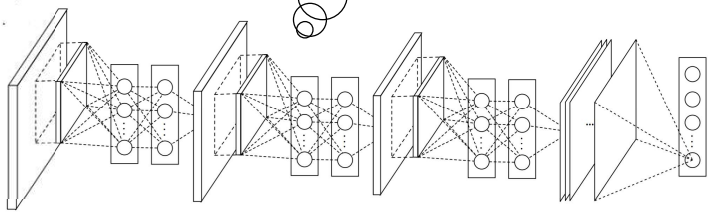
Deep Learning – NiN, Inception

Bernhard Kainz

Networks in Networks

Non-linear mapping layers introduced as mlpconv. Consists of multiple fully connected layers with non-linear activation function



Deep Learning – Bernhard Kainz

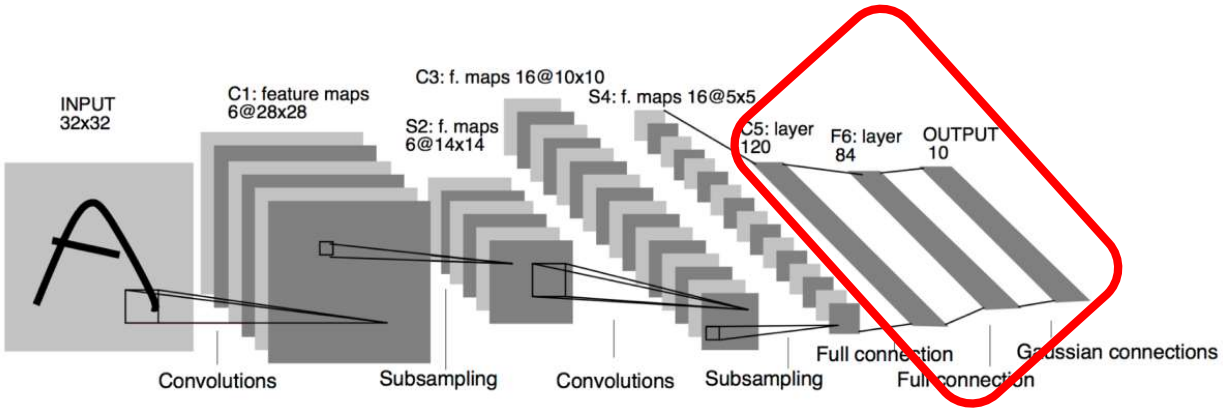
how can we make these CNNS a little bit smaller without loosing much accuracy?

networks and networks are a convenient strategy for doing this.

If you look at this figure you see that there's a multi-layer perceptron inserted as intermediate layers in my convolutional Network.

when that idea was published people thought that this was silly mainly because it didn’t outperform larger state of the art models.

Networks in Networks



Deep Learning – Bernhard Kainz

if we look at the last layers of LeNet, well they're not that big. I mean it's basically just 120 hidden units 84 and then 10 for output.

In Alex Net and VGG, well those things get quite massive, so if you look at the last layer just connecting from the convolution you have
number of channels times the width and the height of the last resolution.

that was 48,000 parameters in LeNet, in AlexNet and 26 million parameters in VGG 4 times that number

whereas convolutions were actually kind of well behaved it's input times output times convolutional kernel size if you had a kernel size of three then that's nine times input times output.
that's quite well behaved.

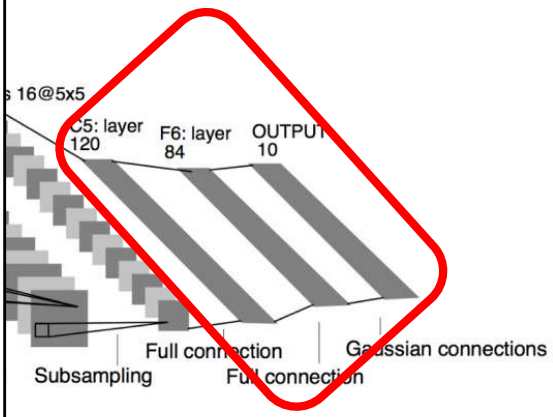
but those last layers are the ones that really made life hard.

so the question is of course how can you fix it. And the issue is in the end you need to produce something that is of a dimensionality that matches the number of classes that you have.

at some point you need to go from a two dimensional times channels representation to something that's a vector.

the challenge is how do you do that.

Networks in Networks



- Convolution layers are parameter cheap

$$c_i \times c_0 \times k^2$$

- Last layer is parameter expensive for n classes

$$c \times m_w \times m_h \times n$$

- LeNet: 16x5x5x120 = 48k parameters
- AlexNet: 256x5x5x4096 = 26M
- VGG 512x7x7x4096 = 103M

let's have a look at it in a bit more detail

if I print the size distribution for vgg you can see that as I'm going from sequential five, so this is basically the last convolutional block, so that's five grand twelve channels with seven by seven to a 4096 dimensional output. That needs a lot of memory.

one way to address this is to get rid of those fully connected last layers. No you don't say!!

well that sounds quite easy in theory but how do you do it in practice?

the problem is convolutions and pooling reduce the resolution so you keep on having things but then at some point you still need to map to this you know number of classes dimensional object and at the same time you also need to have some degree of non-linearity to mix and transfer the information between the various channels into a representation that works for the number of classes.

this is exactly where one by one convolutions come in handy.

they only act per pixel on all the channels so this is a multi-layer perceptron applied to pixel wise activations.

so if in the end you only have maybe a five by five activation wide resolution then this really allows you to get a large amount of per channel information quite nicely presented in the formyou need.
That will tell you how many classes you need.

in the very last layer essentially they just got rid of it so rather than bothering with one last dense layer they just perform global average pooling.

so if you look at it again, well, the one by one convolution is just a multi-layer perceptron and that's what address the issue. This led to something called an in block networking Network block.

