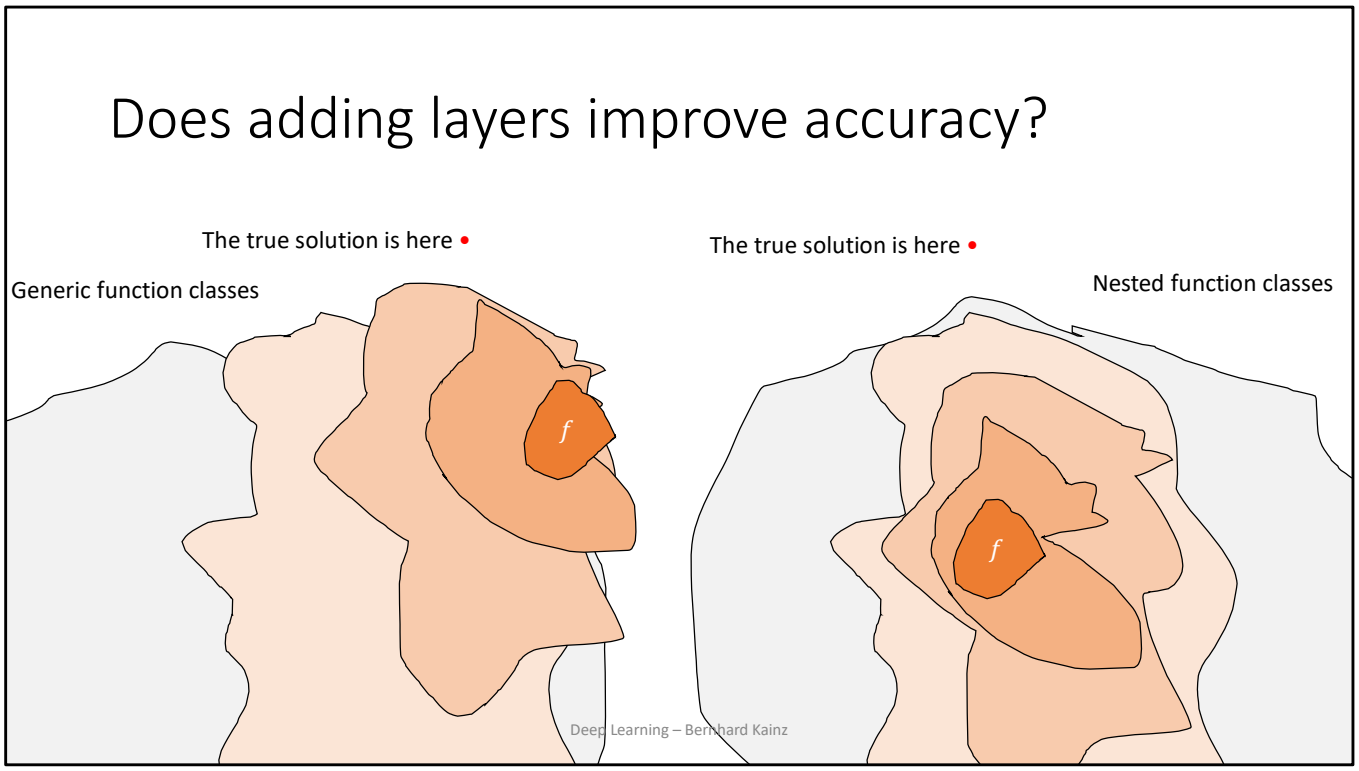
Deep Learning - ResNet

Bernhard Kainz



so let's talk about ResNet

if you had to pick a worry-free off-the-shelf Network I would probably recommend that you go and pick a ResNet

there's a reason why they worked significantly better and again the initial motivation was bit strange and about gradients not making it through the entire stack

Let's say we have a deep network and then we decide we'll go and make it even deeper

so we can make it deeper by just by adding another layer and by adding another layer we get a slightly different function class

if we add another layer we get again a slightly different function

We have more parameters so we can do more interesting and powerful things.

but basically my function classes get bigger and more powerful but they're all so different

so if you look at the picture on the left you can see a set of function classes each of

which is a little bit bigger than the previous one so that's a little bit more powerful but they're all a little bit different

now let's assume that the truth is out there

this red dot is the truth and I want to approximate that truth

as you can see by adding more layers and layers you're going to get closer and closer to that red dot and once you've reached about that third network you can get kind of closest to the truth

and then as you start increasing the function class further you start moving away again and that's really undesirable

there's no guarantee that you might not eventually also come back again.

