STEVENS INSTITUTE OF TECHNOLOGY

Deep Learning: Convolutional Neural Network

Rensheng Wang,
https://sit.instructure.com/courses/29876

October 20, 2019

1X1 Convolution

- Some CNN architectures use a 1x1 filter. In this case, the filter maps input of shape

  `( height_i, width_i, num_filters_i)`

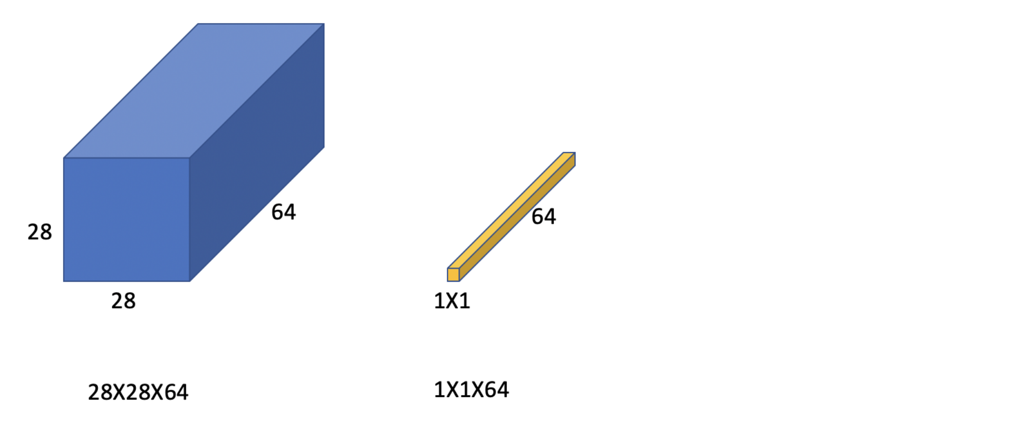
  to an output of shape:

  `( height_i, width_i, num_filters_o)`

- Note that only the number of features changes, while height and width remain the same.
- In this case, each output pixel is a vector of `num_filters_o` features that depends only on one input pixel (a vector of size `num_filters_i`).
- Each output feature is a (different) linear combination of the input features for the same pixel which is a receptive field of size 1x1.

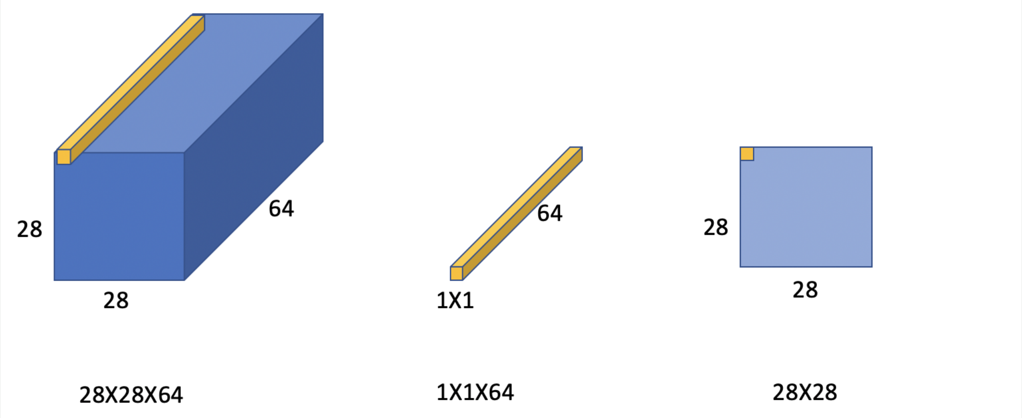
1X1 Convolution

- Example: 28X28X64 input, the filter size 1X1X64



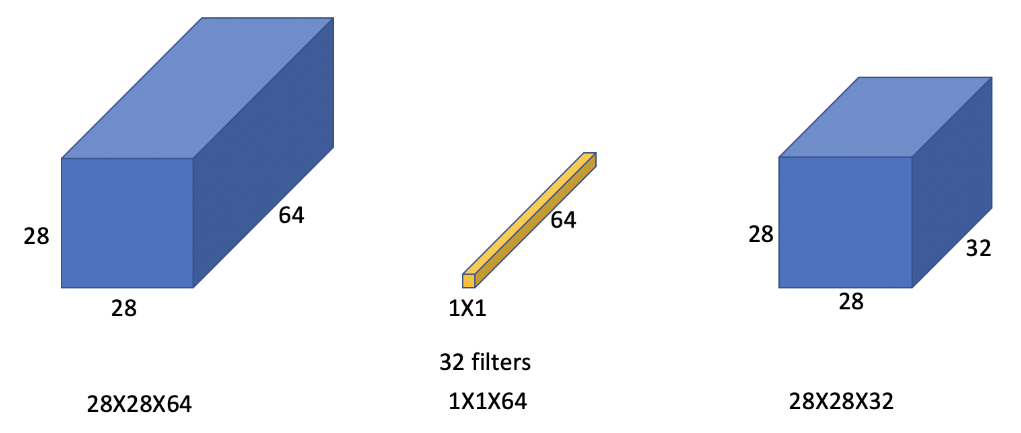
1X1 Convolution

- Example: 28X28X64 input, the filter size 1X1X64
- In this case, each output pixel is a vector of `num_filters_o` features that depends only on one input pixel (a vector of size `num_filters_i`).