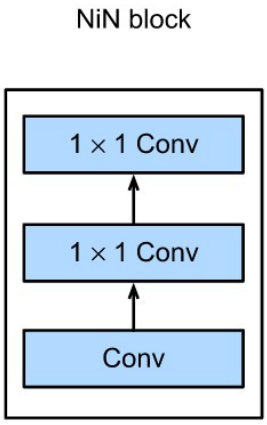
so you basically have a convolution followed by two one by one convolutions

These act in the same way as you would have a dense layer.

ou repeat that a few times and so you get the following architecture

NiN block

- A convolutional layer
  - Kerel size, stride, and padding are hyper parameters
- Two 1x1 convolutions
  - 1 stride and no padding, share the same output channels as first layer
  - These act like dense layers



https://d2l.ai/chapter_convolutional-modern/nin.html

so the new net had a number of you know convolutions followed by one by one convolutions max pooling in order to reduce the resolution and you did that three times in order to get a meaningful network in networks network.

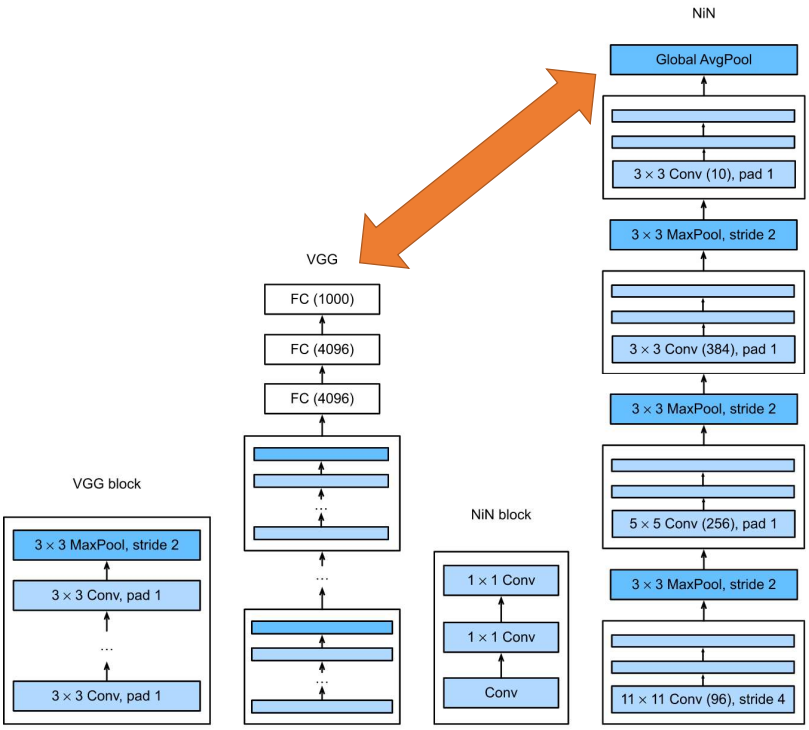
and this completely got rid of the dense layer, which was the contribution of network in networks

they didn't perform quite as well as vgg which is one of the reasons why people initially overlooked it

But they were a key component in order to go to things like Inception or ResNet which we're going to cover now.

NiN

- Final dense layers get replaced by ‘internal’ quasi dense layers
- Mapping to number of output classes is done via globalAveragePooling

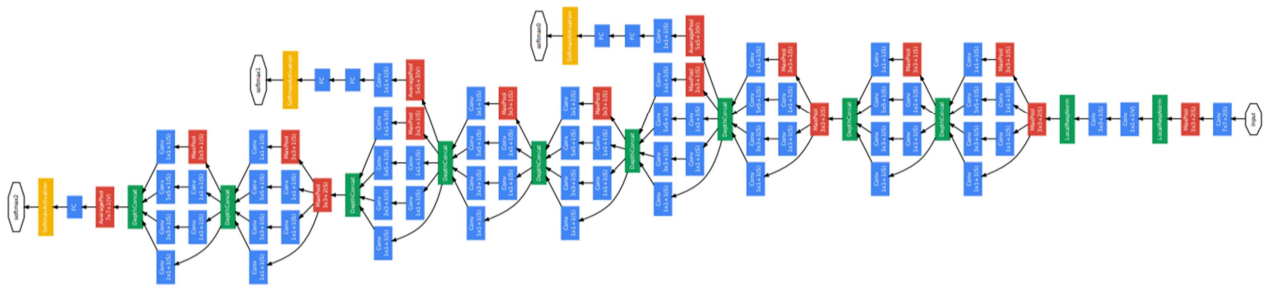


NiN in pytorch

- Inception and ResNet superseded this approach

```
from torch import nn

class NIN(nn.Module):
    def __init__(self, num_classes):
        super(NIN, self).__init__()
        self.num_classes = num_classes

        self.features = nn.Sequential(
            nn.Conv2d(3, 192, 5, padding=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 160, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(160, 96, 1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, stride=2, ceil_mode=True),
            nn.Dropout(inplace=True),

            nn.Conv2d(96, 192, 5, padding=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(3, stride=2, ceil_mode=True),
            nn.Dropout(inplace=True),

            nn.Conv2d(192, 192, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 192, 1),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, self.num_classes, 1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(8, stride=1)
        )
        self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), self.num_classes)
        return x

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                m.weight.data.normal_(0, 0.05)
                if m.bias is not None:
                    m.bias.data.zero_()
```

