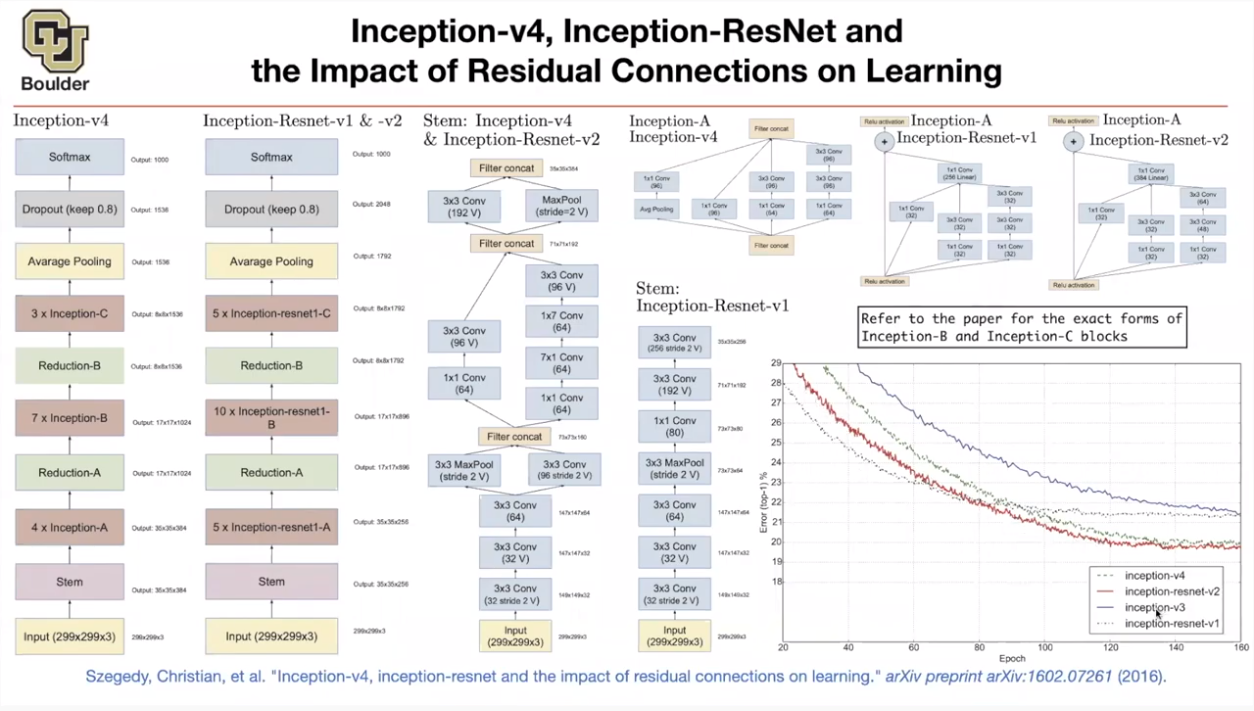
InceptionNet



# Version 1 | Going Deeper with Convolutions

#### Paper Link

[Going Deeper with Convolutions](https://arxiv.org/pdf/1409.4842.pdf)

#### Explained

[C4W2L06 Inception Network Motivation](https://www.youtube.com/watch?v=C86ZXvgpejM)

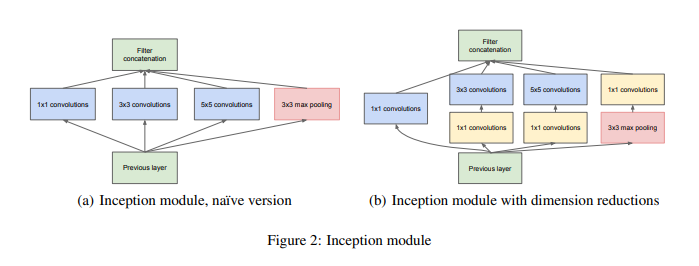
[C4W2L07 Inception Network](https://www.youtube.com/watch?v=KfV8CJh7hE0)

