Show, attend and tell

Inception v3

Name : Monique Ehab
ID : 4928

Inception v3 Model :

- Inception v3 is a widely-used image recognition model. The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers.
- Loss is computed via Softmax.
- The pre-trained network on ImageNET can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

Why did it emerge ?

Prior to its inception most popular CNNs just stacked convolution layers deeper and deeper, hoping to get better performance.

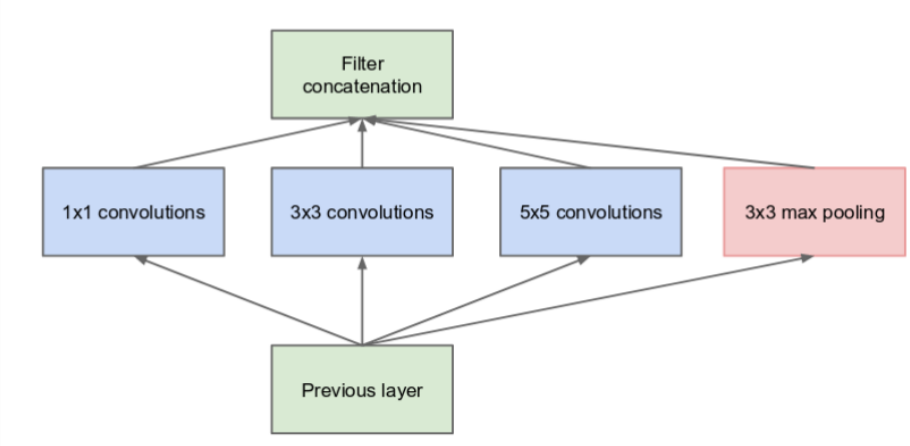
- Very deep networks are prone to over-fitting. It also hard to pass gradient updates through the entire network.
- Naively stacking large convolution operations is computationally expensive.

Main idea of Inception :

Having filters with multiple sizes operate on the same level. The network essentially would get a bit “wider” rather than “deeper”. It allows the internal layers to pick and choose which filter size will be relevant to learn the required information. So even if the size of the object in two different images is different, the layer works accordingly to recognize the object according to its size.

It performs convolution on an input, with 3 different sizes of filters (1×1, 3×3, 5×5). Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module.

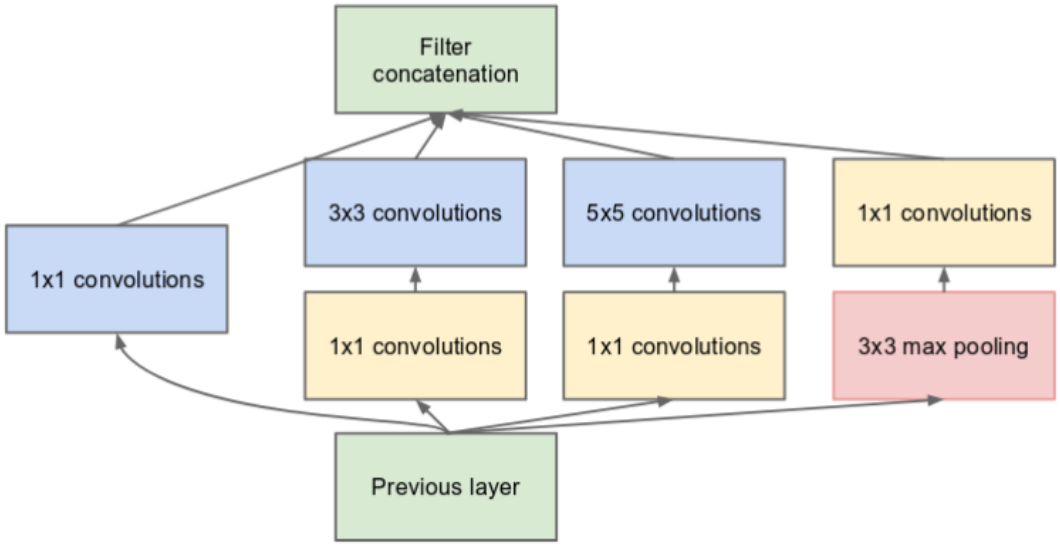
A single inception module :