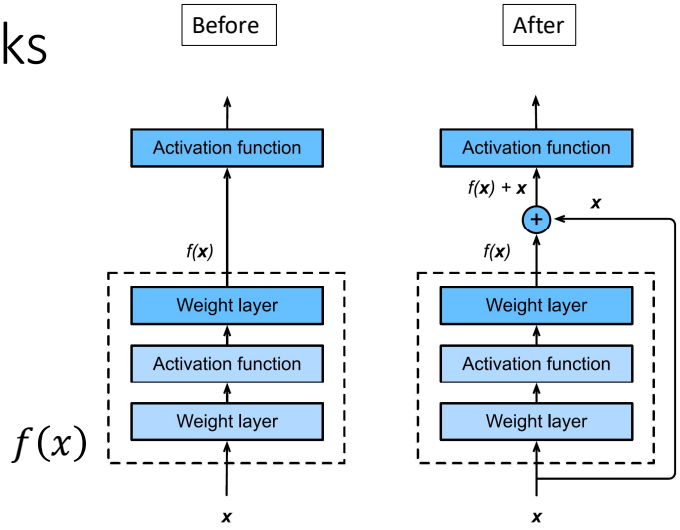
And this is awkward, because this makes it very hard to reason in some meaningful engineering way over how many layers we should pick?
how complex should my function class be?
Now, on the other hand, if you look at the picture on the right, things are much nicer.
And typically, if you open any paper, on statistical learning theory, they pretty much all assume that you have a nice, functional class.

Unfortunately, with deep networks we have the situation on the left,
so how can we make it a little bit closer to the one on the right.
The function class still get larger, as a matter of fact, I drew that right plot by just centering it.
So it's similarly powerful function classes, it's just that they're nested.
So as I move from one to the other, they become more powerful, more expressive, I can reach a larger function class
And so now, as I keep on adding, I move closer to the truth in the red dot.

They're not necessarily convex, but, that's just what it is. So, I want to have the situation on the right, how do I get there?

Residual Networks

- Adding a layer changes function class
- We want to add to the function class
- 'Taylor expansion' style parametrization



He et al. 2015 https://arxiv.org/abs/1512.03385

https://d2l.ai/

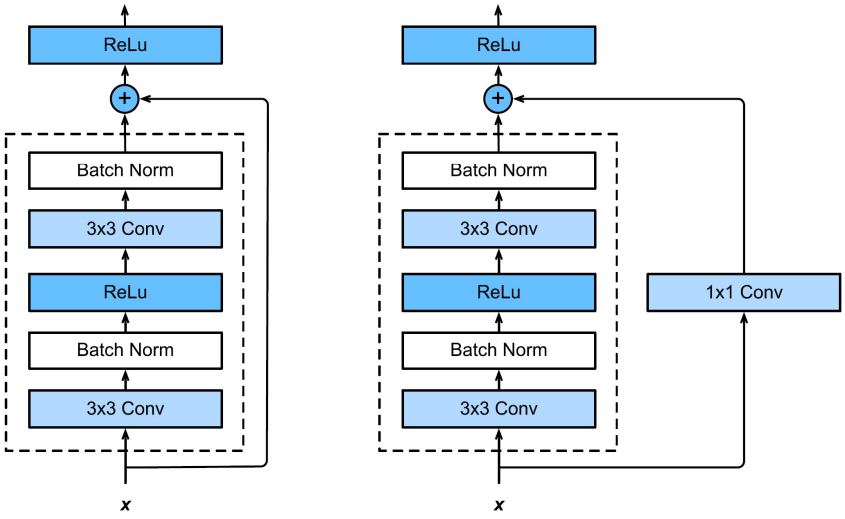
And this was the ingenious idea by He et al 2015. Namely, rather than parameterizing around the function, f of x equals zero being the simplest function, and that's basically for the weights of zero, I parameterize, around the function f of x equals x.

Because if all the parameters and everything is zero, if I add the input to the output, then I get the identity function as the simplest function.
That's quite brilliant. So it's like a Taylor expansion for neural networks!

