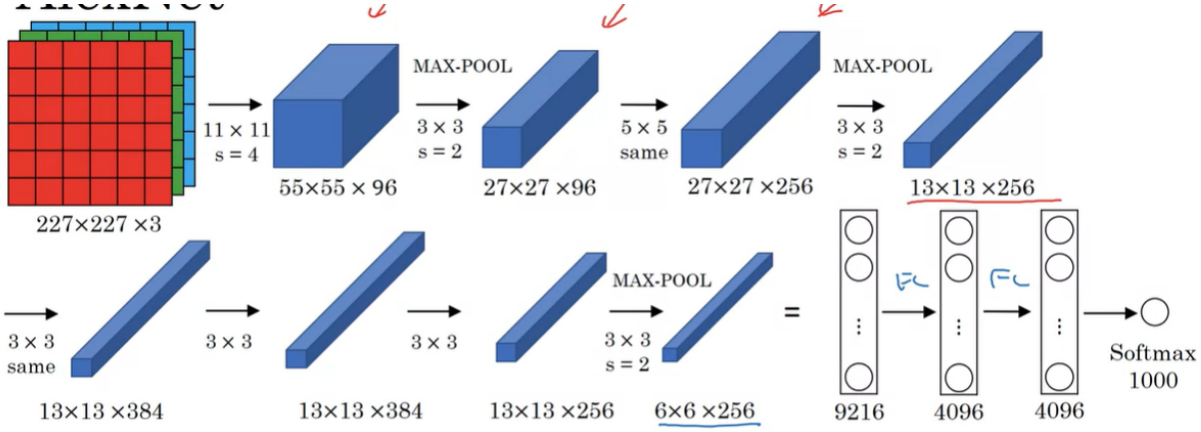
a)Classic Networks:

LeNet-5



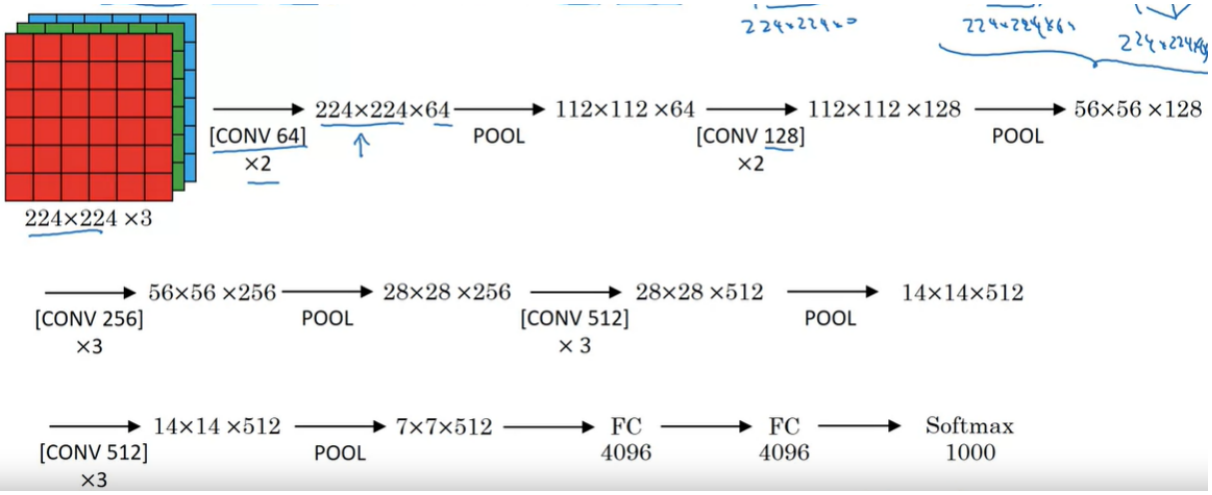
LeNet-5 is a smaller model with only 60k parameters for mnist dataset as no of chanels in input layer is1.