#### Summary

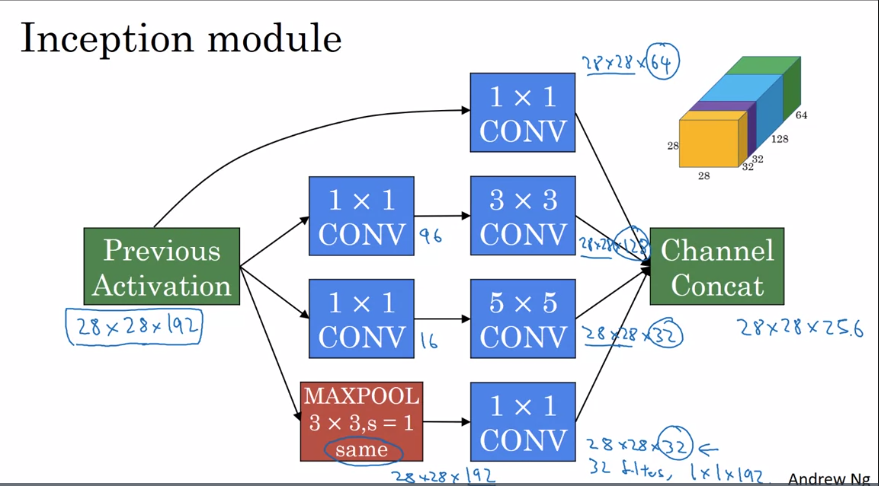
Because of the huge variation in the location of the information in images, choosing the **right kernel size** for the convolution operation becomes tough. A **larger kernel** is preferred for information that is distributed more **globally**, and a **smaller kernel** is preferred for information that is distributed more **locally.** So when designing a layer for a ConvNet, you might have to pick, do you want a 1 by 3 filter, or 3 by 3, or 5 by 5, or do you want a pooling layer? What the inception network does is it says, why don’t you do them all? And this makes the network architecture more complicated, but it also works remarkably well.

The main hallmark of this architecture is the improved utilization of the computing resources inside the network. This was achieved by a carefully crafted design that allows for increasing the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network.