At least a similar flavour.
So now as you increase your parameters, you're starting to deviate from the identity function.
We don't have to learn the identity function that does nothing to my data anymore.
So it's essentially assuming a different inductive bias,
it also means that as I add another layer, the identity function still goes through and it will still leave the outputs of the previous layers unchanged.
just that we could now do additional things to my output.
Of course, it's not exactly how it works. So they didn't exactly implemented those nested function classes.

But ResNet did something that was pretty close to that. In the picture there is the ResNet block on the right. You add the input to the output residual.

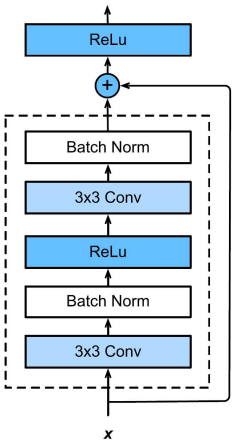
ResNet Block



https://d2l.ai/

It has convolutions and a batch norm, rectified linear unit, another convolution another batch norm.
And to that, you then add the input.
Or you run a one by one convolution on the input, and you add that to the output.
But at the very least, you're making it fairly easy for the network not to change things, if not changing things is the best thing to do.
This is now skipping a lot of details of setting the variables, this is just the forward function.
And what I get is y, is batch norm one, of convolution one of x. Then we apply relu, then we apply another batch norm two have y two.
And then if we are in the situation where we have the path on the right here, then we just apply conv 3 to X and add the two results together.
Of course, I need to pick my convolutions and everything in such a way that the dimensionality still match up.
But we already saw how to do that, for inception. Now we can use it for batch norm followed by a basic relu block.

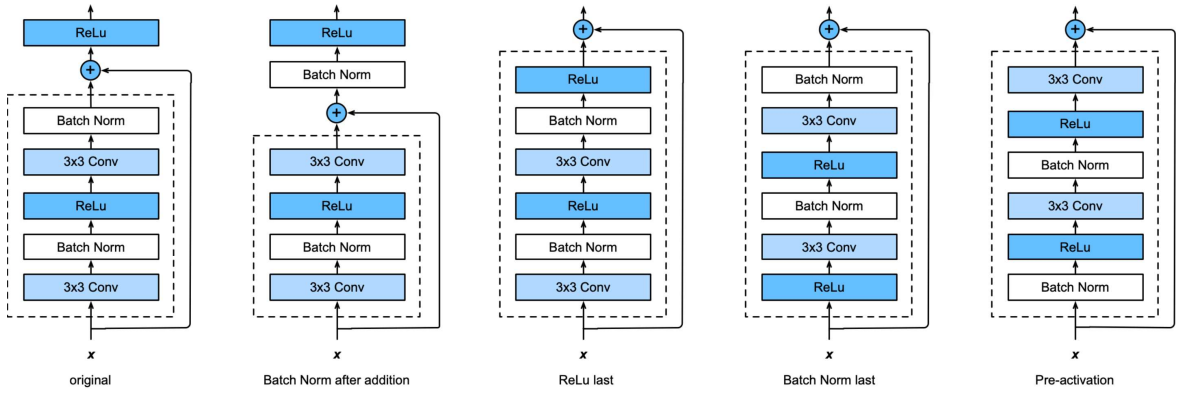
ResNet Block




```
def forward(self, X):
    Y = self.bn1(self.conv1(X))
    Y = nd.relu(Y)
    Y = self.bn2(self.conv2(Y))
    if self.conv3:
        X = self.conv3(X)
    return nd.relu(Y + X)
```


This is how this looks approximately in code

ResNet block flavours



Trial and error of every permutation

https://d2l.ai/

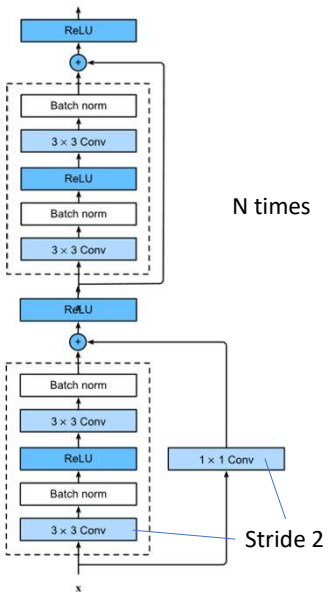
people tried all those different flavours of ResNet.

they try and you know, where to place the batch norm and where they place the addition.
So sometimes, and in some papers, they use the batch norm after the addition.
In some cases, they used the addition at the very end.
In some cases, they permute the order of, RELU, convolutions and batch norm, and so they tried it out all three different permutations.
So the sequence between the blocks is the same, but the order is changed.
For some combinations, some days, in some settings, some of those things work better.