AlexNet



- It is similar to lenet but larger,with 60 million parameters specifically for imagenet(1000)
- It had additional Local Response Nuetralization Layers which take all values having same position across multiple channels and normalize them.(it was useless)

VGG-16

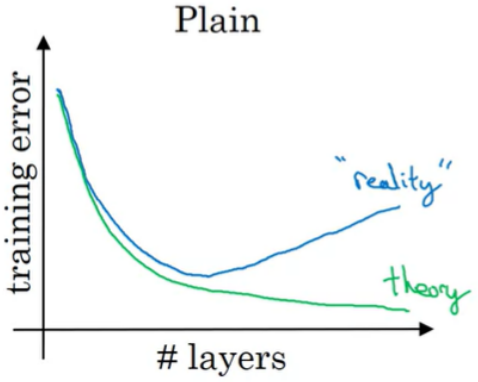


- Vgg-16 is very deep model with 138 million parameters specifically for imagenet(1000).
- It has 16 slayers but is simple as it always has s=1,same paddimg convolution with 3*3 filters,iit uses a 2*2 s=2 max pool after every 2 or 3 convolution layers

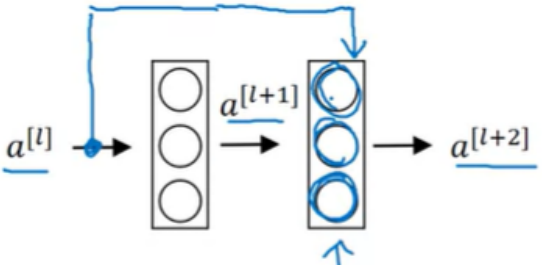
b)Resnet

why do we need resnet?

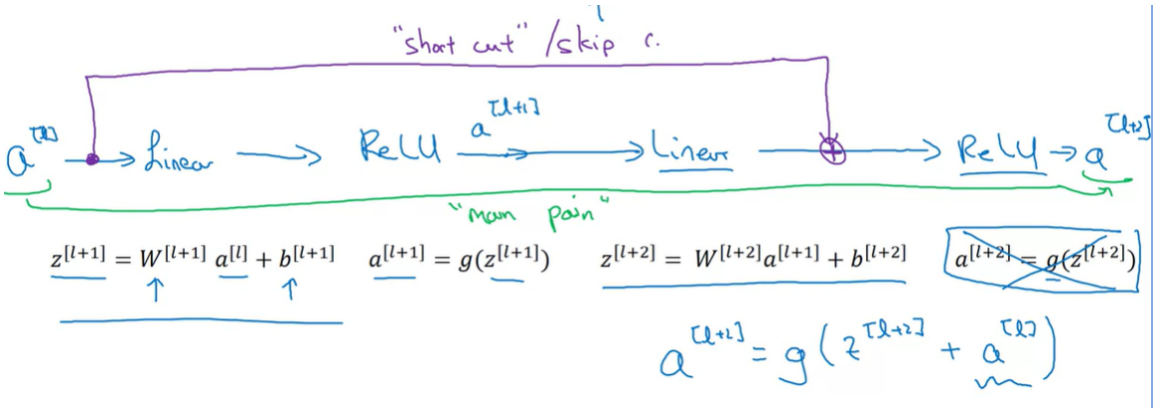
- In theory we except training error to decrease when we keep on adding layers to the model.However in practice we see that the training error actually starts increasing after a point.



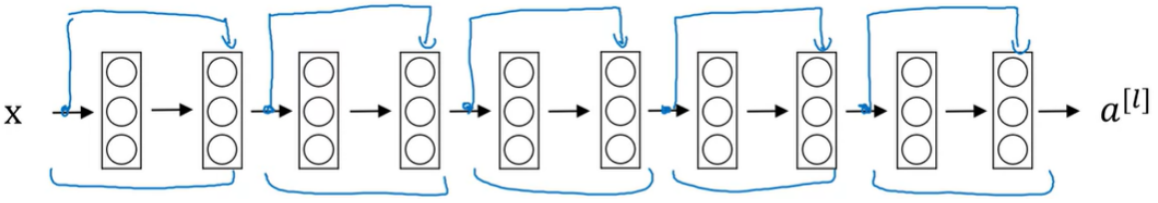
- This may be happening because as we add more layers optimizing all the parameters becomes harder(no overfiting crap).Resnets simply ensures that as we add more layers our training error never increases atleast.



- This is called as one residual block.Where we basically have one shortcut which allows us to bypass the main route.

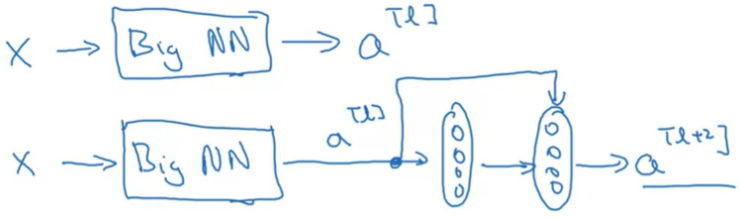


- This Network is made up of 5 residual blocks.Note that we dont usally have two overlaping shortcuts.



why does resnet work?

