Common CNN architectures

Classic network architectures (included for historical purposes)

- LeNet-5
- AlexNet
- VGG 16

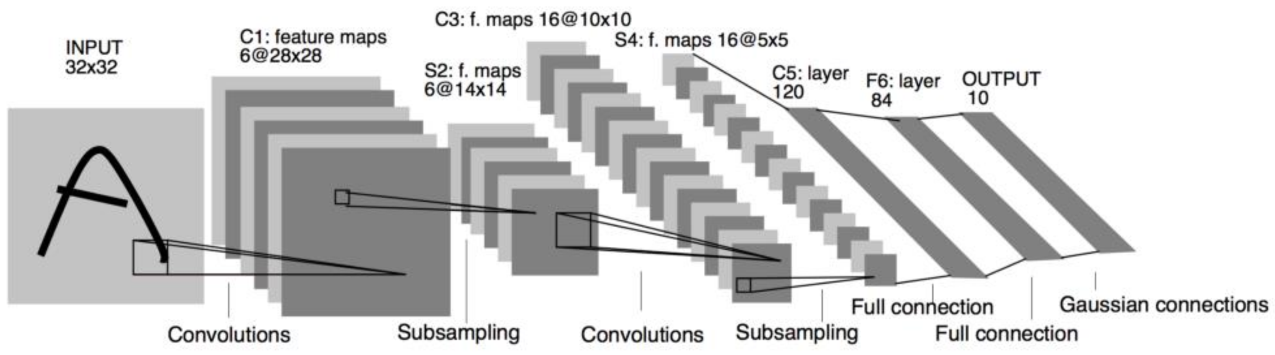
Modern network architectures

- Inception
- ResNet
- ResNeXt
- DenseNet

In this section we will talk about classic networks which are **LeNet-5**, **AlexNet**, and **VGG**., **ResNet, Inception**

- **LeNet-5 -** LeNet-5, a pioneering 7-level convolutional network by LeCun et al in 1998, that classifies digits, was applied by several banks to recognise hand-written numbers on checks (cheques) digitized in 32x32 pixel greyscale input images. The ability to process higher resolution images requires larger and more convolutional layers, so this technique is constrained by the availability of computing resources.

    ✓ It has 60k parameters.



Brief description

By modern standards, LeNet-5 is a very simple network. It only has 7 layers, among which there are 3 convolutional layers (C1, C3 and C5), 2 sub-sampling (pooling) layers (S2 and S4), and 1 fully connected layer (F6), that are followed by the output layer. Convolutional layers use 5 by 5 convolutions with stride 1. Sub-sampling layers are 2 by 2 average pooling layers. Tanh sigmoid activations are used throughout the network. There are several interesting architectural choices that were made in LeNet-5 that are not very common in the modern era of deep learning.
Ref - https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4

LeNet-5 layers:

1. Convolution #1. Input = 32x32x1. Output = 28x28x6 conv2d

2. SubSampling #1. Input = 28x28x6. Output = 14x14x6. SubSampling is simply Average Pooling so we use avg_pool
3. Convolution #2. Input = 14x14x6. Output = 10x10x16 conv2d
4. SubSampling #2. Input = 10x10x16. Output = 5x5x16 avg_pool
5. Fully Connected #1. Input = 5x5x16. Output = 120
6. Fully Connected #2. Input = 120. Output = 84
7. Output 10

A Keras implementation of LeNet-5 network would be extremely readable and maintainable. We can change it and experiment with it with ease.

```
model = keras.Sequential()

model.add(layers.Conv2D(filters=6, kernel_size=(3, 3),
activation='relu', input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())

model.add(layers.Conv2D(filters=16, kernel_size=(3, 3),
activation='relu'))
model.add(layers.AveragePooling2D())

model.add(layers.Flatten())

model.add(layers.Dense(units=120, activation='relu'))

model.add(layers.Dense(units=84, activation='relu'))

model.add(layers.Dense(units=10, activation = 'softmax'))
```