Now, this sounds thoroughly unsatisfactory, but this is unfortunately, the awful truth of deep learning training.
In some cases, you just have to try out things. Then at some point, somebody does this figures out, okay, this is a good architecture, and then people pretty much stick with it.

ResNet Module

- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks



https://d2l.ai/

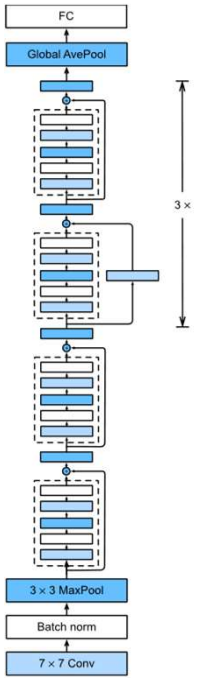
Okay, so here's the resnet module.
So this looks, quite familiar to what we saw before, just that now, it's using resnet blocks rather than Inception blocks.
So you might have one of those blocks, which downsamples, so stride two, and then you have a couple of the other ResNet blocks.
And that works well.

ResNet

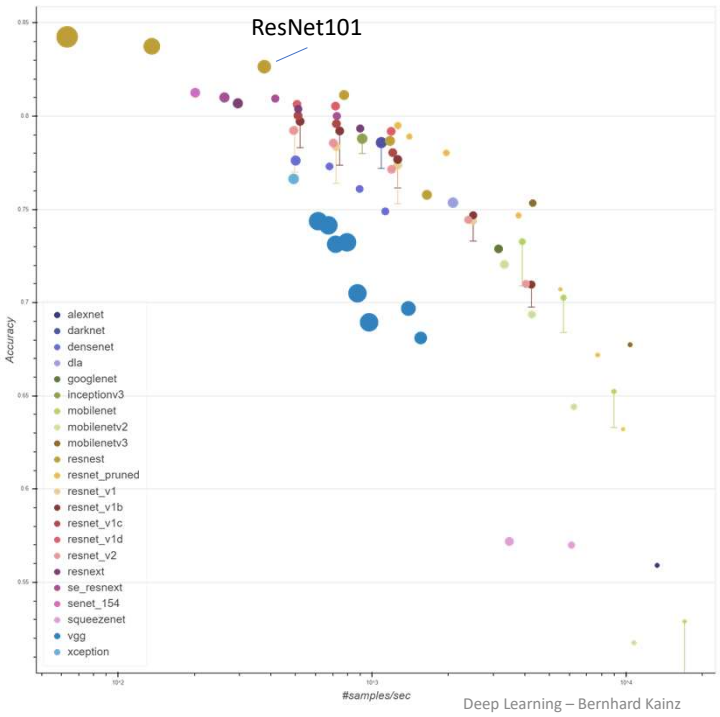
Same block structure as e.g. VGG or GoogleNet

- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

- Trainable at scale
- Variant name depends on how many blocks (18 layers = ResNet-18 ->)



Here's the entire network in its full glory. That is resnet 18 and it goes up all the way to 200+. So that's 200+ layers.
The bottom of the network looks very similar to whatever we saw before with convolution, batch norm pooling.