Inception (GoogLeNet)

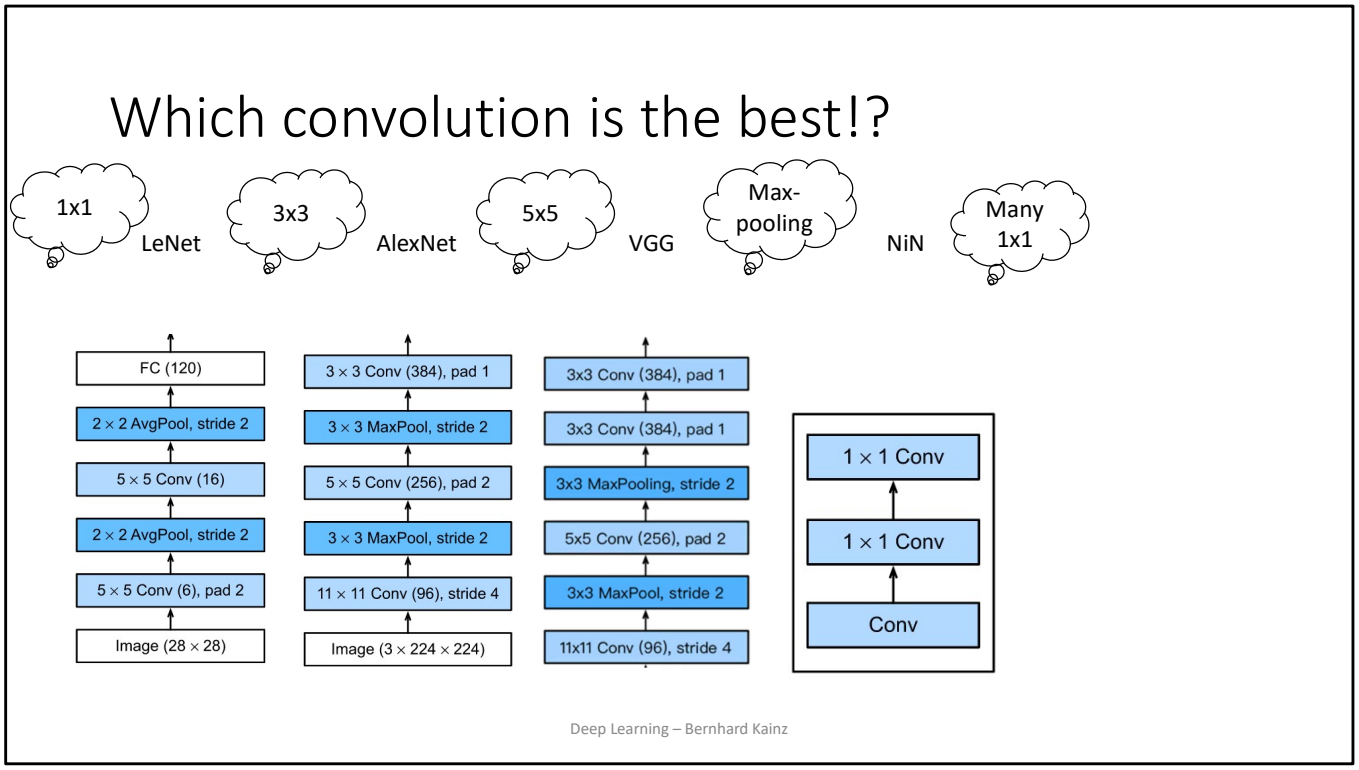


Deep Learning – Bernhard Kainz

https://arxiv.org/abs/1409.4842

So now we will look into a quite popular network Inception.

This is how the authors from google outlined their network. It is a) deep and b) this is the first time that you see parallel paths through a network.



let's review a little bit what we did so far

On the right so we have LeNet and this one has 5x5 convolutions and

then AlexNet here used 11 by 11 3 by 3 and 5 by f5 convolutions

Then vgg used some other mix of tools and then it used mostly one by ones

Well, this is a mess right

We do not know which convolution should we use, right?

the 1x1 the 3x3 is the 5x5 or max pooling or multiple one by ones and yeah I mean you can't decide right?

so this was essentially the dilemma that people back then in the deep learning stone age faced in trying to figure out how to build a good convolutions block

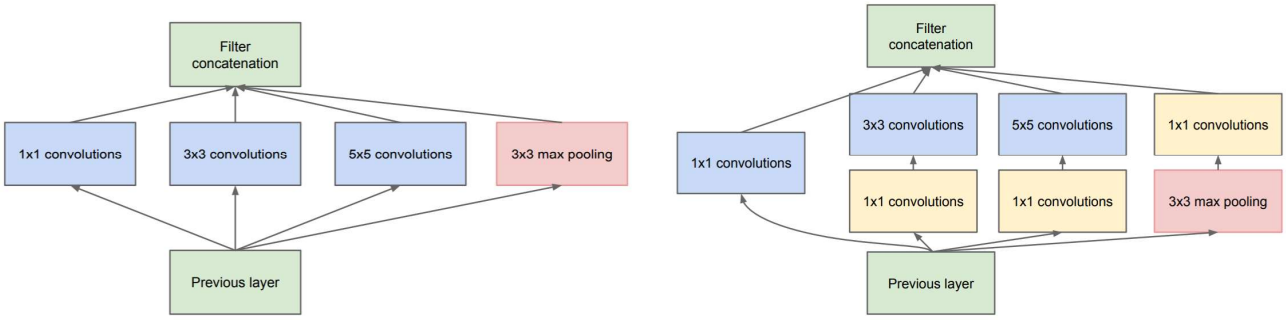
the issue is the following: if you pick 5x5's you end up with a fair number of parameters and end up having to do a lot of multiply-adds and both of those things are really costly and make networks slow. A lot of parameters also means it won't generalize that well

but it's on the other hand very it is expressive

On the other side, if you pick 1x1 convolutions then it's very well controlled and it doesn't need much memory area but at the same time well it maybe doesn't work so well.

so what are you gonna do?