✓ AlexNet 2012

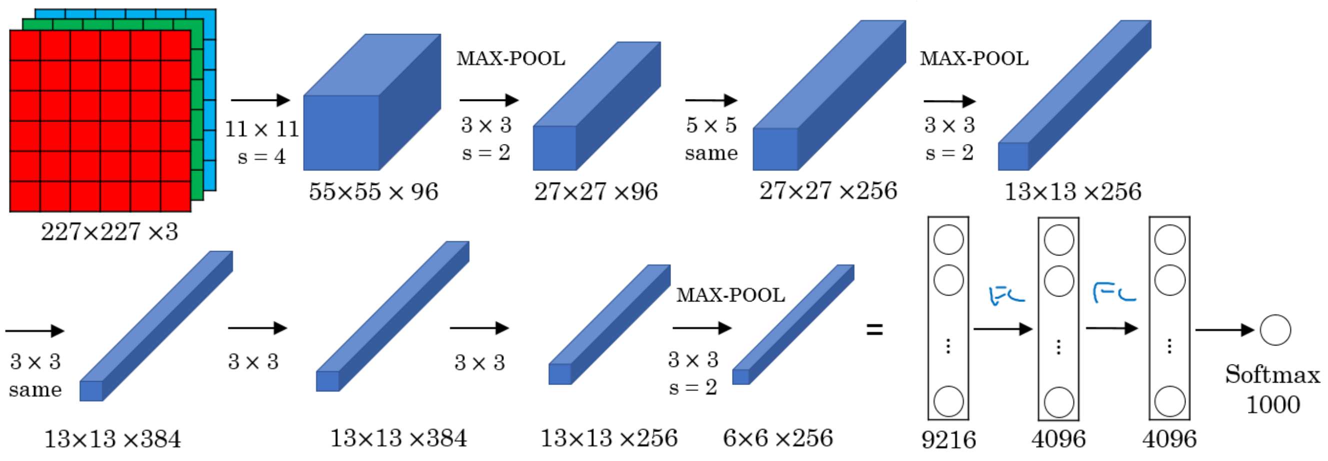
There are 5 convolutional layers, 3 fully connected layers and with Relu applied after each of them, and dropout applied before the first and second fully connected layer.

The highlights
Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.

1. Use dropout instead of regularisation to deal with overfitting. However the training time is doubled with the dropout rate of 0.5.
2. Overlap pooling to reduce the size of network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, repectively.

The architecture

It contains 5 convolutional layers and 3 fully connected layers. Relu is applied after very convolutional and fully connected layer. Dropout is applied before the first and the second fully connected year. The image size in the following architecutre chart should be 227 * 227 instead of 224 * 224, as it is pointed out by Andrei Karpathy in his famous CS231n Course. More insterestingly, the input size is 224 * 224 with 2 padding in the pytorch torch vision. The output width and height should be (224–11+4)/4 + 1=55.25! The explanation here is pytorch Conv2d apply floor operator to the above result, and therefore the last one padding is ignored.



- ✓ Has 60 Million parameter compared to 60k parameter of LeNet-5.

