Home work: make sure the activation and #parameters are correct?

Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	– 3,072 $a^{[0]}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Note: Trends in CNN are as follows:

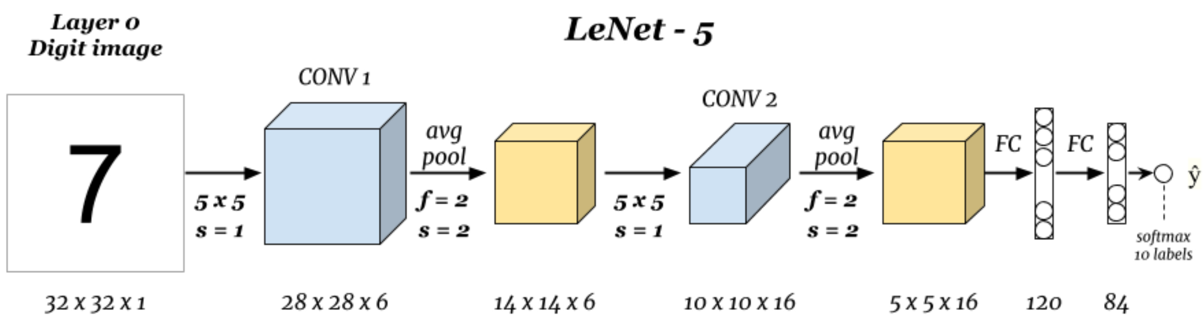
i. POOL layer requires no learnable parameters
ii. CONV layer requires fewer number of parameters compare FC layers
iii. Activation size decreases gradually with depth of CNN
iv. Number of channels increases gradually with depth of CNN

Well Known Architectures: Classic Network

- If a network is good for cat detection, I may work for other problems, like a car detection.
- To see the general trend for hyperparameters

1. **LeNet – 5 (for digit recognition)**

One example of classic networks is LeNet-5, from *Gradient-Based Learning Applied to Document Recognition* paper by Y. Lecun, L. Bottou, Y. Bengio and P. Haffner (1998):



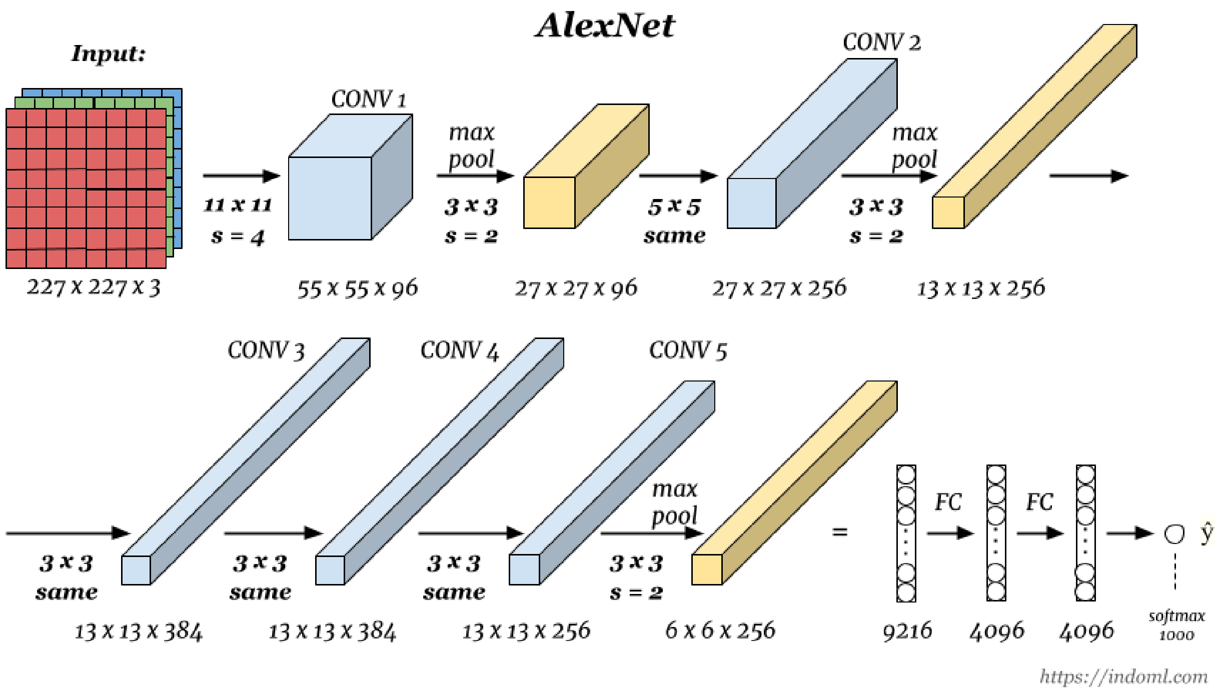
- Number of parameters: ~ 60 thousands.

General trend for hyperparameters:

- Size (height and width) of feature maps are decreased with depth
- While the number of channels are increased with depth
- General pattern of layers are CONV- POOL- CONV- POOL – FC – logistic/softmax

2. AlexNet (8 layer architecture)

AlexNet is another classic CNN architecture from *[ImageNet Classification with Deep Convolutional Neural Networks]* paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever **(2012)**. [8 layers architecture]