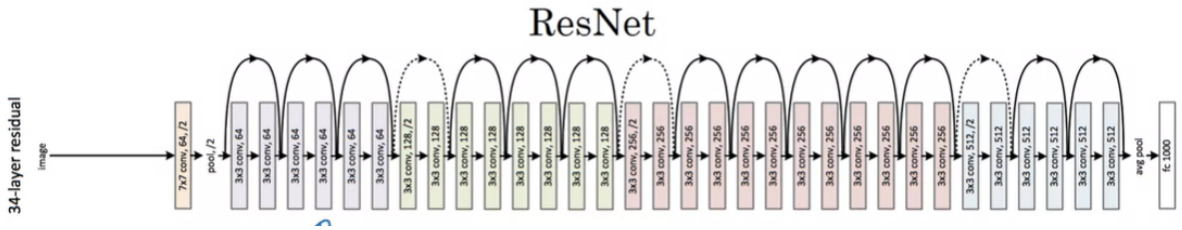
- Let say we have a big nueral network and we want to add a residual block to it



- How can we ensure that this two new layers (residual block) won't hurt the performance of the model.

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$
$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

- we can see that if we put $w = 0$ and $b = 0$ we get $a^{[l+1]} = g(a^{[l]})$ and since g is relu we get $a^{[l+1]} = a^{[l]}$ the identity function.
- residual block just make it easier for the model to learn identity function which is otherwise harder with such a big model.
- If the new layers don't add anything useful we can use the identity function to retain our previous performance,If they do we can consider them.
- $a^{[l+1]}, a^{[l]}$ need to be of the same dimension hence resnet is full of same convolutions.

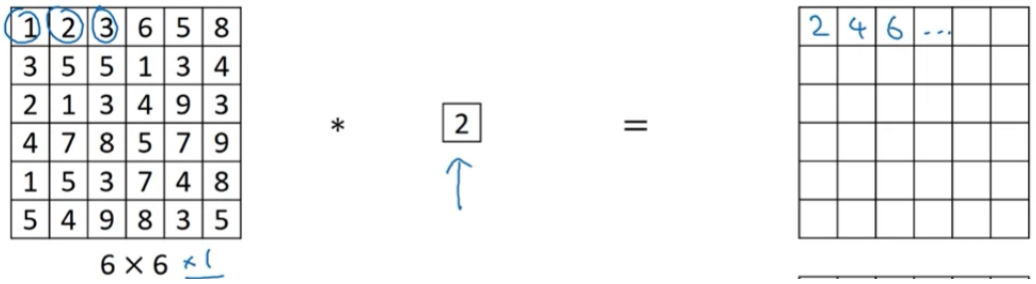


- the dotted line is show residual blocks having pooling layers,$a^{[l]}$ dimension must be adjusted.
- I feel like whenever you introduce a shortcut you are giving the model new paths to choose from. The model could just take the first shortcut or the last shortcut or all of the shortcuts.
- It is also not just about taking a path resnet allows us to give weightages to paths.