Inception block



https://arxiv.org/abs/1409.4842

The solution was just not to decide.

that was the brilliant idea when this was published and yeah it's also called the inception network after the movie.
like we need to go deeper and that was literally their motivation for coming up with that name

so here's the inception block and it's really the well we don’t know what to do to well do all of it

it has 1x1 convolutions because yeah why not

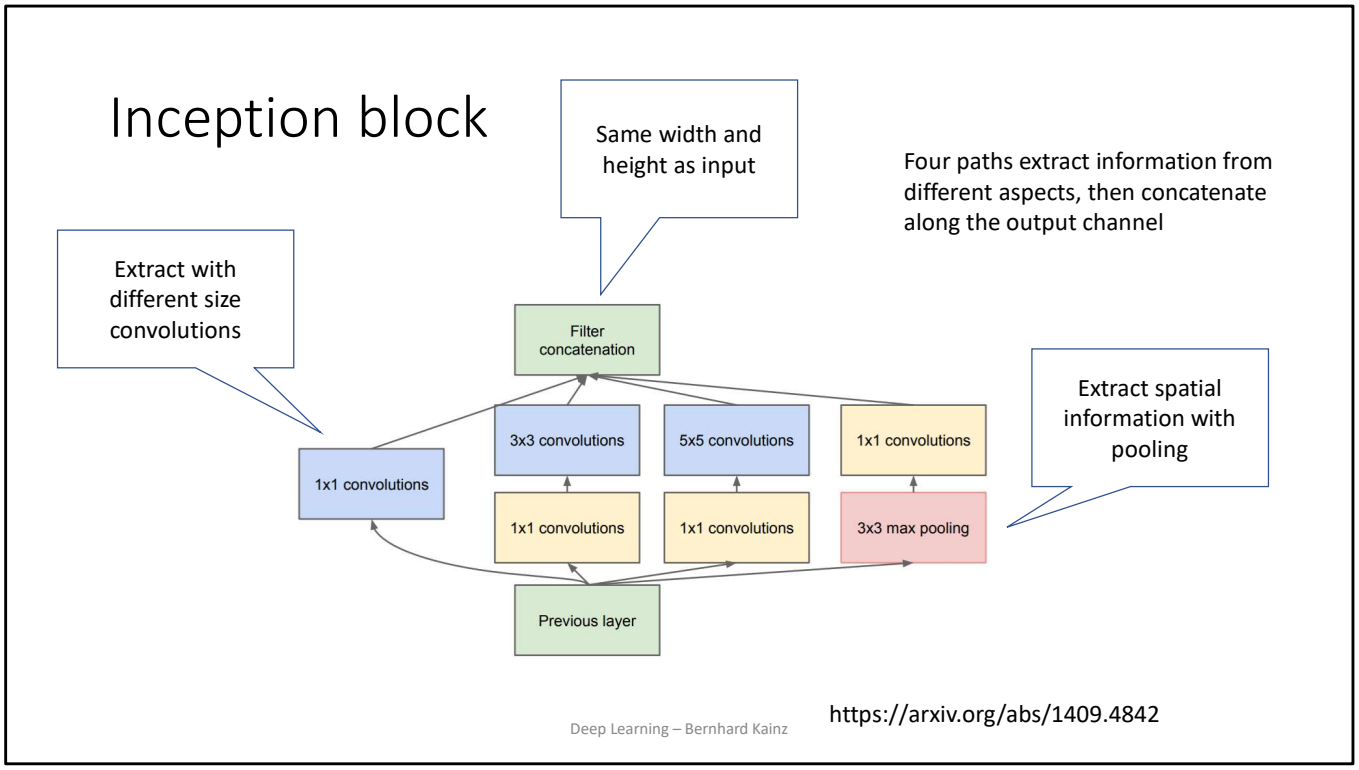
it has 1x1 convolutions followed by 3x3 convolutions

it has 1 by 1 followed by 5 by 5

it has max pooling followed by 1 by 1 convolutions

We just combine this all and the hope is well if you throw it all on the wall something will stick.

On the left they proposed a simple block and on the right they propose a more complex one also including 1x1 convolutions.



now to make sure that this all has the same dimensionality you need to use the appropriate padding so that's why there are 3x3 half padding by one 5x5 and half-padding by two .

so at least in terms of sizing the inputs and the outputs have the same size.

and then you just stack it all together.

so you now have an architecture with different channels doing different things and the hope is one of those channels is gonna work for the cats and one is gonna work for I some birds and so on.

This is the inception block.