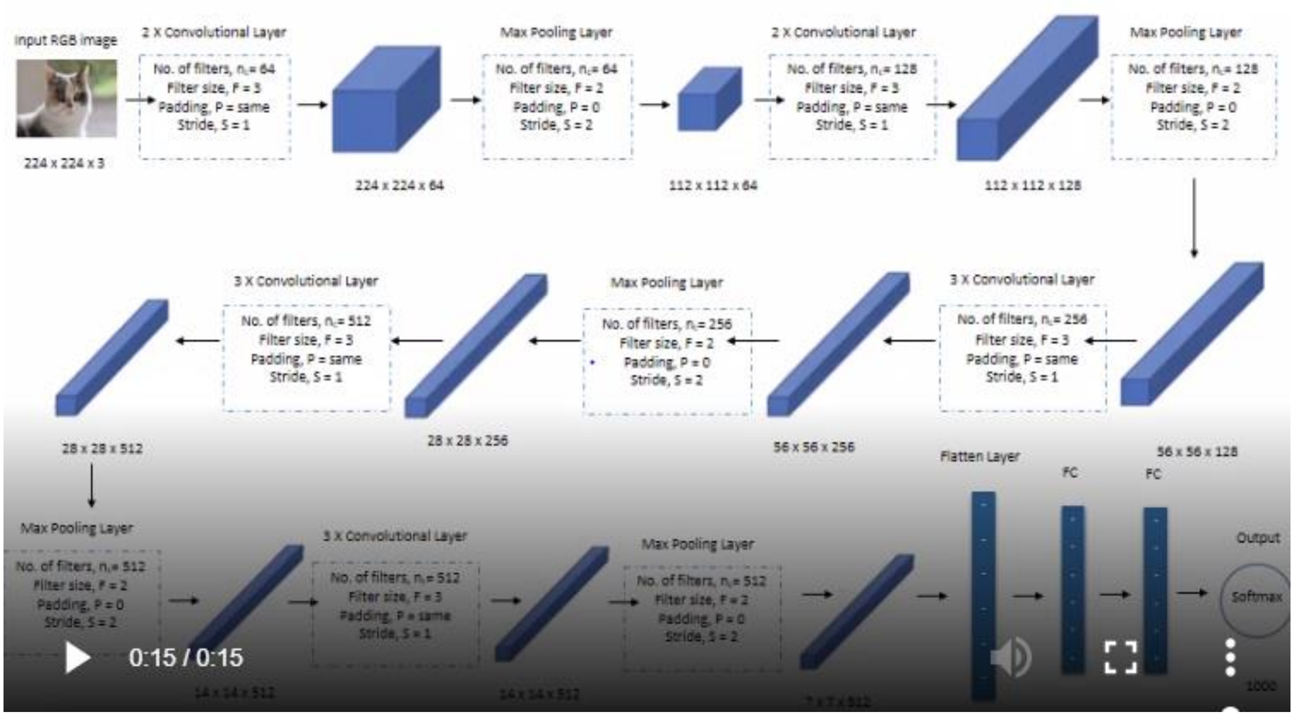
- **VGG16**

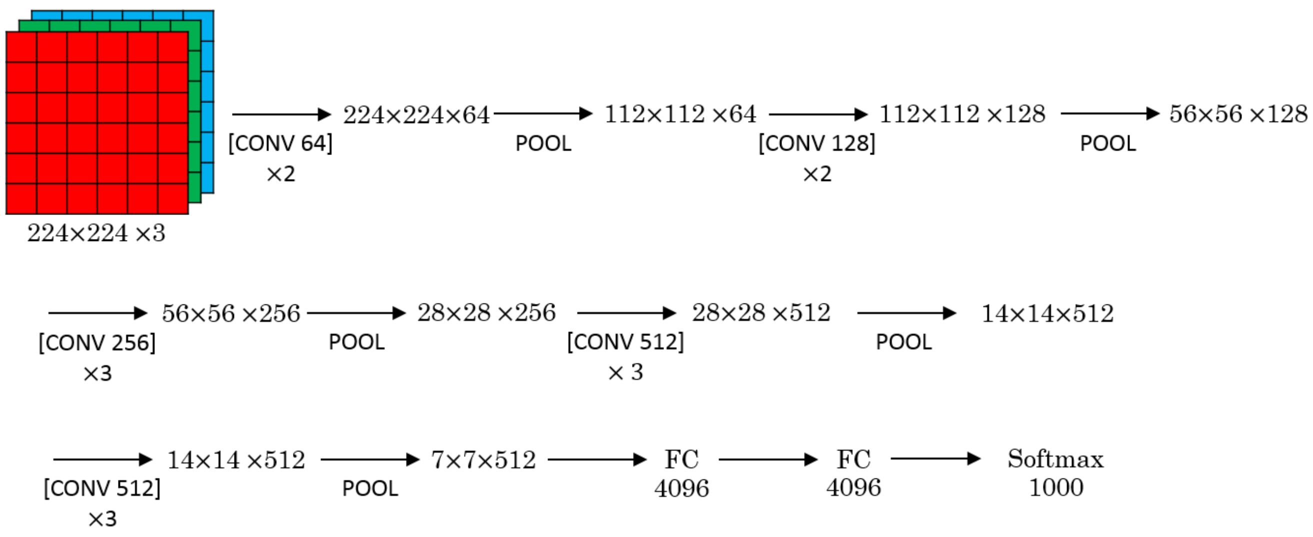
Karen Simonyan and Andrew Zisserman investigated the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. They increased the depth of their architecture to 16 and 19 layers with very small (3×3) convolution filters. They named their finding as VGG16 (Visual Geometry Group) and VGG19. The team won the first and the second places in the localization and classification tracks respectively at the ImageNet Challenge 2014 submission.