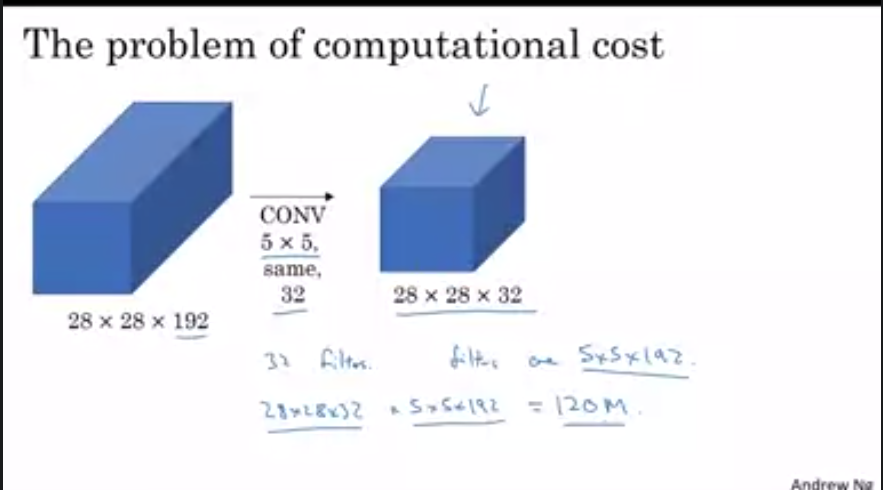
## Architecture

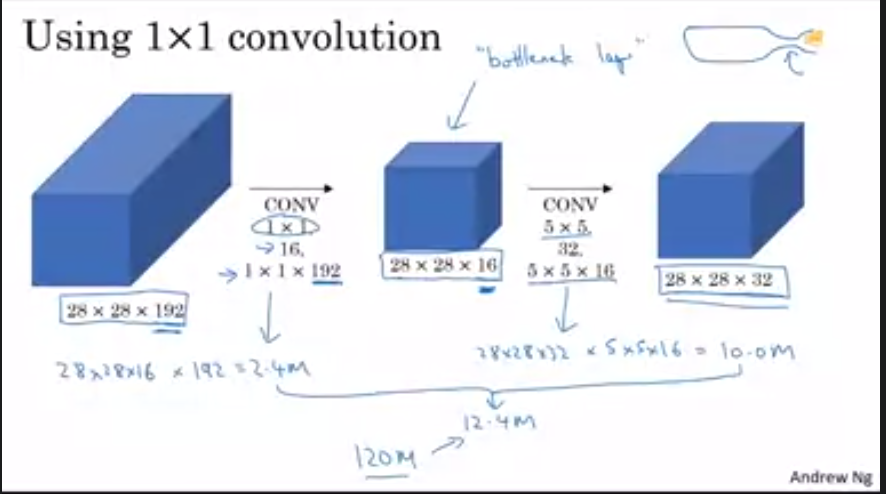


***So the inception module takes as input the activation or the output from some previous layer. So let's say for the sake of argument this is 28 by 28 by 192. The example we worked through in depth was 1 by 1 followed by 5 by 5. There, so maybe the 1 by 1 has 16 channels and then the 5 by 5 will output a 28 by 28 by, let's say, 32 channels. And this is the example we worked through on the last slide of the previous video. Then to save computation on your 3 by 3 convolution you can also do the same here. And then the 3 by 3 outputs, 28 by 28 by 1 by 28. And then maybe you want to consider a 1 by 1 convolution as well. There's no need to do a 1 by 1 conv followed by another 1 by 1 conv so there's just one step here and let's say these outputs 28 by 28 by 64. And then finally is the pulling layer. So here I'm going to do something funny. In order to really concatenate all of these outputs at the end we are going to use the same type of padding for pooling. So that the output height and width is still 28 by 28. So we can concatenate it with these other outputs. But notice that if you do max-pooling, even with the same padding, a 3 by 3 filter is tried at 1. The output here will be 28 by 28, By 192. It will have the same number of channels and the same depth as the input that we had here.***

******

***So, it seems like it has a lot of channels. So what we're going to do is actually add one more 1 by 1 conv layer to what we saw in the one by one convilational video, to strengthen the number of channels. So it gets us down to 28 by 28 by let's say, 32. And the way you do that is to use 32 filters, of dimension 1 by 1 by 192. So that's why the output dimension has a number of channels shrunk down to 32. So then we don't end up with the pulling layer taking up all the channels in the final output. And finally you take all of these blocks and you do channel concatenation. Just concatenate across this 64 plus 128 plus 32 plus 32 and this if you add it up this gives you a 28 by 28 by 256 dimension output. Concat is just this, concatenating the blocks that we saw in the previous video. So this is one inception module, and what the inception network does, is, more or less, put a lot of these modules together.***





***But what we've done is we're taking this huge volume we had on the left, and we shrunk it to this much smaller intermediate volume, which only has 16 instead of 192 channels. Sometimes this is called a bottleneck layer, right? I guess because a bottleneck is usually the smallest part of something, right? So I guess if you have a glass bottle that looks like this, then you know this is I guess where the cork goes. And then the bottleneck is the smallest part of this bottle. So in the same way, the bottleneck layer is the smallest part of this network. We shrink the representation before increasing the size again. Now let's look at the computational costs involved. To apply this 1 by 1 convolution, we have 16 filters. Each of the filters is going to be of dimension 1 by 1 by 192, this 192 matches that 192. And so the cost of computing this 28 by 28 by 16 volumes is going to be well, you need these many outputs, and for each of them you need to do 192 multiplications. I could have written 1 x 1 x 192, right? Which is this. And if you multiply this out, this is 2.4 million, it's about 2.4 million. How about the second? So that's the cost of this first convolutional layer. The cost of this second convolutional layer would be that well, you have these many outputs. So 28 by 28 by 32. And then for each of the outputs you have to apply a 5 by 5 by 16 dimensional filter. And so by 5 by 5 by 16. And you multiply that out is equal to 10.0. And so the total number of multiplications you need to do is the sum of those which is 12.4 million multiplications. And if you compare this with what we had on the previous slide, you reduce the computational cost from about 120 million multiplies, down to about one tenth of that, to 12.4 million multiplications. And the number of additions you need to do is about very similar to the number of multiplications you need to do. So that's why I'm just counting the number of multiplications. So to summarize, if you are building a layer of a neural network and you don't want to have to decide, do you want a 1 by 1, or 3 by 3, or 5 by 5, or pooling layer, the inception module let's you say let's do them all, and let's concatenate the results. And then we run into the problem of computational cost. And what you saw here was how using a 1 by 1 convolution, you can create this bottleneck layer thereby reducing the computational cost significantly. Now you might be wondering, does shrinking down the representation size so dramatically, does it hurt the performance of your neural network? It turns out that so long as you implement this bottleneck layer so that within reason, you can shrink down the representation size significantly, and it doesn't seem to hurt the performance, but saves you a lot of computation.***