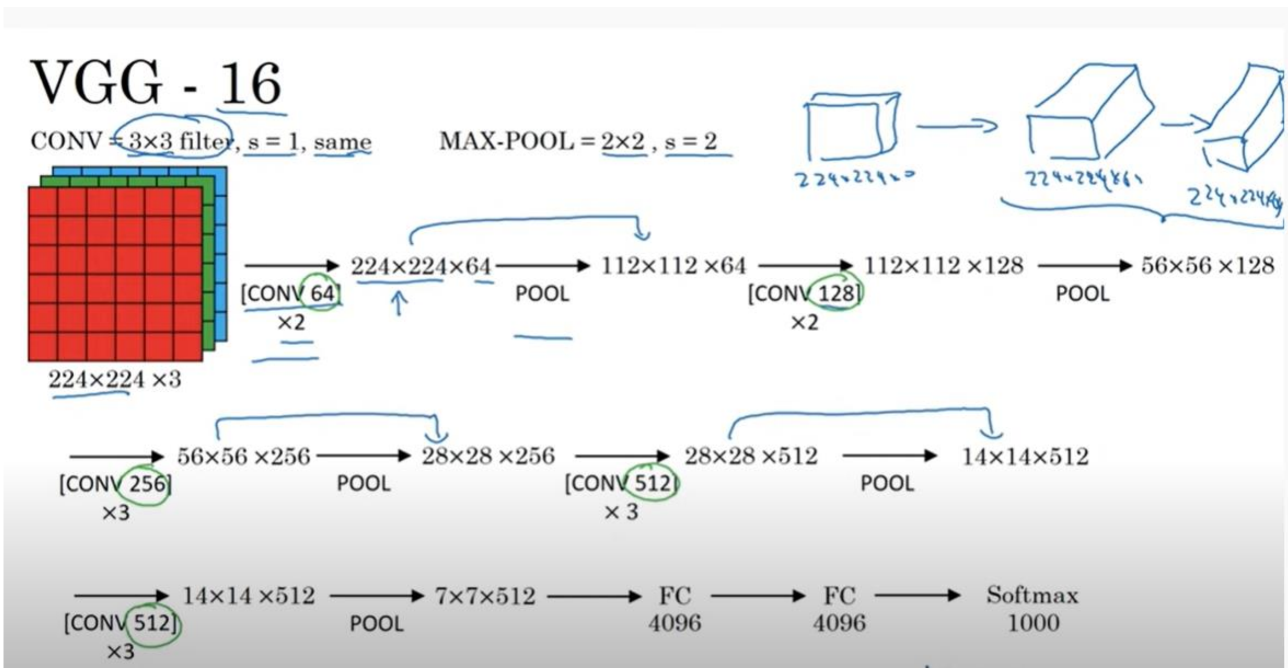
- CONV-POOL-FC pattern
- Size of feature maps decrease, while no of feature maps increase with increase in depth
- Number of parameters: ~ 60 millions.

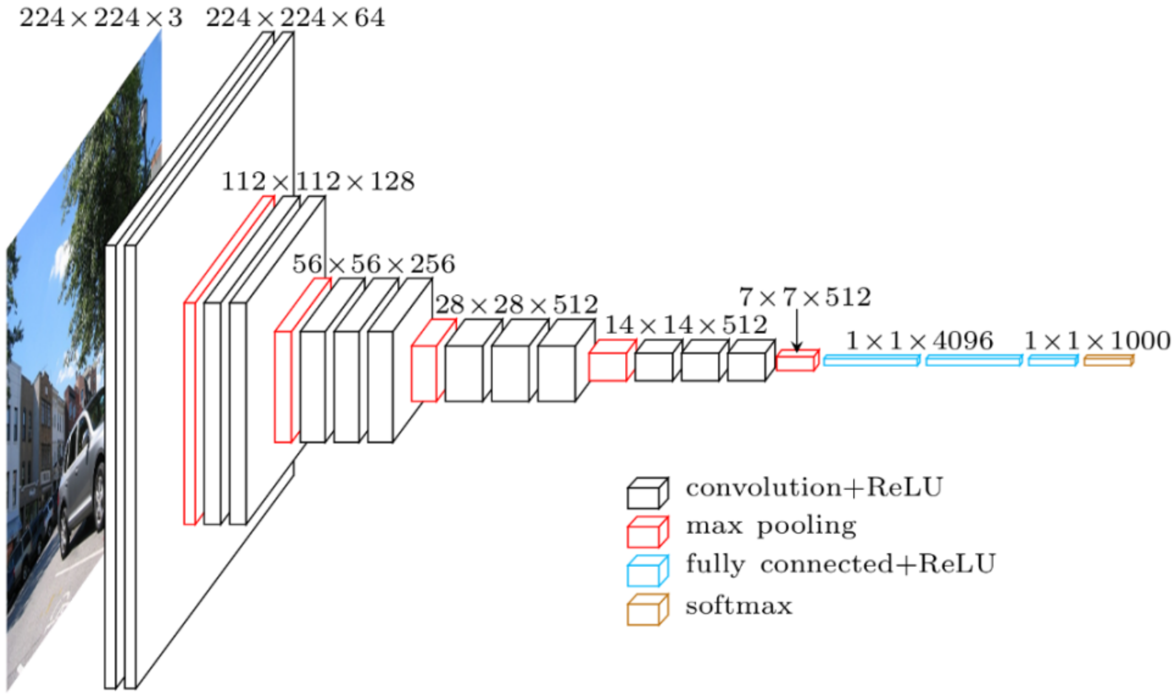
3. Classic Network: VGG-16

VGG-16 from *Very Deep Convolutional Networks for Large-Scale Image Recognition* paper by Karen Simonyan and Andrew Zisserman **(2014)**.

The number 16 refers to the fact that the network has 16 trainable layers (i.e. layers that have weights).

- The strength is in the simplicity: the dimension is halved and the depth is increased(double) on every step (or stack of layers)



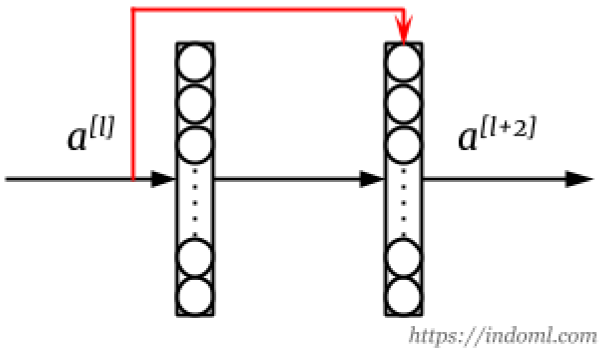


- Number of parameters: ~ 138 millions.