The VGG16 architecture consists of **twelve convolutional layers,** some of which are followed by maximum pooling layers and then four fully-connected layers and finally a 1000-way softmax classifier.

Focus on having only these blocks:

- CONV = 3 X 3 filter, s = 1, same
- MAX-POOL = 2 X 2 , s = 2





Source deeplearning.ai

First and Second Layers:
The input for AlexNet is a 224x224x3 RGB image which passes through first and second convolutional layers with 64 feature maps or filters having size 3×3 and same pooling with a stride of 14. The image dimensions changes to 224x224x64.
Then the VGG16 applies maximum pooling layer or sub-sampling layer with a filter size 3×3 and a stride of two. The resulting image dimensions will be reduced to 112x112x64.

Third and Fourth Layer:
Next, there are two convolutional layer with 128 feature maps having size 3×3 and a stride of 1.
Then there is again a maximum pooling layer with filter size 3×3 and a stride of 2. This layer is same as previous pooling layer except it has 128 feature maps so the output will be reduced to 56x56x128.

Fifth and Sixth Layers:
The fifth and sixth layers are convolutional layers with filter size 3×3 and a stride of one. Both used 256 feature maps.
The two convolutional layers are followed by a maximum pooling layer with filter size 3×3, a stride of 2 and have 256 feature maps.
Seventh to Twelveth Layer:
Next are the two sets of 3 convolutional layers followed by a maximum pooling layer. All convolutional layers have 512 filters of size 3×3 and a stride of one. The final size will be reduced to 7x7x512.

Thirteenth Layer:
The convolutional layer output is flatten through a fully connected layer with 25088 feature maps each of size 1×1.

Fourteenth and Fifteenth Layers:
Next is again two fully connected layers with 4096 units.

Output Layer:
Finally, there is a softmax output layer ŷ with 1000 possible values.

Summary of VGG16 Architecture

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Summary Table – VGG16 Architecture

- This network is large even by modern standards. It has around 138 million parameters.
    - Most of the parameters are in the fully connected layers.
- It has a total memory of 96MB per image for only forward propagation!
    - Most memory are in the earlier layers.

RESNET

Researchers observed that it makes sense to affirm that “***the deeper the better***” when it comes to convolutional neural networks. This makes sense, since the models should be more capable (their flexibility to adapt to any space increase because they have a bigger parameter space to explore). However, it has been noticed that after some depth, the performance degrades.

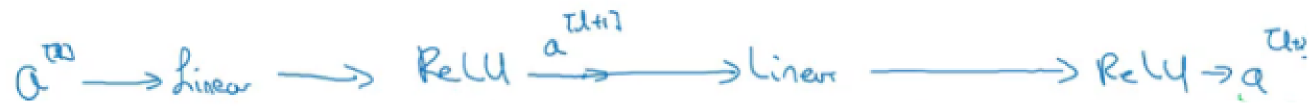
Deep NNs are difficult to train because of vanishing and exploding gradients problems. Skip connection which makes you take the activation from one layer and suddenly feed it to another layer even much deeper in NN which allows you to train large NNs even with layers greater than 100.

What problems ResNets solve?