**Full Architecture** [GoogLeNet-network-with-all-the-bells-and-whistles.png](https://www.researchgate.net/profile/Dumitru-Erhan/publication/305196650/figure/fig1/AS:614000287031351@1523400484514/GoogLeNet-network-with-all-the-bells-and-whistles.png)

***It turns out that there's one last detail to the inception network if we read the optional research paper. Which is that there are these additional side-branches that I just added. So what do they do? Well, the last few layers of the network are a fully connected layer followed by a softmax layer to try to make a prediction. What these side branches do is it takes some hidden layer and it tries to use that to make a prediction. So this is actually a softmax output and so is that. And this other side branch, again it is a hidden layer, passes through a few layers like a few connected layers. And then has the softmax try to predict what the output label is. And you should think of this as maybe just another detail of the inception that's worked. But what it does is it helps to ensure that the features are computed. Even in the heading units, even at intermediate layers. That they're not too bad for protecting the output because of an image. And this appears to have a regularizing effect on the inception network and helps prevent this network from overfitting. And by the way, this particular Inception network was developed by authors at Google. Who called it GoogleNet, spelled like that, to pay homage to the network.***

## Dataset

ImageNet - ILSVRC 2014

## Unique Features Introduced

GoogLeNet has **9 such inception modules** stacked linearly. It is **22 layers deep** (27, including the pooling layers). It uses global average pooling at the end of the last inception module.

Needless to say, it is a pretty deep classifier. As with any very deep network, it is **subject to the vanishing gradient problem**.

To prevent the middle part of the network from “**dying out**”, the authors introduced two auxiliary classifiers (The purple boxes in the image). 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. Weight value used in the paper was 0.3 for each auxiliary 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

Needless to say, auxiliary loss is purely used for training purposes, and is ignored during inference.