1X1 Convolution

- Example: 28X28X64 input, the filter size 1X1X64
- Each output feature is a (different) linear combination of the input features for the same pixel which is a receptive field of size 1x1.



Why 1X1 Convolution?

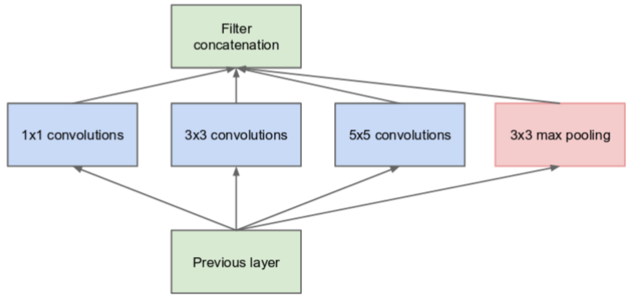
- The 1x1 filter is used to reduce the number of output features thus reducing the computational cost while keeping the spatial dimension unchanged.
- For example, the inception network uses 1x1 filters to reduce the features and create bottlenecks which make the architecture more computationally affordable. However, if the bottleneck is too tight it may end up hurting the network performances.
- When the size of the convolution kernel is larger than 1 x 1, each output feature is still a linear combination of all the input features in the receptive field, which in this case is ¿1 pixel wide.

Inception Network: motivations

- Salient parts in the image can have extremely large variation in size
- Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough
- Very deep networks are prone to overfitting
- Naively stacking large convolution operations is computationally expensive

Inception Network

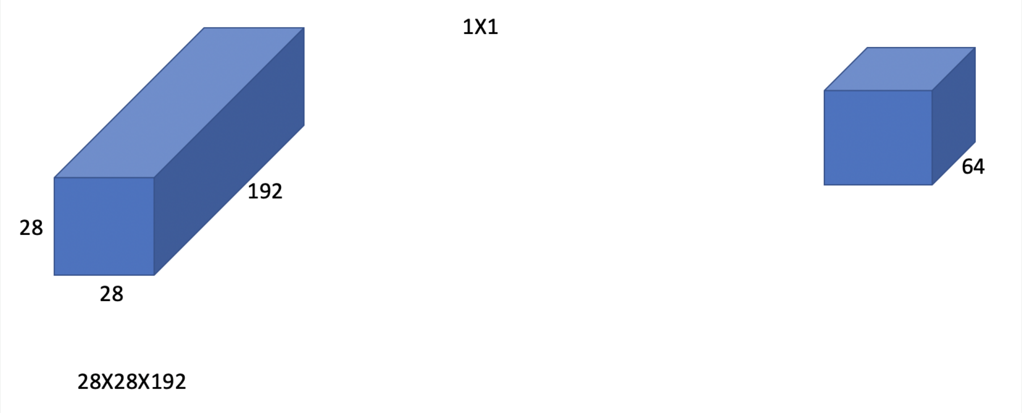
- Solution: Why not have filters with multiple sizes operate on the same level? The network essentially would get a bit wider rather than deeper.
- It performs convolution on an input, with 3 different sizes of filters (1x1, 3x3, 5x5).
- Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module.