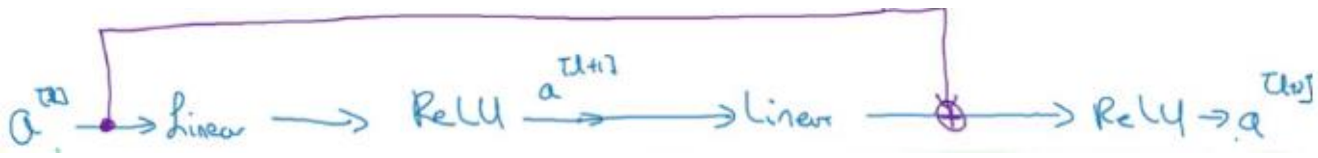
One of the problems ResNets solve is the famous known **vanishing gradient**. This is because when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This result on the weights never updating its values and therefore, no learning is being performed.

With ResNets, the **gradients can flow directly through the skip connections backwards from later layers to initial filters**.

This is how we calculate the activations a[l+2] using the activations a[l] and then a[l+1]. a[l] needs to go through all these steps to generate a[l+2]:



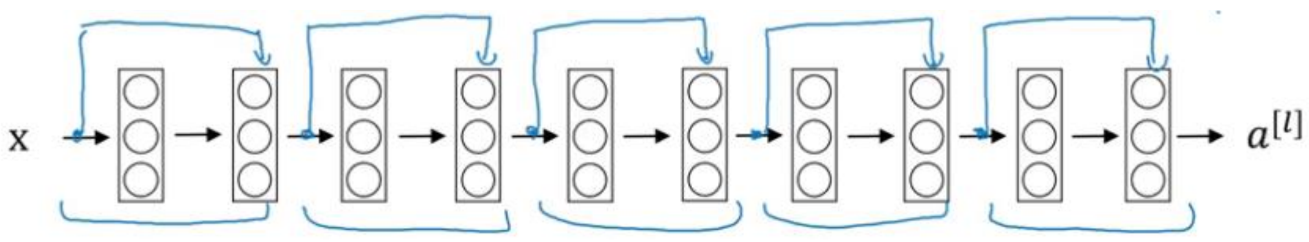
In a residual network, we make a change in this path. We take the activations a[l] and pass them directly to the second layer:



So, the activations a[l+2] will be:

a[l+2] = g(z[l+2] + a[l])

The residual network can be shown as:



1*1 Convolutions

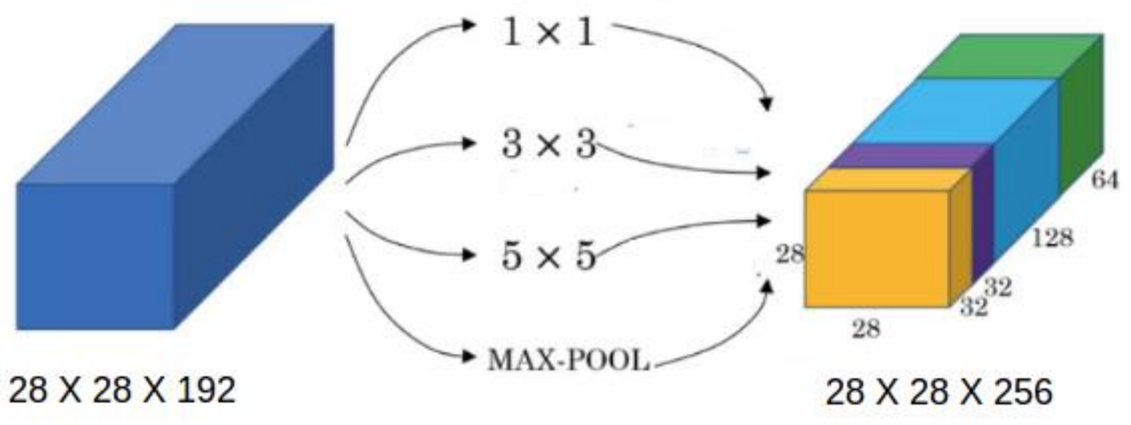
The basic idea of using 1 X 1 convolution is to reduce the number of channels from the image. A couple of points to keep in mind:

- We generally use a pooling layer to shrink the height and width of the image
- To reduce the number of channels from an image, we convolve it using a 1 X 1 filter (hence reducing the computation cost as well)