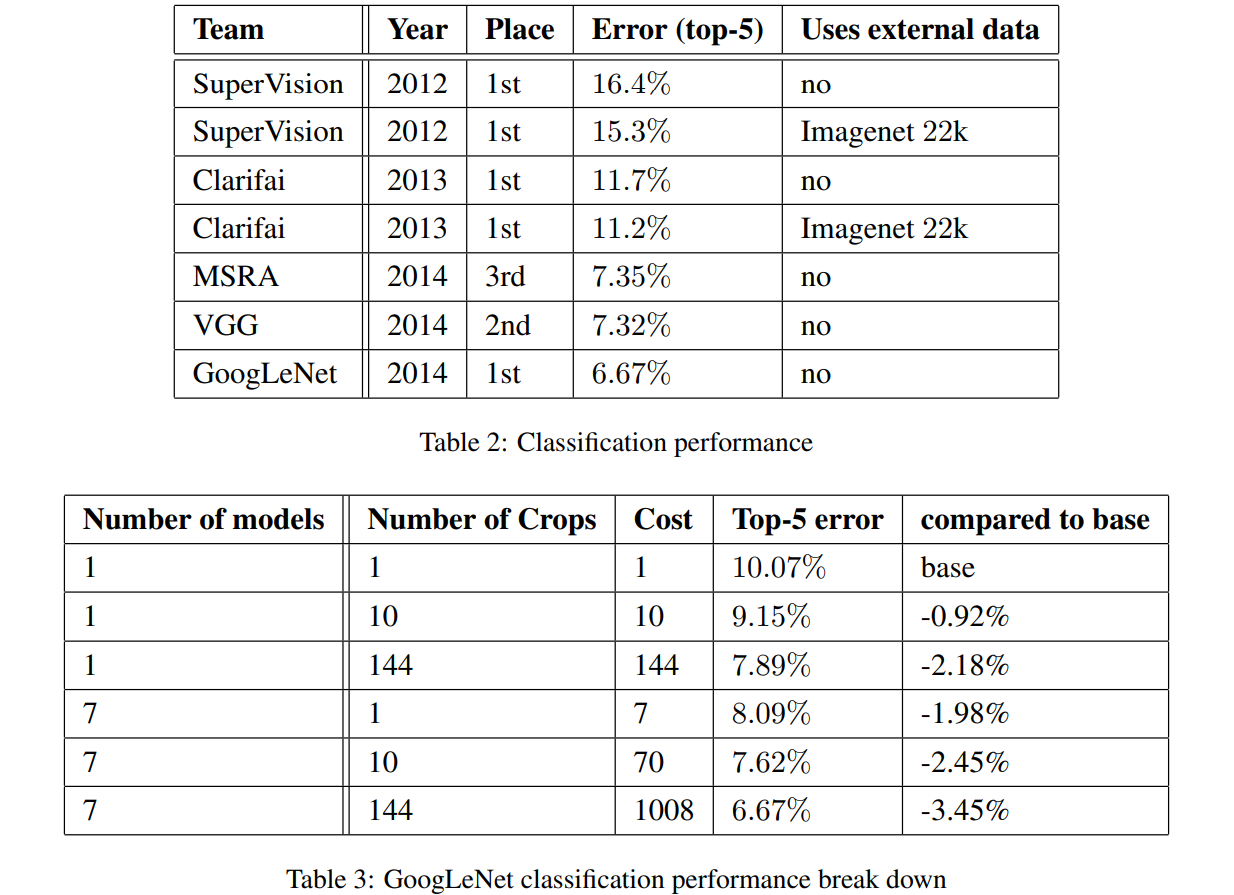
## Hyperparameters Setting

Our networks were trained using the DistBelief [4] distributed machine learning system using a modest amount of model and data-parallelism. Although we used CPU-based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using a few high-end GPUs within a week, the main limitation being the memory usage.

*Our training used* ***asynchronous stochastic gradient descent with 0.9 momentum*** *[17], fixed learning rate schedule (****decreasing the learning rate by 4% every 8 epochs****).* ***Polyak averaging*** *[13] was used to create the final model used* ***at inference time****.*

*To complicate matters further, some of the models were mainly trained on smaller relative crops, others on larger ones, inspired by [8]. Still,* ***one prescription that was verified to work very well after the competition includes sampling of various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area and whose aspect ratio is chosen randomly between 3/4 and 4/3****. Also, we found that the photometric distortions by Andrew Howard [8] were useful to combat overfitting to some extent. In addition, we started to use random interpolation methods (bilinear, area, nearest neighbor and cubic, with equal probability) for resizing relatively late and in conjunction with other hyperparameter changes, so we could not tell definitely whether the final results were affected positively by their use.*

## Evaluation And Result



# Version 2 and 3 | Rethinking the Inception Architecture for Computer Vision

#### Paper Link

[Rethinking the Inception Architecture for Computer Vision](https://arxiv.org/pdf/1512.00567v3.pdf)

#### Explained

#### Summary

# Version 4 | Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

#### Paper Link

[Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning](https://arxiv.org/pdf/1602.07261.pdf)

#### Explained

#### Summary

# Sources to Clear Concepts on Inception

1. [A Simple Guide to the Versions of the Inception Network | by Bharath Raj | Towards Data Science](https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202) \*\*\*\*
2. [Inception V1 Architecture Explained | by Abheer Bandodker | Medium](https://medium.com/@abheerchrome/inception-v1-architecture-explained-454b2eb66baf)
3. [Inception Network | Implementation Of GoogleNet In Keras](https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/#:~:text=The%20paper%20proposes%20a%20new,a%20layer%20called%20inception%20layer).
4. [GitHub - calmisential/TensorFlow2.0\_InceptionV3: A TensorFlow\_2.0 implementation of InceptionV3.](https://github.com/calmisential/TensorFlow2.0_InceptionV3)
5. [Tensorflow implementation of Inception Module](https://github.com/Natsu6767/Inception-Module-Tensorflow)