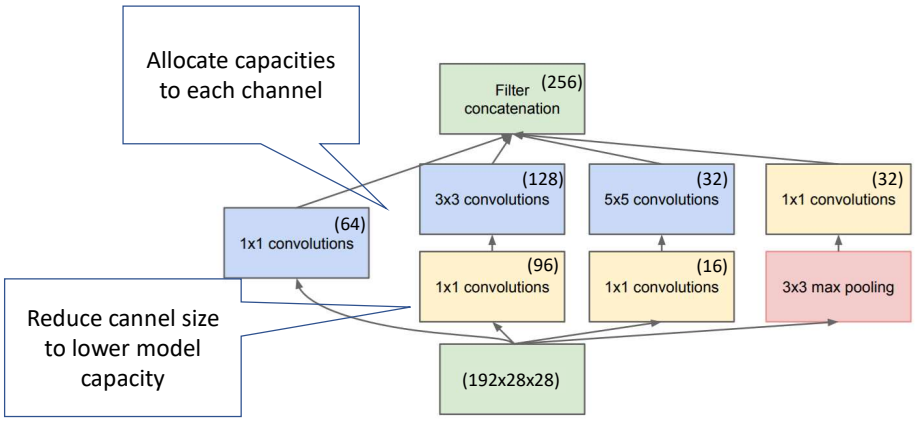
now you might wonder why on earth is this specific block a good idea?

well they probably tried out different variations and this one worked best

for now let's just assume it's the well we tried out a lot of

Inception block

The first inception block has channel sizes specified



Deep Learning – Bernhard Kainz

https://arxiv.org/abs/1409.4842

so for instance if you look at the first inception block well it uses 64 channels for the one by ones
128 for the three by threes

thirty-two for the five by five because they have a lot of parameters already anyway right so it's 25 times 32, in the other case it's 9 times 128 right and then you have

just a few other max pooling dimensions thrown in because well you want this all to add up to 256 that's really what it is.

right so there isn't anything terribly deep in there

the input input number of channels doesn't really matter that much

it's just features in right channels in and then you get some appropriate number of channels out

Inception blocks

- Inception blocks have fewer parameters and less computation complexity than single 3x3 or 5x5 convolution layers
- They are a mix of different functions, which makes them a powerful function class
- Computing and memory wise they are efficient (good generalisation)

	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M

As: replace all conv block with 3x3 or 5x5 in Inception

but what it does, is that you now have a number of parameters, a number of floating point operations

That are actually not higher than doing something a lot more simple

so that's really the key benefit of that

so if you do a parameter count and you say well if I wanted to have 256 output layers well 256 output channels then with inception you only need 160,000

parameters and it costs 128 mega flops.

whereas the 3 by 3 would cost you about three times that and a five by five about eight times that amount and in terms of floating-point operation.

so the assumption still is you know if I can get the same thing done with fewer parameters then they work better and yeah that's essentially what

motivated the inception block.

so why is it simpler?

let's just look look at the algebra

Less parameters?

- $k^2 \times \boxed{c_{in}} \times c_{out} \times \boxed{m_h \times m_w}$

  fixed fixed

- $c_{in} \times m_h \times m_w \times [\sum_{path\ \ j} k_j^2 \times c_{out,j}]$

allocating compute to different channels = better computing

Deep Learning – Bernhard Kainz

let's just look look at the algebra

right so for a particular layer I have K by K so since they are like three by three or five four five.

I'll just write K squared times C in times C out times then I have basically MH x MW

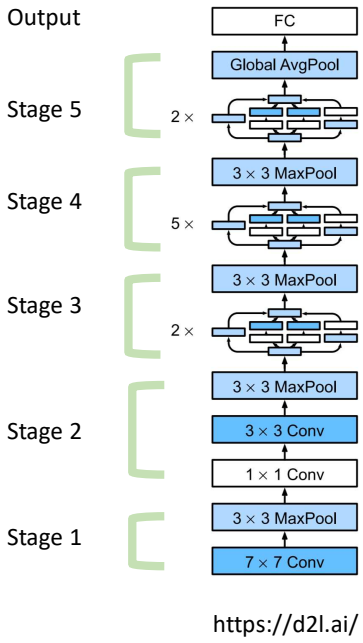
well this one's fixed this one's fixed well that one to some extent I can play with

so now what you get is you know CI x MH x MW x

now a sum over the various paths j KJ squared times C out J and so by judiciously allocating compute to different channels different numbers of channels for different kernel width
you can end up in a better spot than picking something homogeneously with the same depth.

Inception

- 5 stages with 9 inception blocks



https://d2l.ai/

okay so here's the network in it's full beauty

this representation from the d2l book is a little bit easier to understand than from the original paper

these are really the five stages of that inception network.

stage one behaves very similar to a lot of other convnets and it starts with a fairly broad convolution and pooling which just make sure that we have some basic

amount of translation invariance and that I am able to reduce the dimensionality reasonably quickly

fairly early on -- that max pooling halves my resolution

then stage two is again very much trying to get some overall spatial correlation and then some pooling operation in the end.

This is fairly vanilla relative to other networks we have seen.

and now is where all the interesting things start happening because we now have those

inception blocks

two of them then max pooling which again reduces resolution

so the 3x3 max pooling each of those operations half my resolution

so what I'm doing is I'm shrinking resolution but I'm also increasing the number of channels because now while I have fewer pixels they have more valuable more higher-order information that I'm going to use later on.

then I have backbone of five inception blocks one after the other

this is where most of the interesting nonlinearities happen

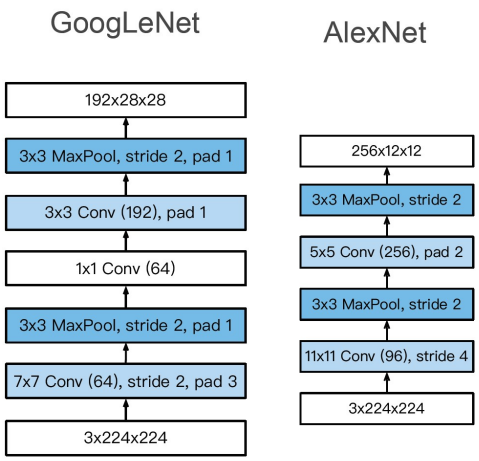
they shrink again and then I'm picking another two

why did they do a split of two five and two? well they probably tried out a whole bunch of networks and that's the best thing that they could come up with

there are some follow-up papers like AmoebaNet which is then beyond Googlenet version 4 where they essentially use a genetic algorithm to just randomly try out whatever stuff works and if you have a lot of computers available and cost is no object neither for getting there for running the computation then you can get really good accuracies with that right anyway.