4. **ResNet (Residual Network)**

- The problem with deeper neural networks are they are harder to train and once the number of layers reach certain number, **the training error starts to raise again**. Deep networks are also harder to train due to **exploding and vanishing gradients problem (Soln we have discussed => xavier initialization and batch normalization)**.
- [He et al., 215. *Deep Residual Learning for Image Recognition*]solves these problems by implementing skip connection where output from one layer is fed to layer deeper in the network:

- Make a residual block:

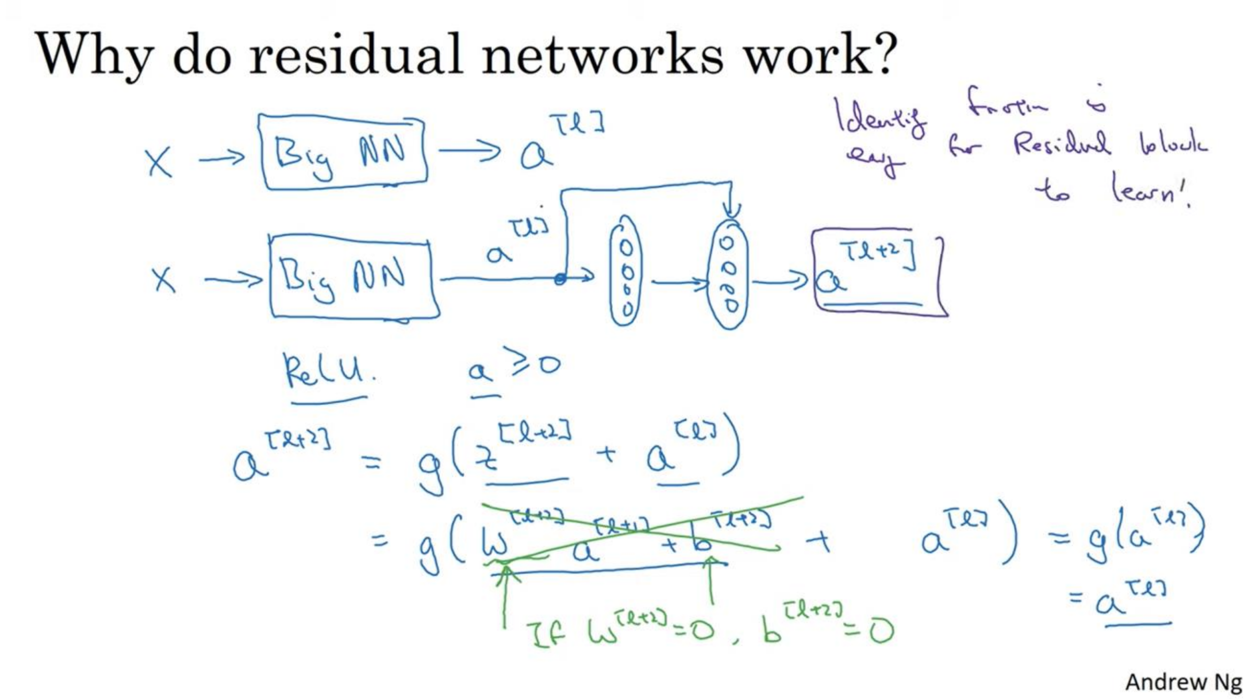


In the image above, the skip connection is depicted by the red line. The activation $a^{[l+2]}$ is then calculated as:

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

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

- The **advantages** of residual blocks in ResNets are:
  - performance doesn't degrade with very deep network
  - ability to train a very deep network due to the idea of residual block



Considering two assumptions:

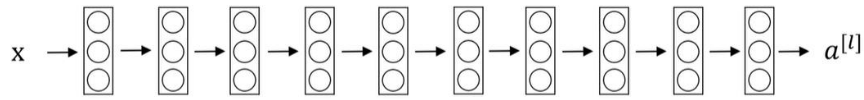
1) Use a ReLU activation as a = max (z,0), if a >= 0, then a = z
2) Use L2 regularization, assume it is apply on both w and b nearly equal to zeros.