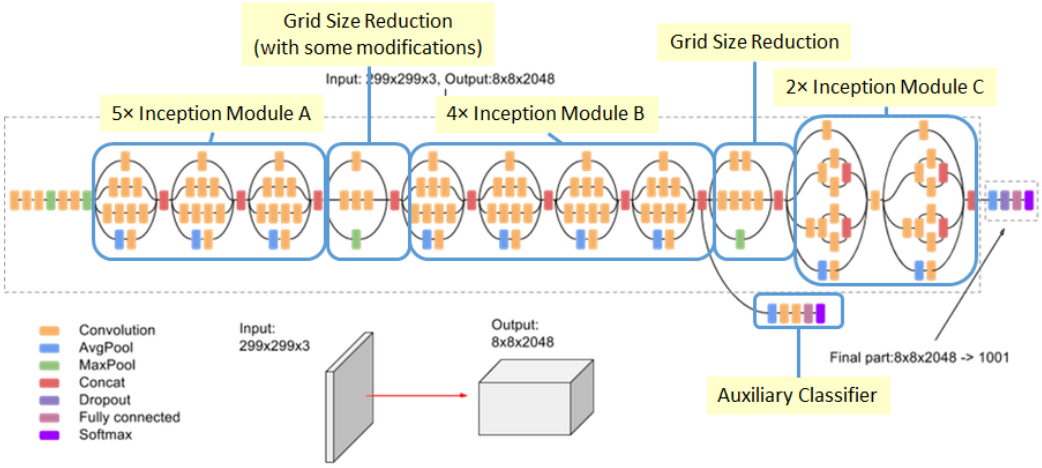
Step 2 :

Dimension reduction :

Deep neural networks are computationally expensive. To make it cheaper, we add an extra 1×1 convolution before the 3×3 and 5×5 convolutions., 1×1 convolutions are far more cheaper than 5×5 convolutions, they shrink down the number of input channels and the reduced number of input channels also help. The 1×1 convolution is introduced after the max pooling layer.



Architecture :



1- It consists of repeated inception modules with pooling layers in between.
2- A fully connected layer and a softmax layer at the end for classification.
3- Auxiliary Classifiers to prevent the middle part of the network from "dying out". The total loss function is a weighted sum of the auxiliary loss and the real loss.

Versions :

1) Inception v1
2) Inception v2
3) Inception v3
4) Inception v4

Inception v1:

It has *9* such inception modules fitted linearly. It is *22* layers deep and can be considered *27* if we included the pooling layers. At the end of the architecture, fully connected layers were replaced by a global average pooling which calculates the average of every feature map. This indeed dramatically declines the total number of parameters.

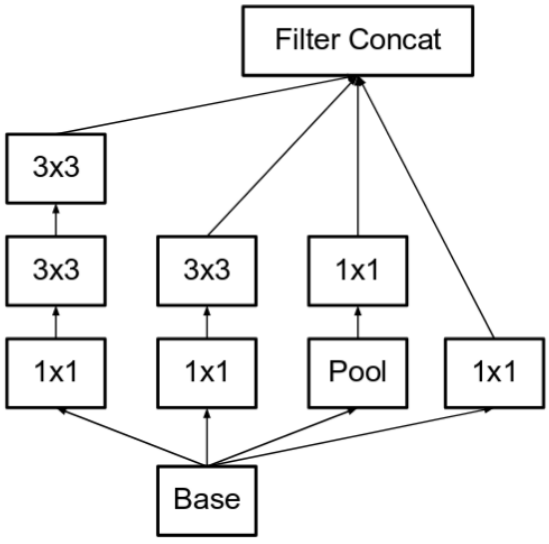
Problems found with v1:

- Representational bottleneck : caused by the drastic change in dimensions made by the convolutions which worsen the performance of neural networks.