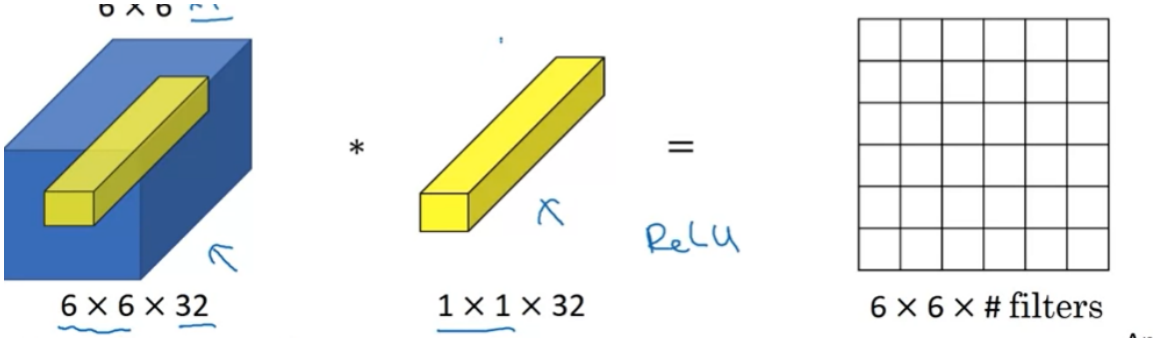
c) INCEPTION NETWORK

1x1 convolution

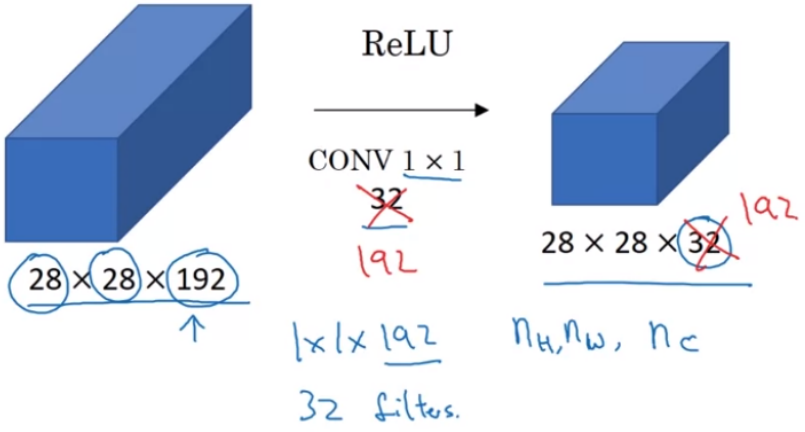
- In 2d when we do convolution with a 1*1 matrix we simply get a scaled version of the original matrix with the same dimension.



- When we do the same operation in a input matrix with multiple channels, we will be able to do operations on an single slice of the input matrix across the channels.



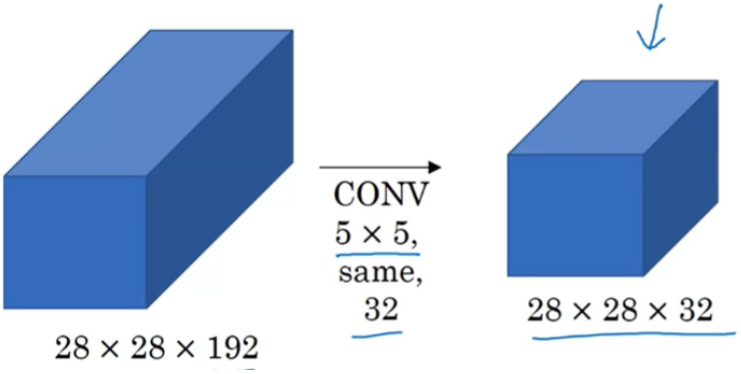
- we take all 32 elements in our input from a given position multiply them with 32 elements in the filter and add them up to get a single number which we store at the same place in the output matrix.
- If we use multiple filters we basically get multiple channels. we can use this to decrease the no of channels without affecting the other two dimensions.



- **Value at each position will only depend on the values at that same location in previous layer, there wont be any regional learning.(learn edges and stuff).**

Bottle-neck layer

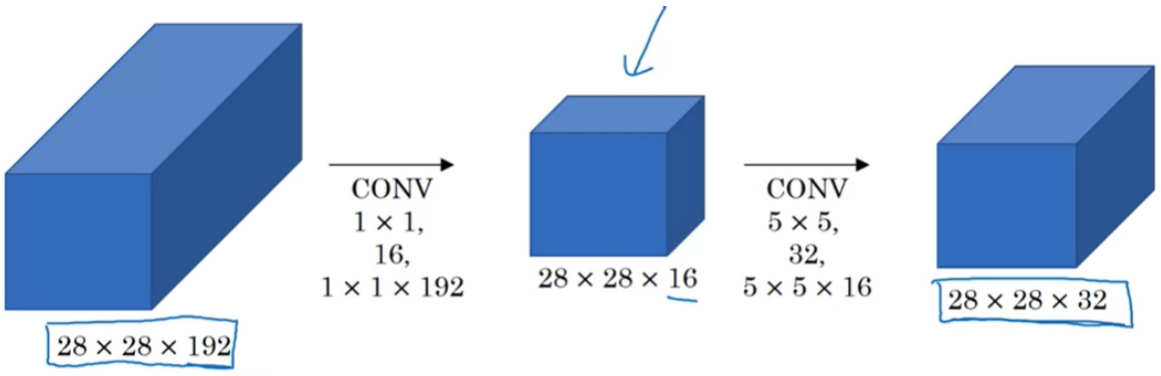
- Convolution layer with 5×5 filters take a lot of computational operations.