Stage 1 and 2

- Smaller kernel size and output channels because of more layers



https://d2l.ai/

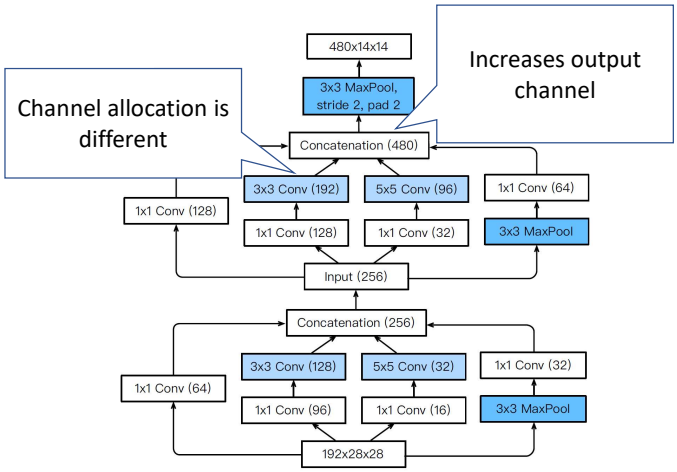
let's look at stage one and two directly so in an Alexnet it’s basically a very wide convolution 11 by 11 and this one's a little bit smaller

it's only a seven by seven right and you have four one by one and then again three by three and pooling so it's not too dissimilar from Alexnet

the only difference is that you have a bit more channels

so in AlexNet ended up with a 12 by 12 in the end this is 28 by 28 so it's still keeping a fair amount of the information that you would have had otherwise

Stage 3

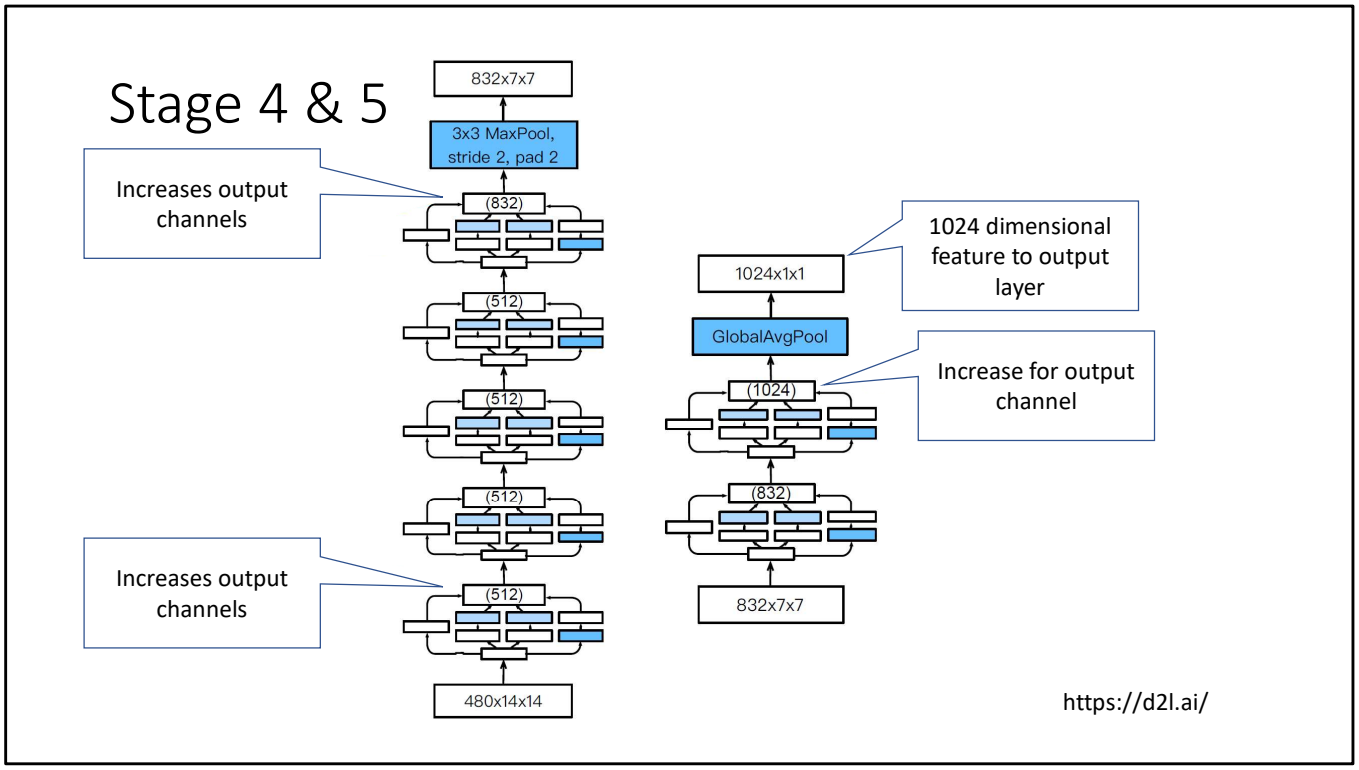


https://d2l.ai/

stage 3 well a fair amount of stuff is already happening and so the allocation is different between channels even between the first and the second block