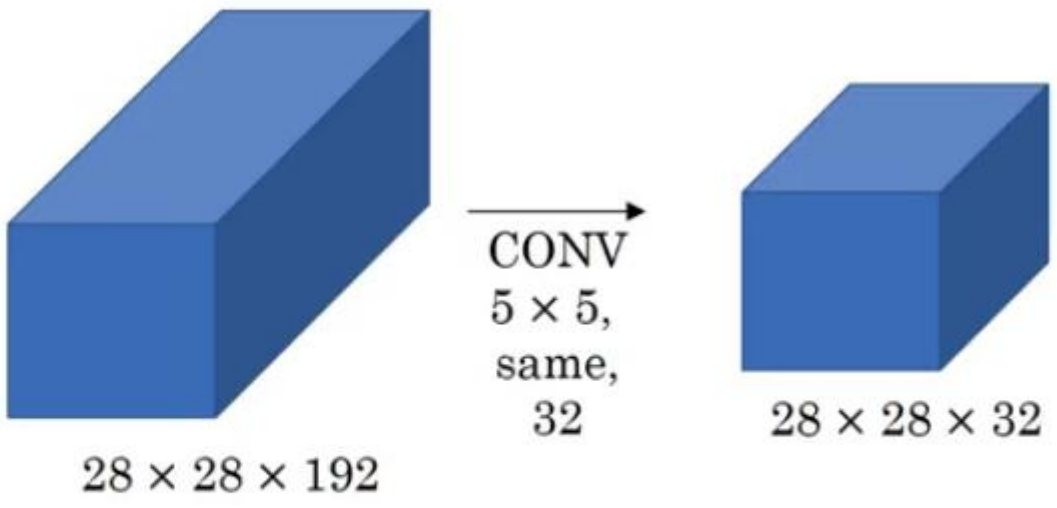
The Motivation Behind Inception Networks

While designing a convolutional neural network, we have to decide the filter size. Should it be a 1 X 1 filter, or a 3 X 3 filter, or a 5 X 5? Inception does all of that for us! Let's see how it works.

Suppose we have a 28 X 28 X 192 input volume. Instead of choosing what filter size to use, or whether to use convolution layer or pooling layer, inception uses all of them and stacks all the outputs:

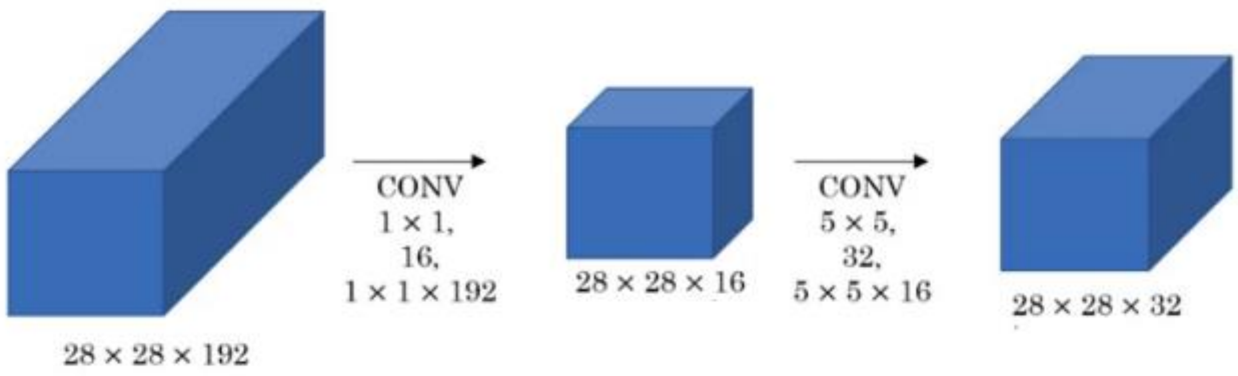


A good question to ask here – why are we using all these filters instead of using just a single filter size, say 5 X 5? Let's look at how many computations would arise if we would have used only a 5 X 5 filter on our input:



Number of multiplies = 28 * 28 * 32 * 5 * 5 * 192 = 120 million! Can you imagine how expensive performing all of these will be?