- In this layer each time we multiply 5×5 elements in the kernel with the input matrix, To complete the convolution in one layer we need to multiply across 28*28 spots in each layer, for each filter we repeat this process over 192 channels, and we have 32 filters.

$$multiplications = 5 * 5 * 192 * 28 * 28 * 32$$

- no of additions is roughly same, we just say computational cost of above thing 120million
- we can use 1×1 convolution to first decrease the layers and then use the 5×5 convolution on a less wider matrix.

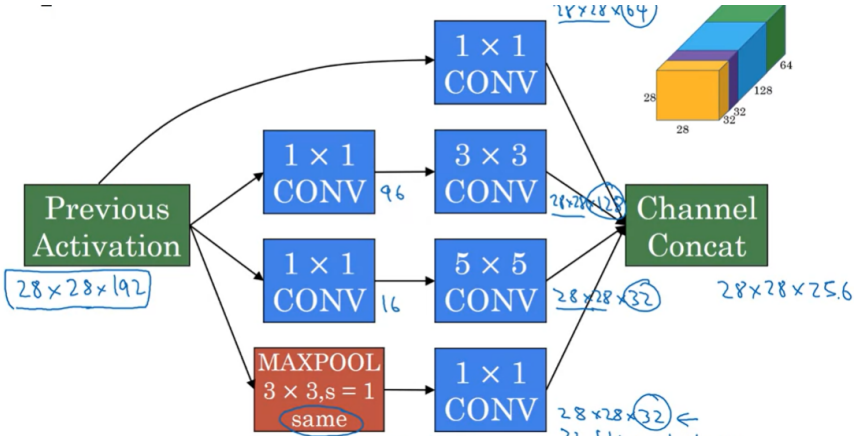


$$computational\ cost = 1 * 1 * 192 * 28 * 28 * 16 + 5 * 5 * 16 * 28 * 28 * 32$$

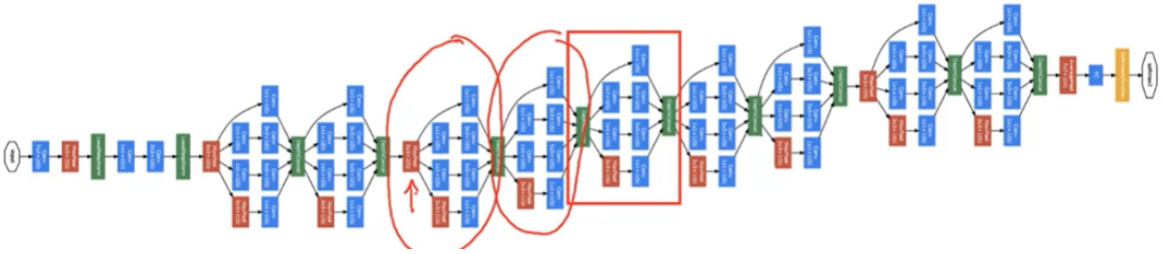
- We get approx 12.4 million parameters about a tenth of the original.
- Simply shrinking the model to 28*28*32 will affect the performance of the model, value at each position will only depend on the values at that same

location in previous layer, there wont be any regional learning.(learn edges and stuff)

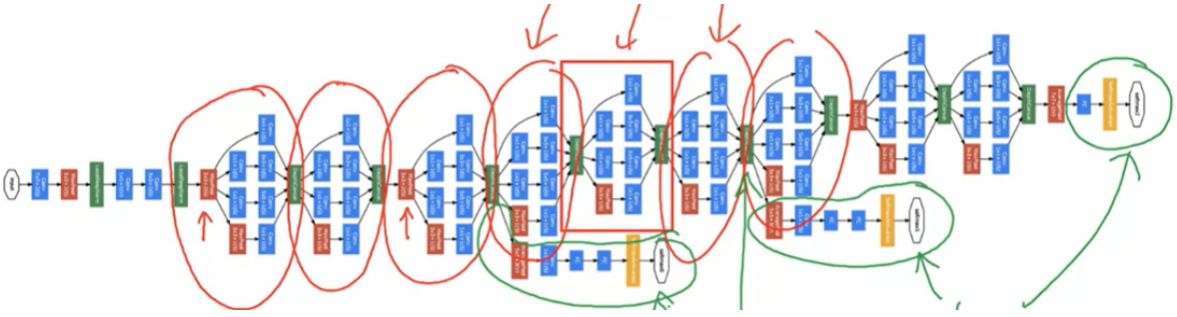
Inception block



- The max pooling is different we use a3×3 filter with same convolution
- The basic idea is that instead of you needing to pick one of these filter sizes or pooling you want and committing to that, you can do them all and just concatenate all the outputs, and let the network learn whatever parameters it wants to use, whatever the combinations of these filter sizes it wants.