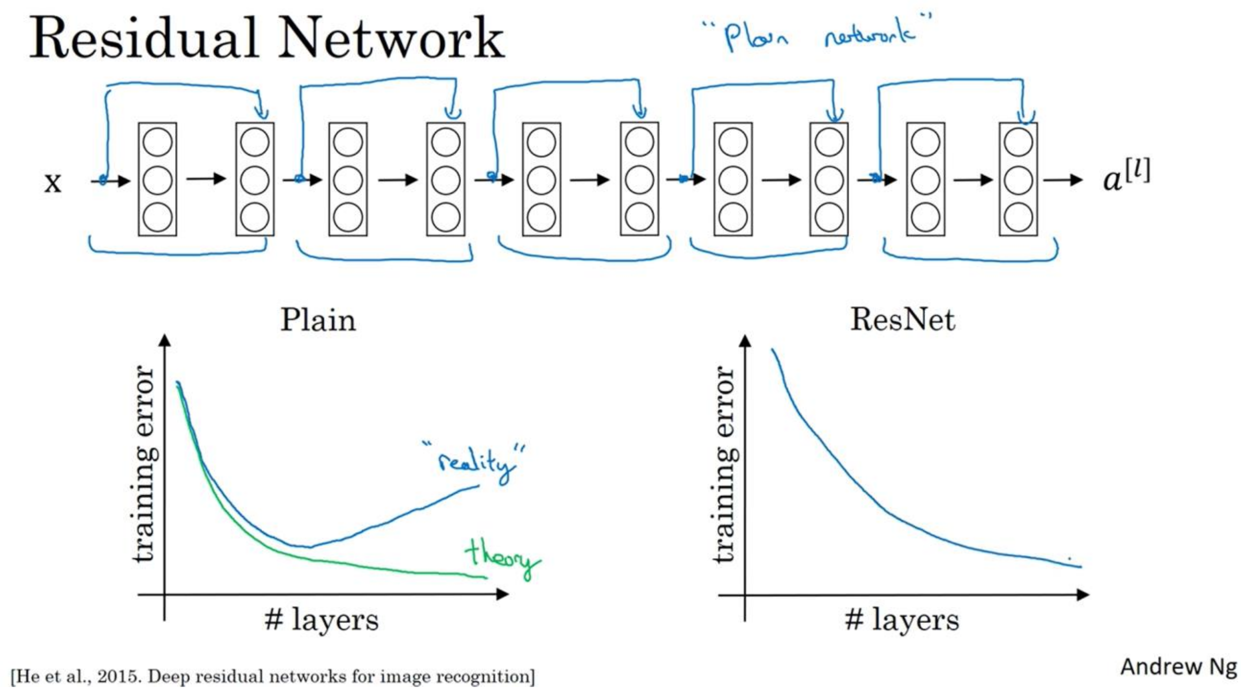
ResNet works because:

- Identify function is easy for residual block to learn
- Residual blocks **guarantee to not hurt the performance of network** and most of the time we are lucky to get good performance.
- So adding extra layers makes a deeper network(which learn more complex nonlinear function) without hurting its performance.

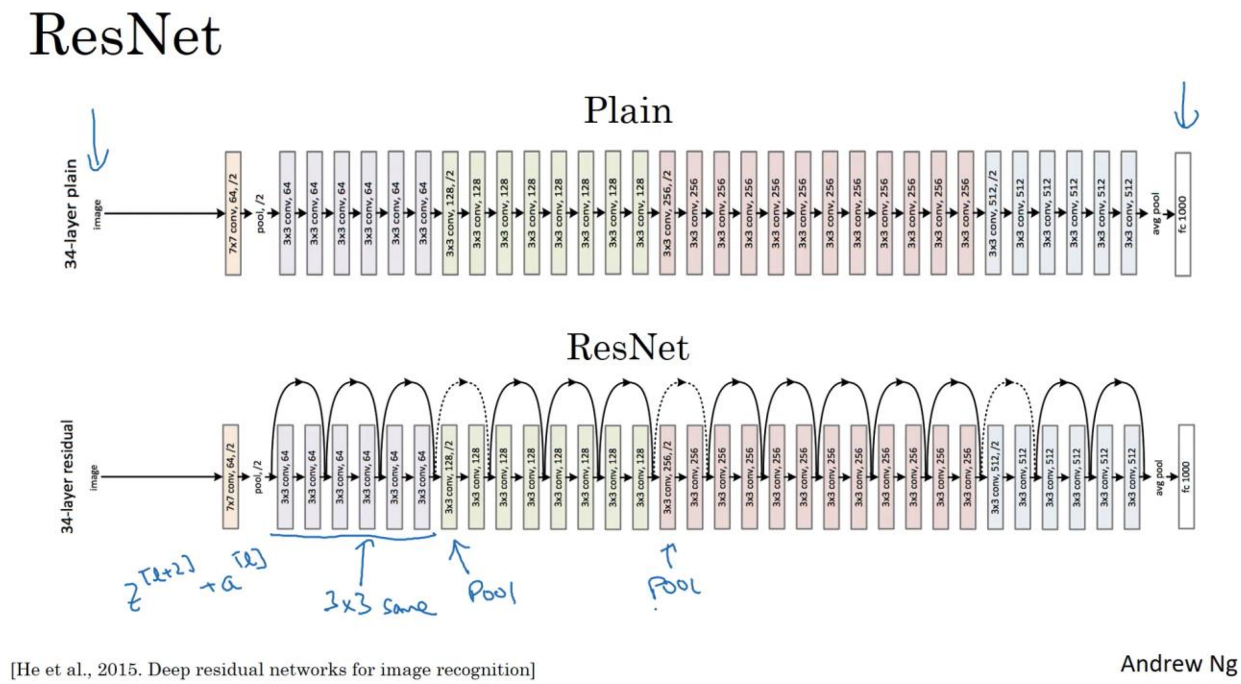
Note: dimensions of $a^{[l]}$ and $a^{[l+1]}$ must match.

Turn a Plain Network into a ResNet





Training loss is decreased with increase in number of layers in Resnet but in plain network it increases.