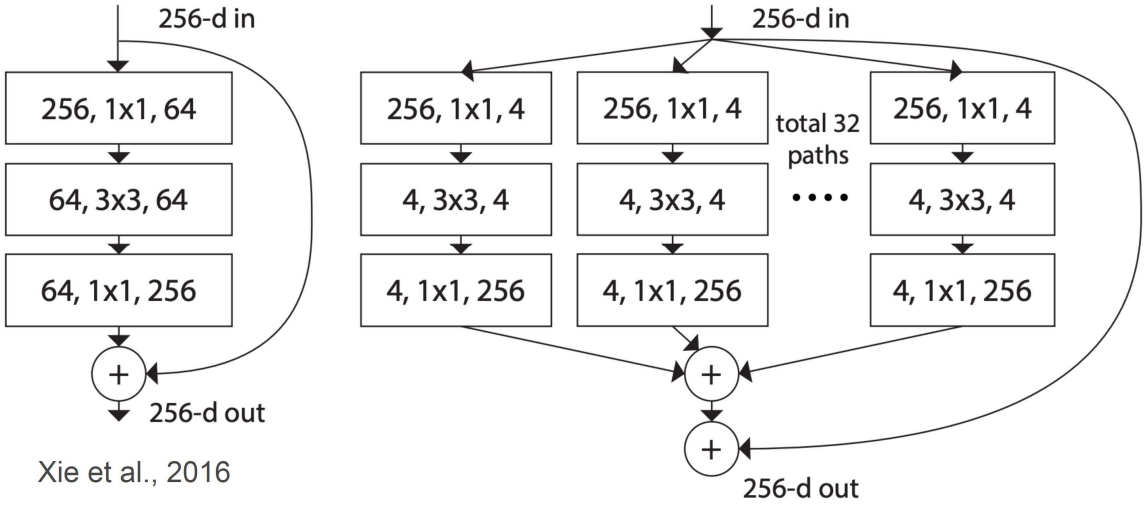


https://cv.gluon.ai/model_zoo/classification.html

What this gets you to is this dot up here, this is now resident 101 is pretty much state of the art. It's slower, it's much more accurate. It's actually smaller in terms of footprint and In the inception network that we saw before, there would have been the orange circle below.

ResNext

https://arxiv.org/abs/1611.05431



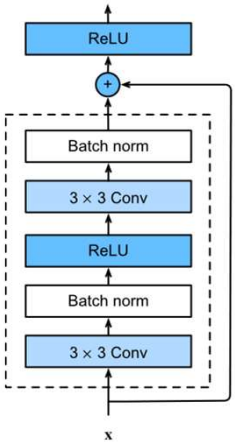
Deep Learning – Bernhard Kainz

People pushed this idea of course further and Xie et al introduced ResNEXT.
And it has a neat idea in it.
Essentially, it relies on the following observation. If this is our resnet block, right, so we have a one by one and maybe a three by three, then we have a one by one.
So we have 64 channels, 64 channels, and then we have 256 channels, right?
Or some choice like this.
So you might ask yourself, well, is this a really good idea.
And it turns out that you may be able to do better by just partitioning things, such that the inter channel interactions now happen only by groups of, let's say, four.
So what we are doing is basically taking the network on the left, and slicing it up into 16 or more networks that just have four channels, or some larger number of channels each.
And then in the end, we stack everything together.

And so we have height times width, times input, times output channel parameters.
And the computation is output height and width, times the corresponding height and width of the kernel plus input times input times output channels.

We can break things up by one by threes, and one by five and five by one.
So, it's what we saw before for inception.
Or we can break up the channels.

Reducing the cost of convolutions



- Parameters
$$k_h \times k_w \times c_i \times c_o$$
- Computation
$m_h \times m_w \times k_h \times k_w \times c_i \times c_o$
- Slicing convolutions (Inception v4) e.g. 3x3 vs. 1x5 and 5x1
- Break up channels (mix only within)
$$m_h \times m_w \times k_h \times k_w \times \boxed{\frac{c_i}{b} \times \frac{c_o}{b} \times b}$$

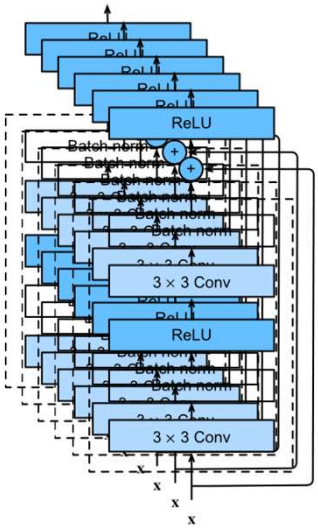
https://d2l.ai/

And so if I break it up into B groups that have ci over B, groups as the inputs, the output of B groups for the output times B, is going to have B groups.

So in other words, I get a reduction by a factor of one over B.
And as a result, I can use more dimensions and capacity for the model.

So this is very easy to add to any code that you might have.