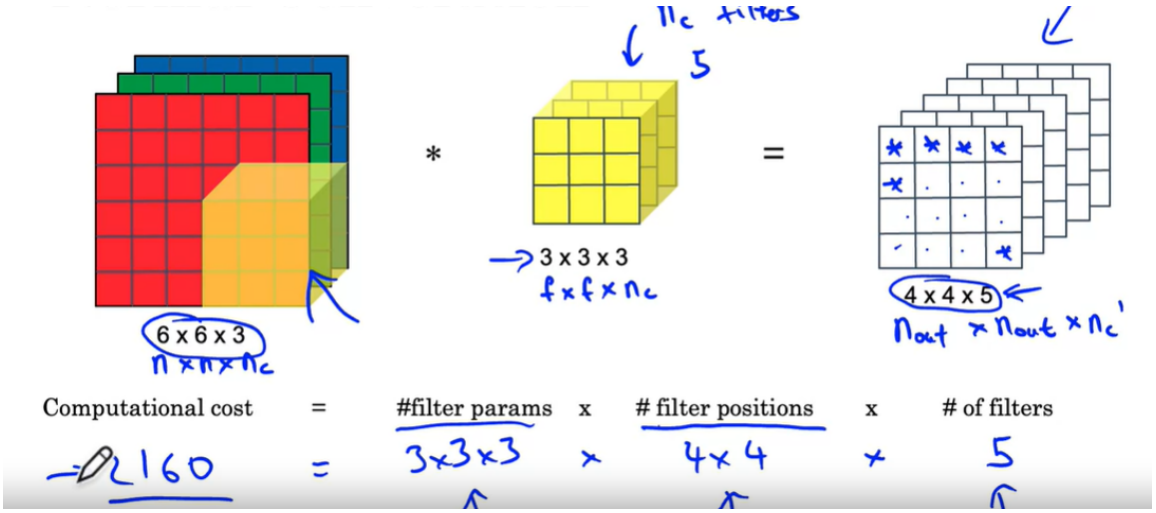
- It is simply a series of inception block after apoint.



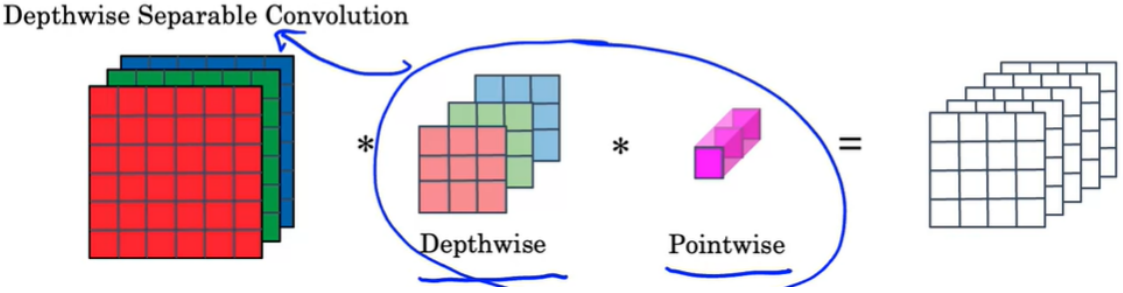
- we add sub outlets to prevent overfiting.