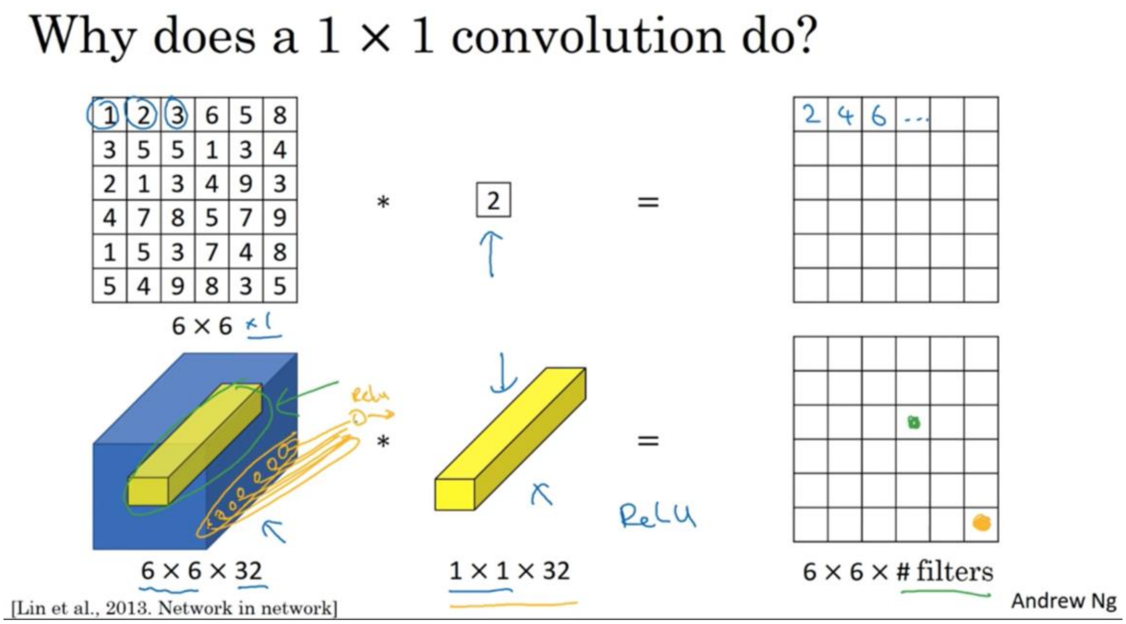


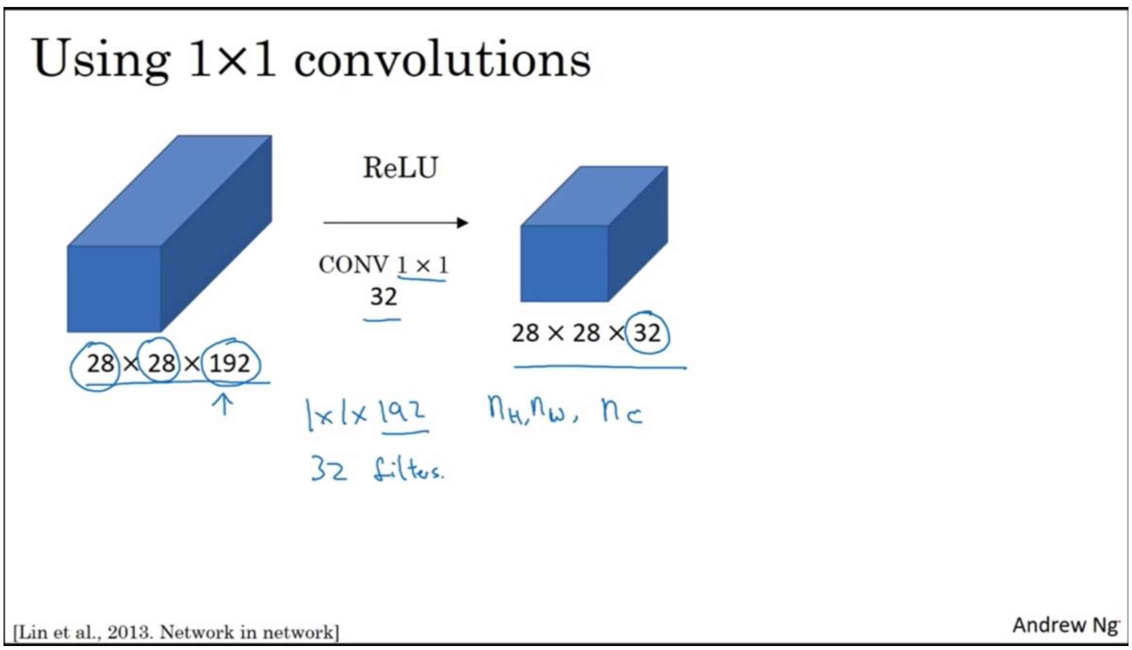
- Same convolution means add padding to maintain the output size equal to input
- Pattern :Conv – Pool --- FC – SoftMax

USE IN PYTORCH:

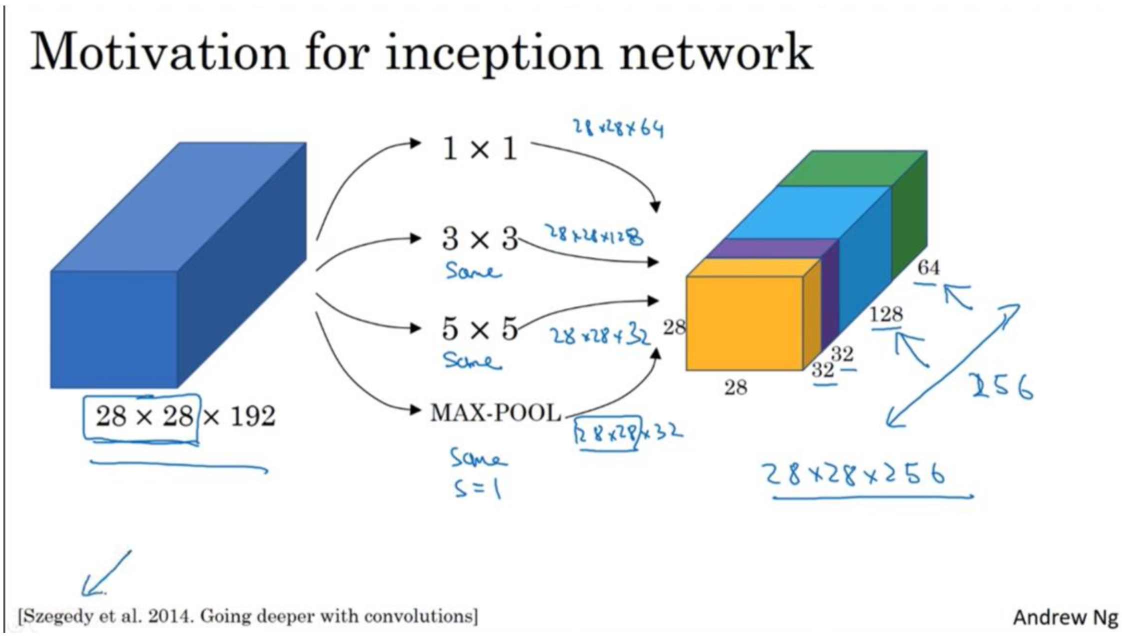
```
import torchvision.models
resnetmodels = models.resnet34(pretrained = True)
```