so you can see for the first the first block you have 256 channels and the second one you have 480 then you actually go and reduce the resolution so you start out with 28 by 28 then the end you get 14 by 14

the different versions of those networks are usually due to different size trade-offs and whether you use 2 3 4 5 stages or blocks



stage 4 and 5 look very similar

again you increase the number of channels to 512 up to 832

Why that before the max pooling? because the max pooling shrinks the resolution so we better store as much information as we possibly can on a per channel basis

and then in the end you have basically a 1024 channels and that just so happens to be the same thing as what you would get for the number of classes that you want to

predict

and then you just perform a global average pooling

now this is a great idea to just do global average pooling over a 7x7

Flavours of Inception Networks

- Inception-BN (v2) – added batch normalisation
- Inception-V3 – Modified the inception block
  - Replace 5x5 by multiple 3x3 convolutions
  - Replace 5x5 by 1x7 and 7x1 convolutions
  - Replace 3x3 by 1x3 and 3x1 convolutions
  - Generally deeper stack
- Inception-V4 – adds residual connections

the thing is of course there isn't just GoogleNet there's GoogleNet v2 and v3 and v4 and those different variants are then improvements on the overall

architectural pattern so we add something called batch normalization and we'll cover that a little bit later then

v3 adds different shapes of convolutions so this is you know going even crazier on the well do we need three by threes

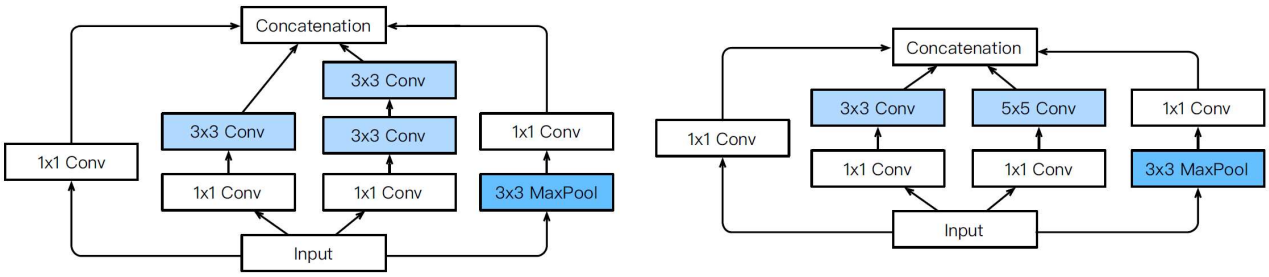
one by one so 5 by fives this you could also add you know a 1 by 5 or a 5 by 1 or 1 by 7 or you know some other shapes

and overall making each of those blocks even deeper and this is exactly where if you didn't you know try out even more

things you might end up with something like Amoebanet which has a lot of different ones

an inception v4 then copies from ResNet and essentially imports the ResNet ideas into Inception but still it doesn't work as well as ResNet directly

Inception V3 block for stage 3



https://d2l.ai/

so let's look at what changed

the left-hand side is version 3 for stage 3 and if you compare left and right you can see well the right-hand

side replaced its five by five convolution by two three by threes

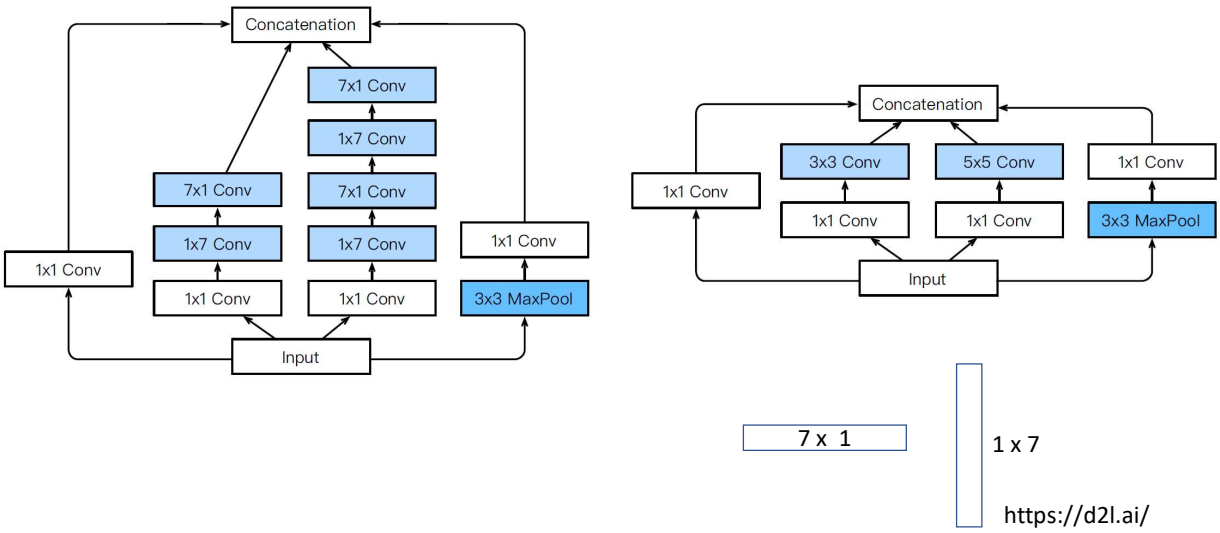
So that's actually what you would expect