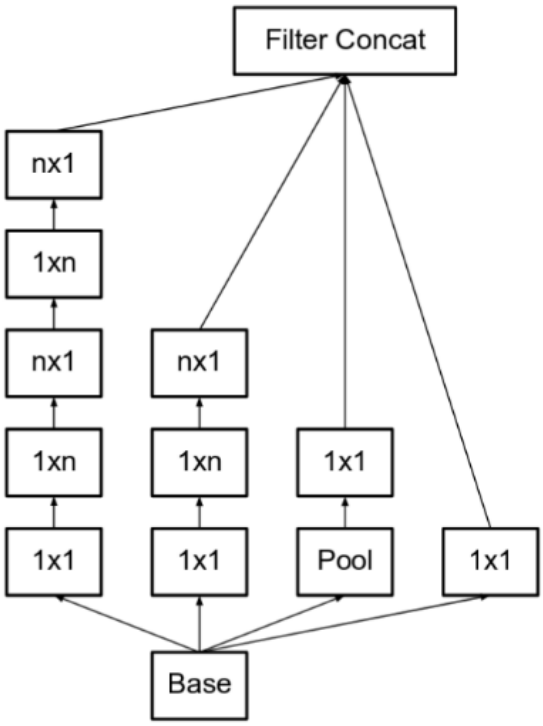
Additions to the new version v2 :

- Using smart factorization methods, by which convolutions can be made more efficient in terms of computational complexity.

Here we can see the factorization of a 5×5 convolution to two 3×3 convolution operations to improve computational speed. Stacking two 3×3 convolutions leads to a boost in performance.



A further step of factorization, factorize convolutions of filter size nxn to a combination of 1xn and nx1 convolutions. They found this method to be 33% more cheaper than the single n×n convolution.



Final architectural view of the v2 :

- The filters in the module were made wider instead of deeper to remove the representational bottleneck that causes under-fitting by excessive reduction in dimensions, and hence loss of information.

Added features in v3 :

1. ***RMSProp Optimizer*** : it is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated.
2. ***Factorized 7×7 convolutions***.
3. ***BatchNorm in the Auxillary Classifiers***: by which batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.
4. ***Label Smoothing*** which is a type of regularizing component added to the loss formula that prevents the network from over fitting.

Added features in v4 :

Make the modules more uniform to boost the performance.

Code :

Model input : It takes images of size (3*299*299), so we resize the images of COCO dataset.

```
images = torch.Tensor(1,3,256,256).uniform_()
print("1. " + str(images.size()))
images = resize2d(images,(299,299))
```

```
Tracing, in train started forward propagation-enc
1. torch.Size([1, 3, 256, 256])
2. torch.Size([1, 3, 299, 299])
```

Model formation :

1) Use pytorch pre-trained Inception v3 model on ImageNet.
2) Exclude the last layer of classification from the model.
3) Pooling to resize the image.
4) Fine tuning : to prevent the computation of gradients for convolutional blocks in the background.

```
#  pretrained ImageNet inception_v3
model = torchvision.models.inception_v3(pretrained=True)

modules = list(model.children())[:-1]
self.inception=nn.Sequential(*modules)
self.adaptive_pool = nn.AdaptiveAvgPool2d((encoded_image_size, encoded_image_size))
self.fine_tune()
```

Output of the model (8*8*2048 channels) :

The decoder input :

```
5. torch.Size([1, 2048, 8, 8])
torch.Size([1, 2048, 8, 8])
6. torch.Size([1, 8, 8, 2048])
7. torch.Size([1, 8, 8, 2048])
```

Layers of inception model :

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Going through the whole code :

1- ***Reading*** the COCO dataset.

2- ***Adjusting the model parameters*** because all pre-trained models expect input images normalized in the same way, in inception the input size is 3-channel RGB images of shape (3 x H x W), where H and W are 299. The images must be normalized.

3- ***Encoder*** : we initialize the inception_v3 as mentioned above together with pooling to obtain the required size of 8x8 images with 2048 channels.

4- ***Attention*** : obtaining the soft attention through using linear layers to transform the encoded image, then to transform decoder's output, then to calculate values to be softmax-ed and finally a softmax layer to calculate weights and the ReLU function which is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

5- ***Decoder*** : after initializing some parameters with values from the uniform distribution, we create the initial hidden and cell states for the decoder's LSTM based on the encoded images.

6- ***Create tensors*** to hold word prediction scores and alphas and decoder outputs.

7- ***At last step we have get printed scores of loss, accuracy and BLEU score***.