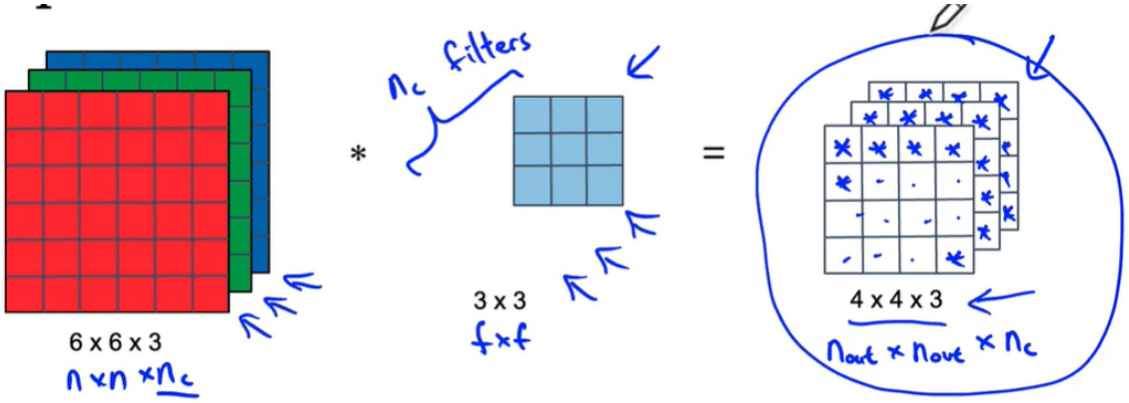
d) MobileNet

depth wise seperable convolution



- In bottle neck we were looking at a way to use a normal convolution over multiple filters with an intermediate matrix, to decrease computational cost. Here we would be looking at changing the convolution itself
- We first do a depthwise convolution (each channels seperately) and then do point wise convolution(1×1) to get the same dimensional output.

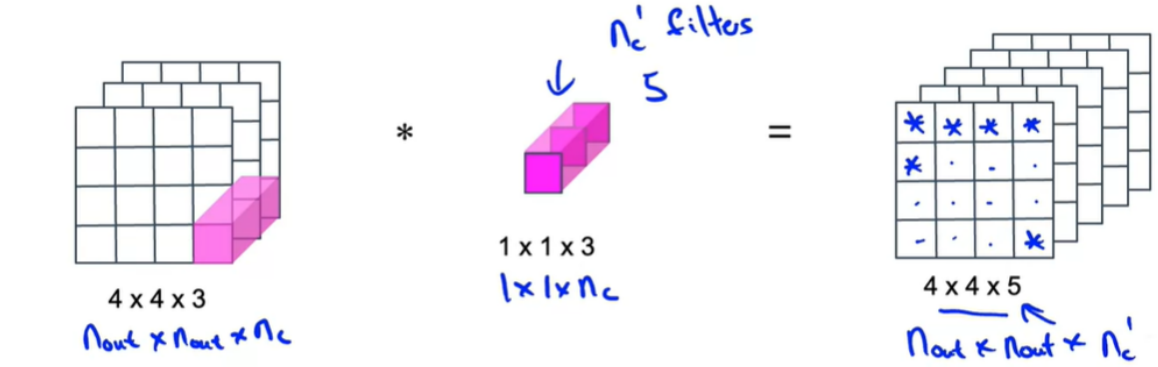




- We are taking 3, 2d filters and convolving it. In each multiplication step we multiply 3∗3 elements of the input channel with corresponding filter channel, the filter moves across 4∗4 positions, we do this process three times.

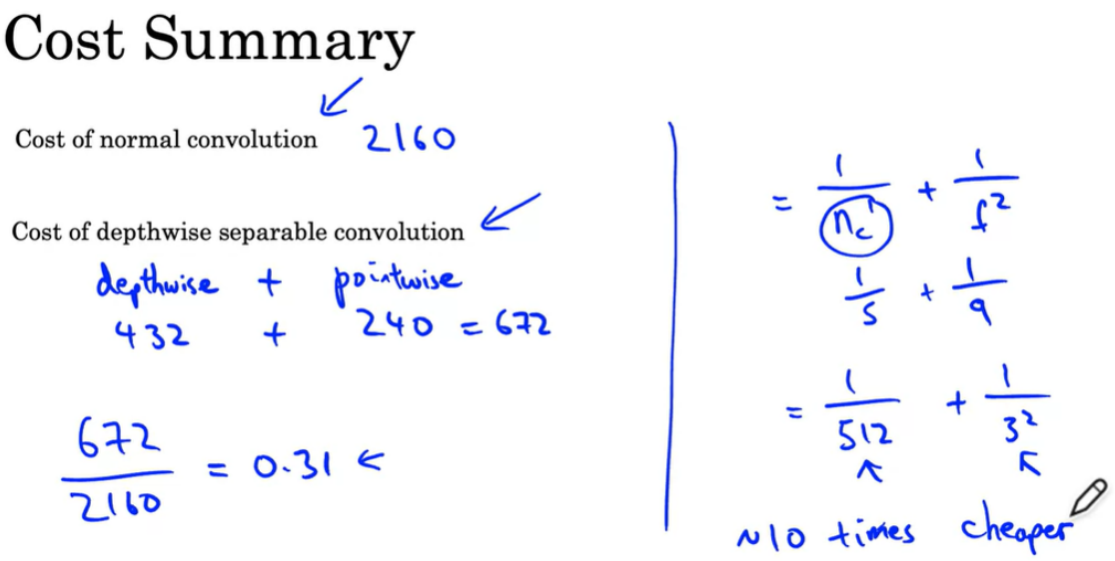
Computational cost	=	#filter params	x	# filter positions	x	# of filters
432	=	3×3	×	4×4	×	3