So if you remember the Simonian Zissermann paper where they looked at whether wide and shallow or deep and narrow networks are better

deep and narrow won and so that's why they replace the five by five by two three by threes

that's a fairly benign change

Inception V3 block for stage 4



now let's look at stage 4 and this is where they where the big changes were made

again it's the five by five that were replaced and replaced by 1 by 7s and 7 by ones

this is actually the first time that people then used asymmetric convolutional shapes for real

7 by 1 is quite narrow right but it requires only 7 parameters per channel

it's actually cheaper to do this than 3x3s

and so this increased computational budget

the 5 by 5 that's 25 parameters

those four layers of seven by one and one by seven convolutions that are alternated cost you twenty eight parameters

This means that the cost is very similar

you have a very similar model class in terms of number of parameters but it's much

more expressive

So why do I need to combine a 1 by 7 with a 7 by one convolution?

why can't I just get away with it and say well I have a 1 by 7 convolution that's it

why do I need both?

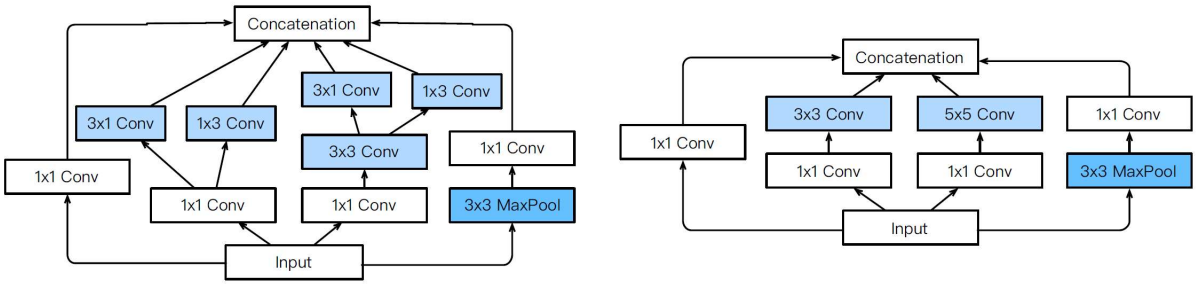
If I'm only going to use on of these then you might only get for example vertically contiguous features in some way

The 1x7 will catch only features that are horizontally contiguous in some way

and so if you only include one of them you’re going to get a network that's particularly well-suited for one type of features but awful for the other

and in general I won't want that unless this is actually my goal in the network design

Inception V3 block for stage 5



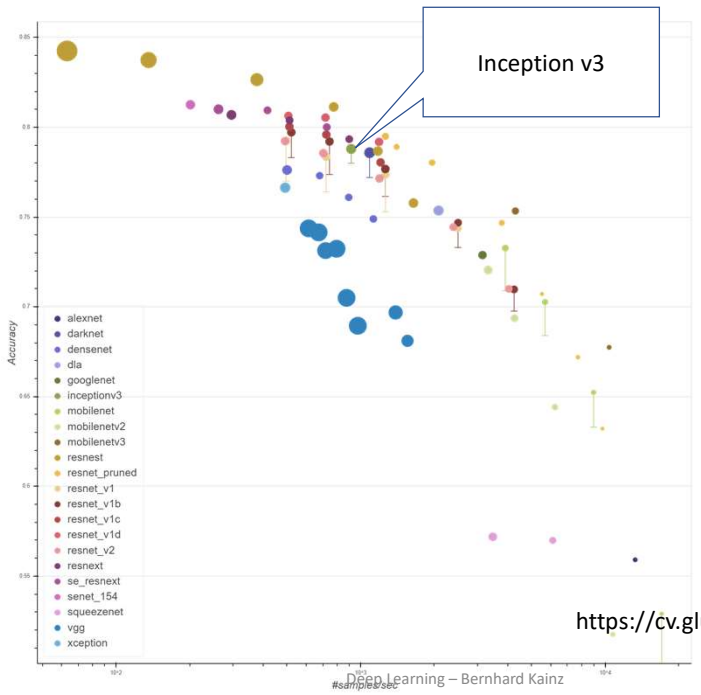
https://d2l.ai/

Stage five went even crazier

so rather than those three by threes it used one by three and three by ones and likewise in the other part

okay so I guess you can see a pattern of what the modification steps were

with that we are at the point where we are now getting closer to what's actually state-of-the-art performance



https://cv.gluon.ai/model_zoo/classification.html

this is a plot of accuracy versus throughput for a fairly large number of models

this is from the gluon model Zoo and what you're seeing there is a large number of architectures implemented and

trained in the same way the size of the dot corresponds to the amount of memory footprint that is needed in order to execute a particular model

so small dots are very efficient models and higher up are more accurate.

The lines show the reproduction by other researchers. This is the difference between the published results and reproduction. So lot's of results are actually due to better training.

Interpolate… ensembles etc.

What do we learn from that?

- Dense layers are computationally and memory intensive. Real-world problems with big input tensors and many classes will prohibit their use.
- Again: 1x1 convolutions act like a multi-layer perceptron per pixel.
- Scientists are humans and need a while to understand the power of new approaches. Eventually they do but a lot of vanity is involved in the process.
- If not sure, just take all options and let the optimization decide or even learn this through trial and error (genetic algorithm, AmoebaNet)