5. Inception network:

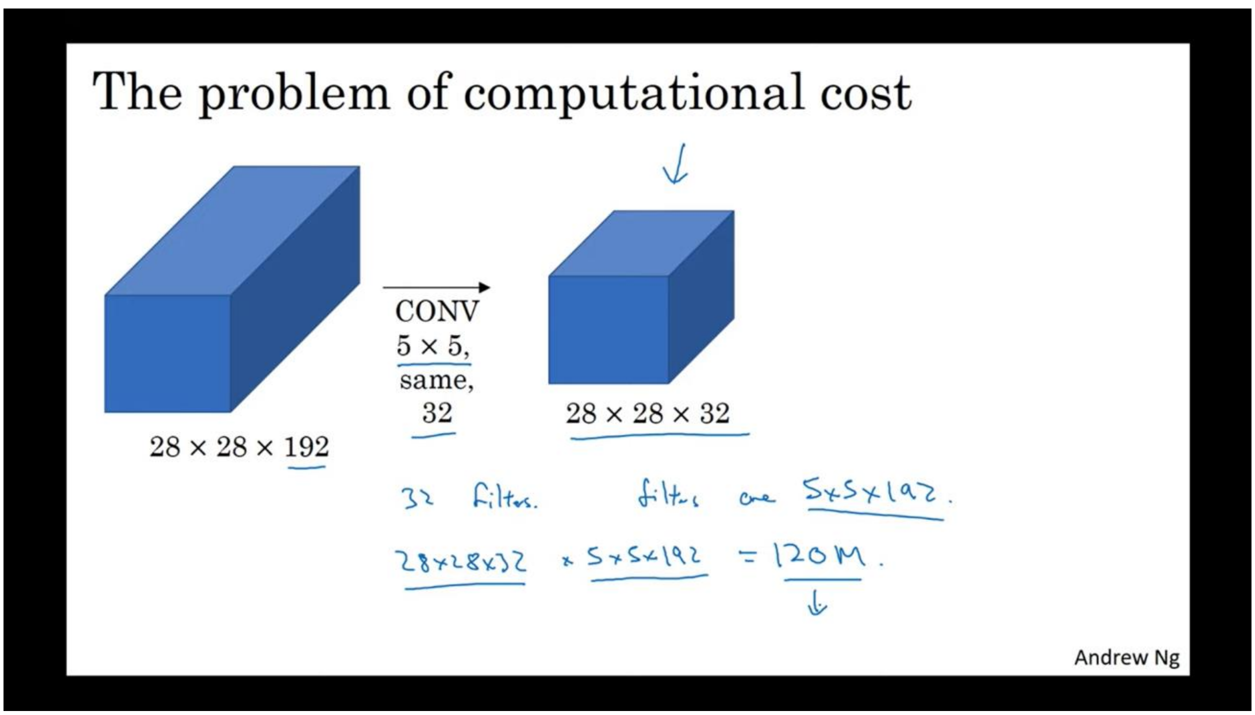




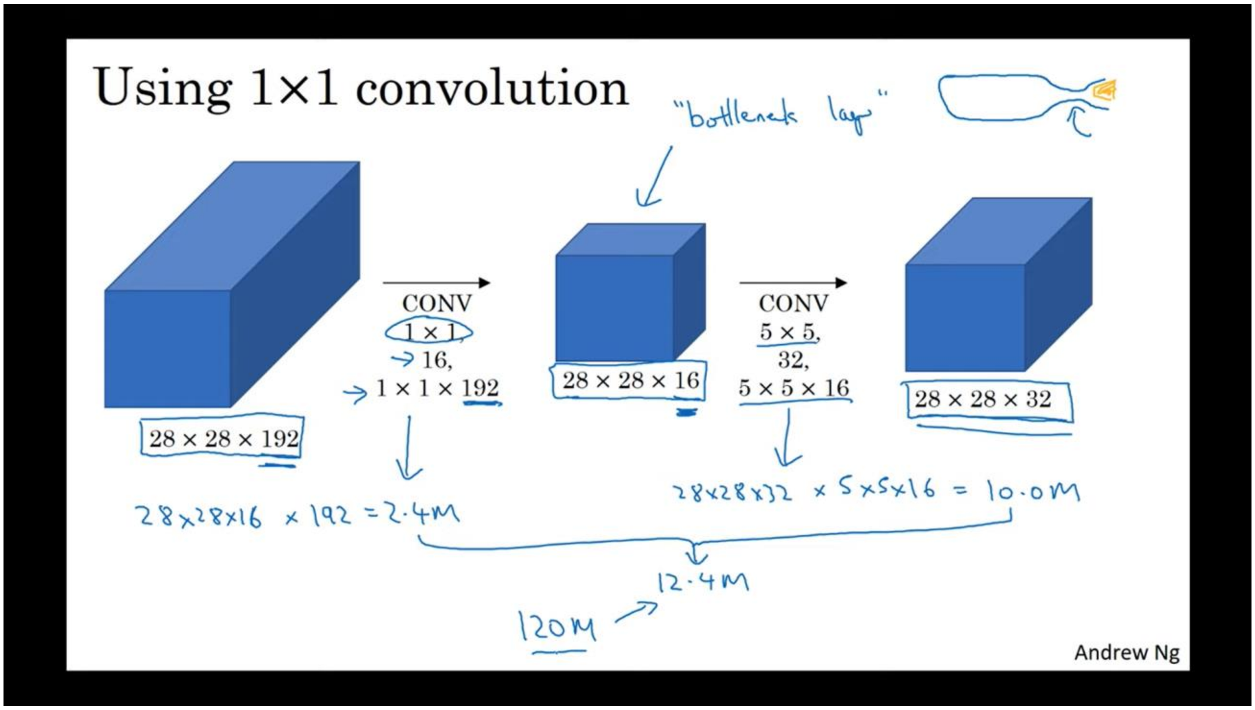
- Conv and Pool operation reduce the height and width of outputs whereas 1x1 conv helps to shrink the number of channels (reduce computations)
- It is helpful in building inception network to reduce computation cast by shirinking the number of channels