Now, let's look at the computations a 1 X 1 convolution and then a 5 X 5 convolution will give us:

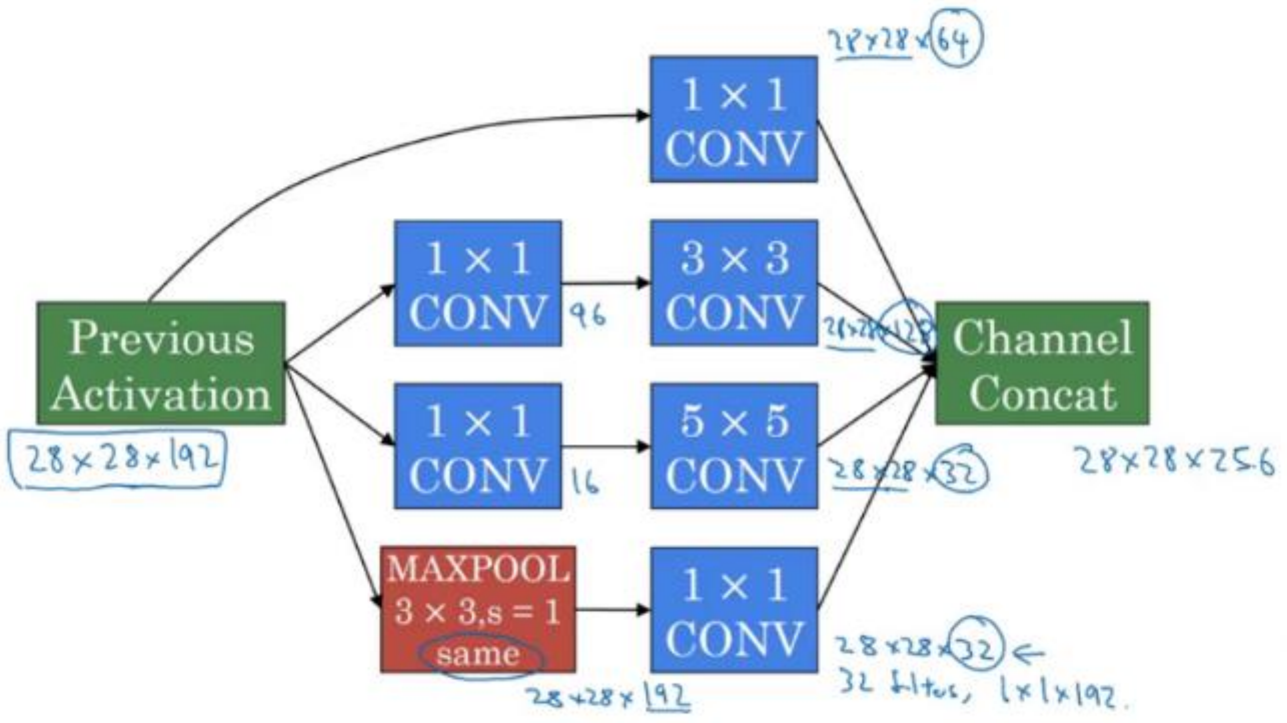


Number of multiplies for first convolution = 28 * 28 * 16 * 1 * 1 * 192 = 2.4 million
Number of multiplies for second convolution = 28 * 28 * 32 * 5 * 5 * 16 = 10 million
Total number of multiplies = 12.4 million

A significant reduction. This is the key idea behind inception.

Inception Networks

This is how an inception block looks:



We stack all the outputs together. Also, we apply a 1 X 1 convolution before applying 3 X 3 and 5 X 5 convolutions in order to reduce the computations. An inception model is the combination of these inception blocks repeated at different locations, some fully connected layer at the end, and a softmax classifier to output the classes.

References –

https://medium.com/@mgazar/lenet-5-in-9-lines-of-code-using-keras-ac99294c8086

https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

https://www.jeremyjordan.me/convnet-architectures/

https://engmrk.com/vgg16-implementation-using-keras/

https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

http://teleported.in/posts/decoding-resnet-architecture/