(a) Inception module, naïve version

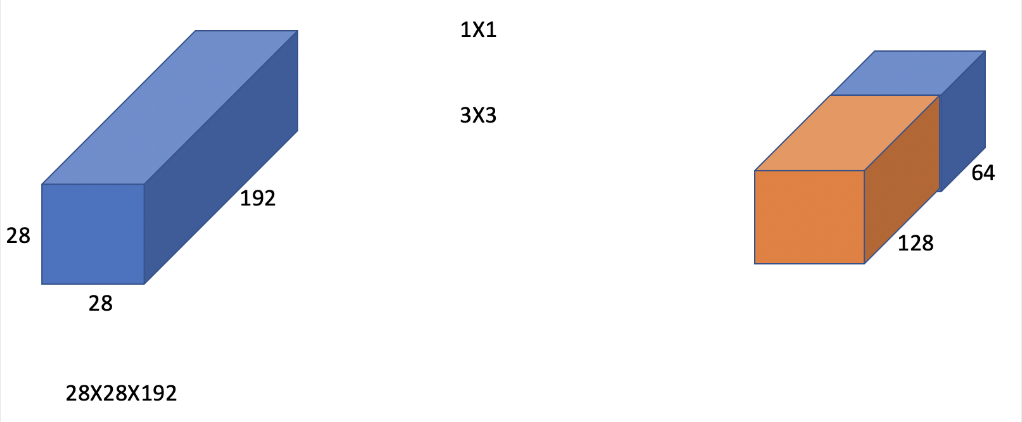
Inception Network

- Example: 1X1 filter



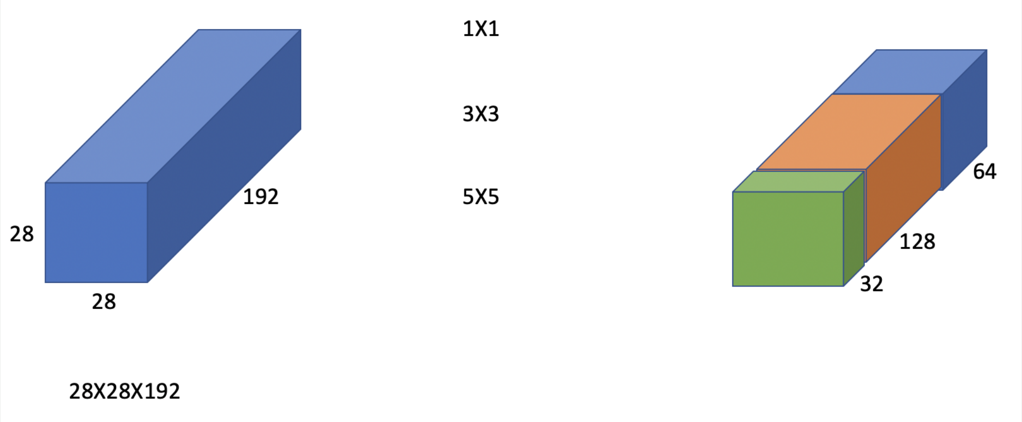
Inception Network

- Example: 3X3 filter



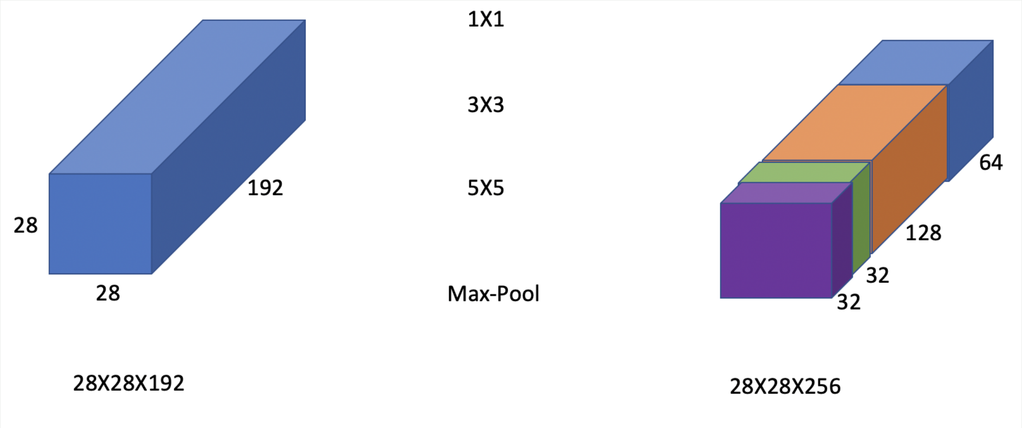
Inception Network

- Example: 5X5 filter



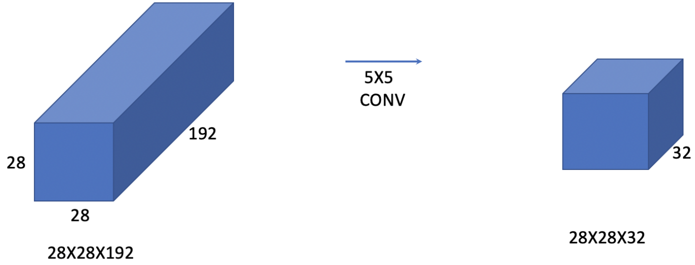
Inception Network

- Example: Max-Pooling



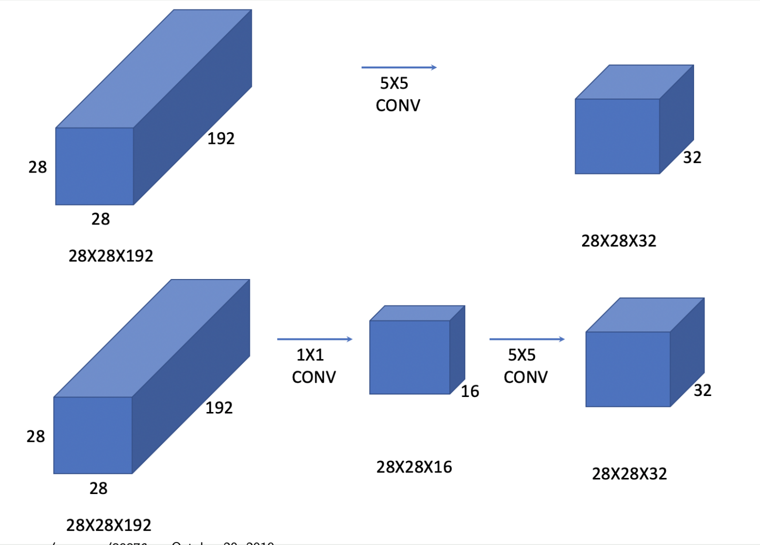
Problem of Computational Cost

- The computational cost from 5X5 convolutional filters is tremendous



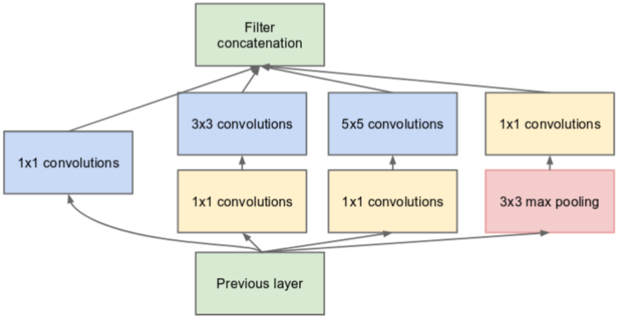
Problem of Computational Cost

- Using 1X1 convolution before 5X5 convolutional filters can reduce the feature size and therefore decrease the computational cost significantly.



Inception Network

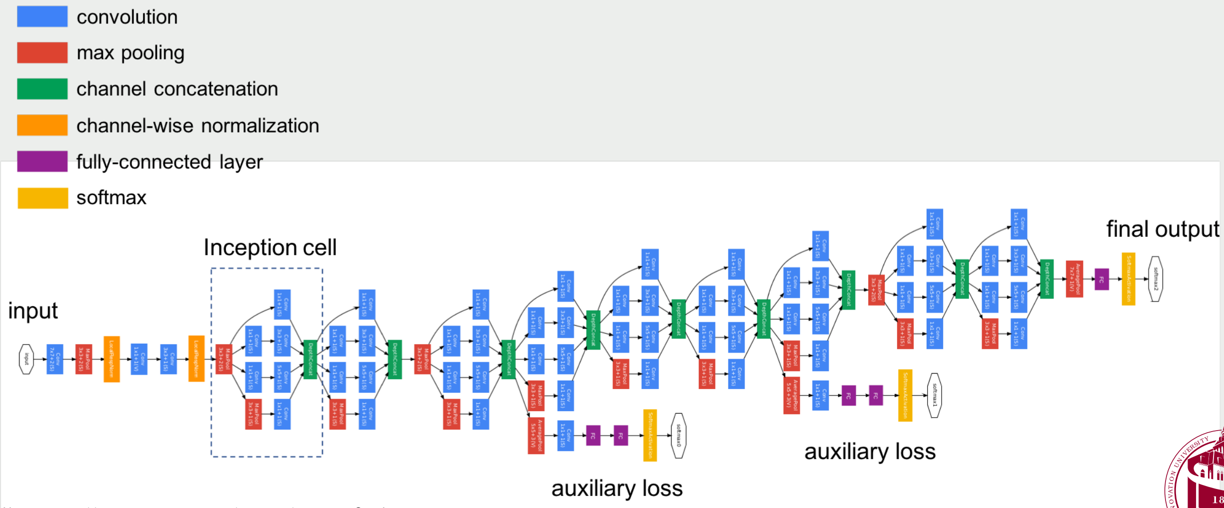
- To make it cheaper, the authors limit the number of input channels by adding an extra 1x1 convolution before the 3x3 and 5x5 convolutions.
- Though adding an extra operation may seem counterintuitive, 1x1 convolutions are far more cheaper than 5x5 convolutions, and the reduced number of input channels also help.



(b) Inception module with dimension reductions

Inception Network

- Using the dimension reduced inception module, a neural network architecture was built. This was popularly known as GoogLeNet (Inception v1).



Inception Network

- To prevent the middle part of the network from dying out, the authors introduced two auxiliary classifiers. They essentially applied softmax to the outputs of two of the inception modules, and computed an auxiliary loss over the same labels. The total loss function is a weighted sum of the auxiliary loss and the real loss.
- # The total loss used by the inception net during training.

  `total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2`