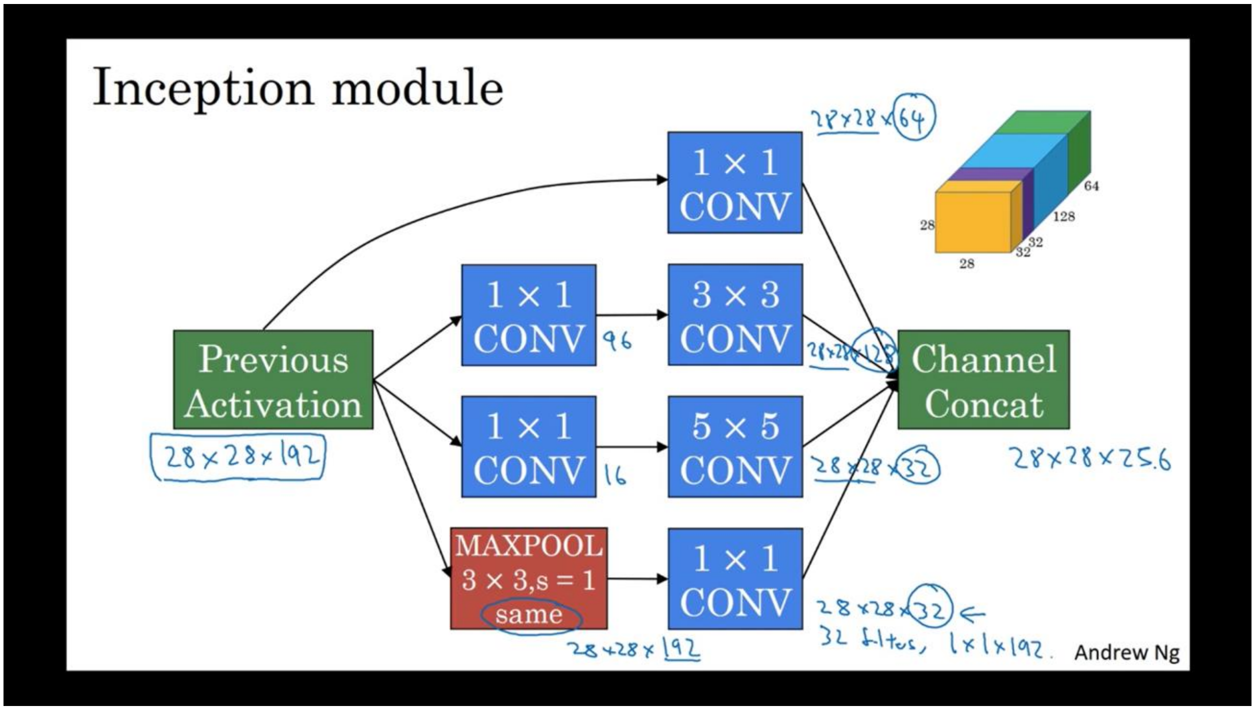
Problem in Inception network: computational cast.