- **If we instead convolute with a 3∗3∗3 3d filter we would have got a 4∗4∗1 matrix as output**



- We have 5:1×1×3 3d filters,in each multiplication step we multiply 1∗1∗3 elements ,and we do this step over 4∗4 positions,we repeat the convolution for 5 filters

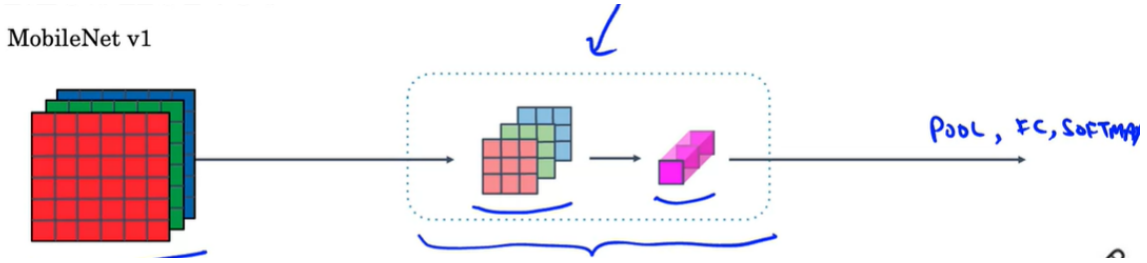
Computational cost	=	#filter params	x	# filter positions	x	# of filters
240	=	1×1×3	×	4×4	×	5



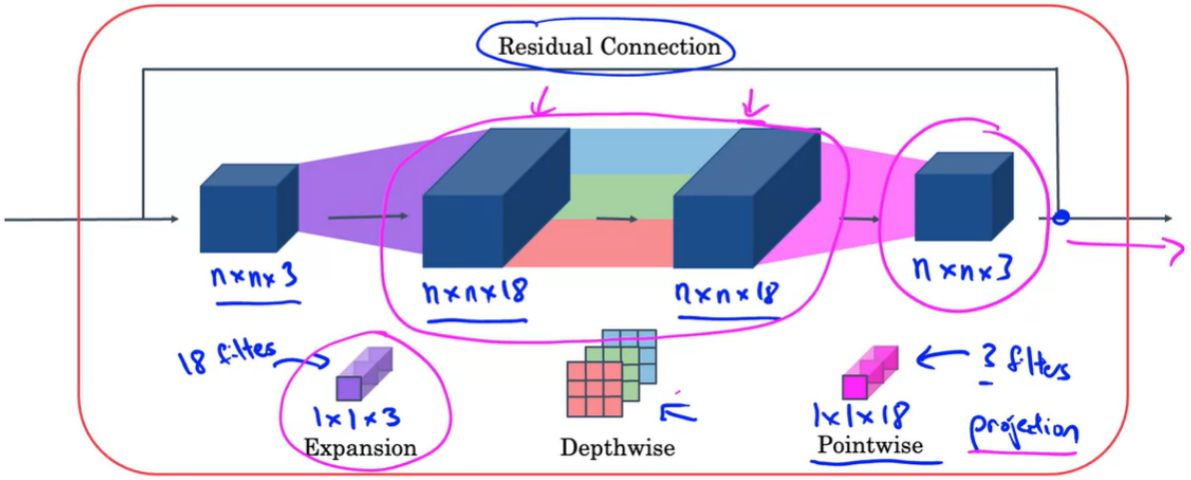
- The decrease in computational cost is due to a decrease in parameters from 3×3×3×5 to 3×3×3 +3×5.Hence a slight reduction in training error can be seen.

The architecture

- The original MobileNet v1 had 13 depthwise seperable convolution along with pool.fc nad softmax layers



- In MobileNet v2 we use 17 residual blocks over bottleneck depth seperable convolutions



- In bottleneck we use an expansion filter to increase the number of channels, then we apply depthwise convolution and then a projection filter to reduce its size again.
- The computational cost will be more than a regular depthwise seperable convolution but still lesser than normal convolution.
- By expanding it we can get a more richer representation from which we could get richer features.
- Even though we have larger matrices inside the block the size of activation functions(memory) which we need to pass from layer to layer is still low.
- That's why the MobileNet v2 can get a better performance than MobileNet v1, while still continuing to use only a modest amount of compute and memory resources.