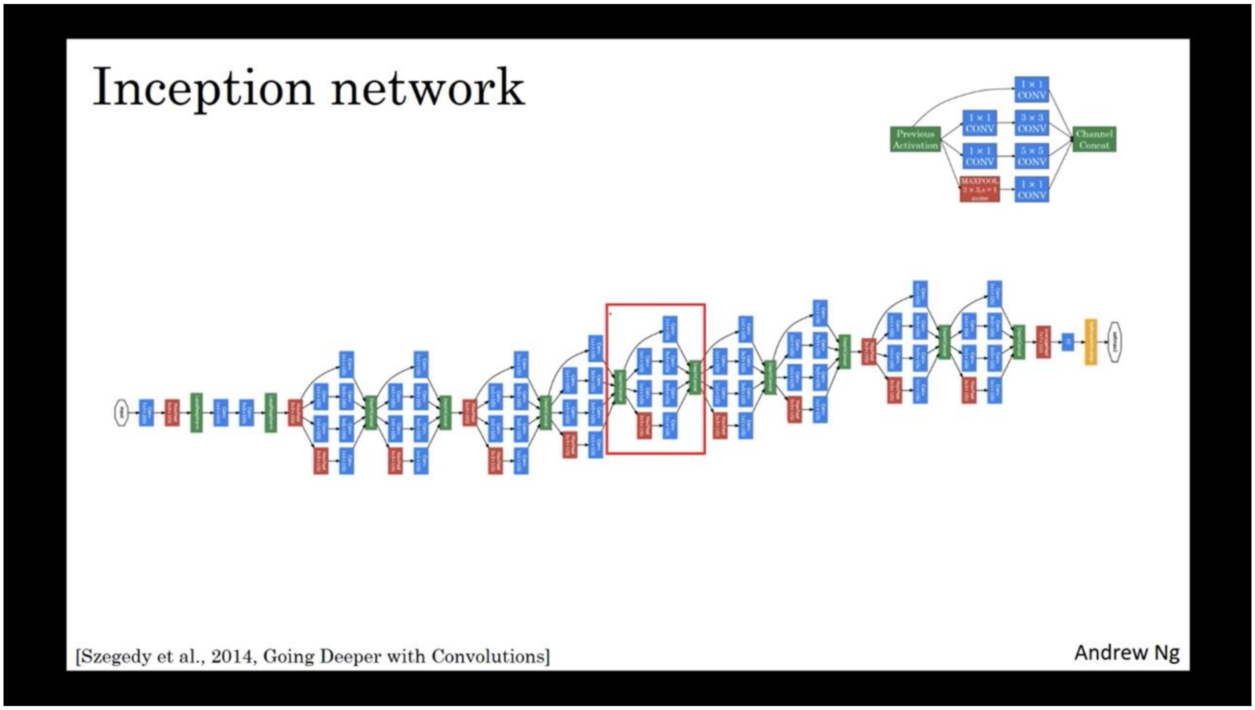


Solution is 1x1 convolution operation

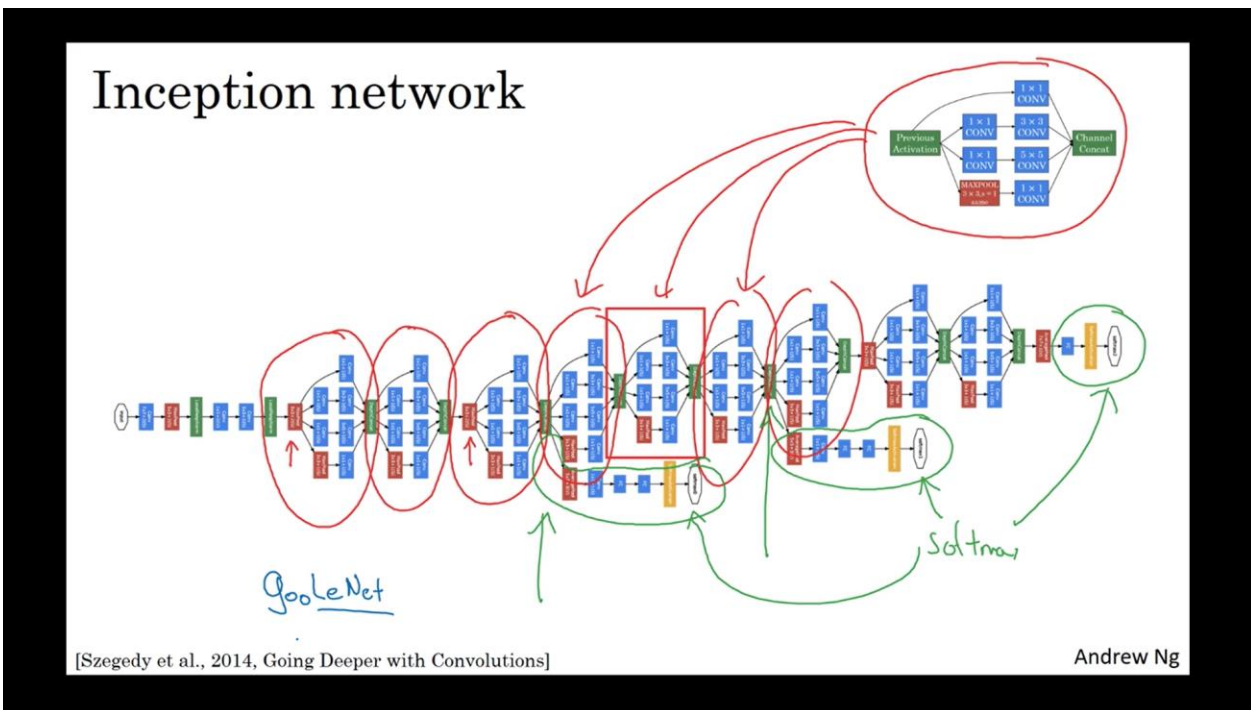


- Number of multiplications reduce 10 times.
- It is observed in Inception network by shrinking the number of channels do not effect the performance of network





- Lot of inception module with maxPool layers
- MaxPool layers to reduce output size



- Side branches at hidden layers:
  - Output is still good as they have the regularization effect.

USE OF OPENSORUCE IMPLEMENATION.

Assignment # 3

Revise Assignment #2 by integrating the following two components:

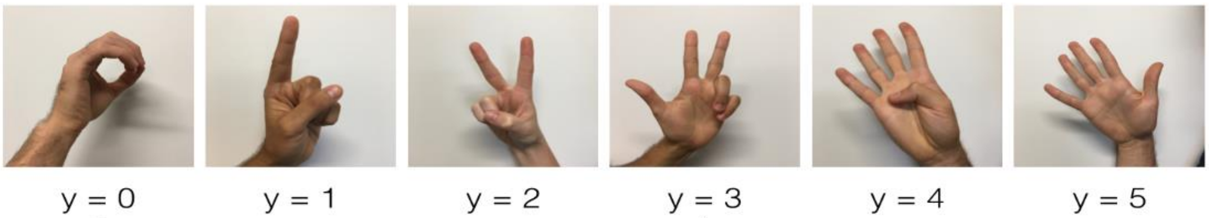
a. Implement the network architecture outlined in the PyTorch slides.

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolutio
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

b. Implement the ResNet architecture.

```
import torchvision.models
resnetmodels = models.resnet34(pretrained = True)
```

Evaluate and compare the performance results.



$$\begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\1\\0\\0\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\0\\1\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix}$$