Reducing the cost of convolutions



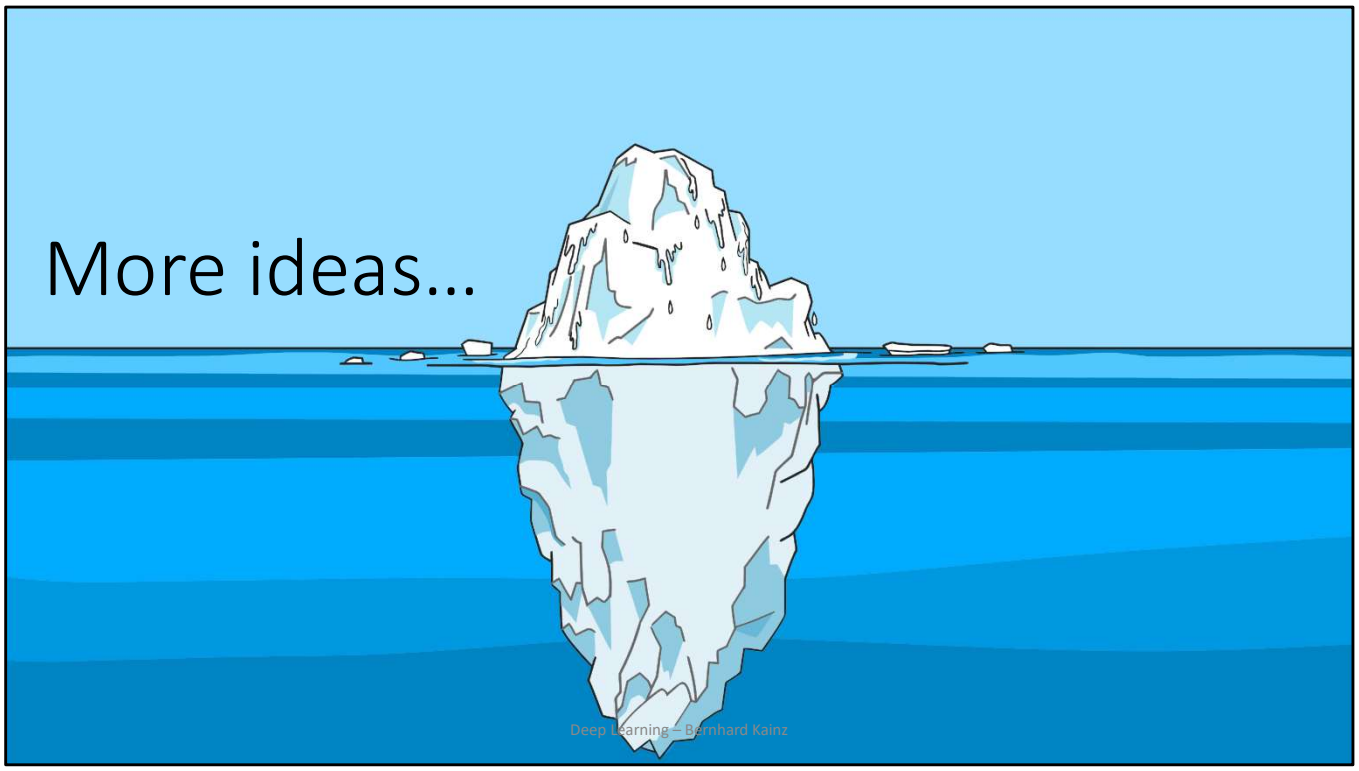
- Parameters
$$k_h \times k_w \times c_i \times c_o$$
- Computation
$m_h \times m_w \times k_h \times k_w \times c_i \times c_o$
- Slicing convolutions (Inception v4) e.g. 3x3 vs. 1x5 and 5x1
- Break up channels (mix only within)
$$m_h \times m_w \times k_h \times k_w \times \boxed{\frac{c_i}{b} \times \frac{c_o}{b} \times b}$$

https://d2l.ai/

Basically, what you do is you just set in the 2d convolution, the variable groups to whatever number of groups that you want. That's it.

Now, if you compare ResNet and REsNEXT, the computational budget is very, very close.

Number of parameters are within few percent of each other,
number of flops is very close, but it works a little bit better.



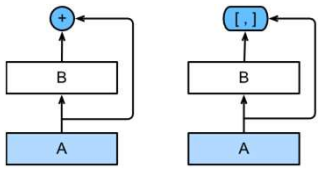
Well, an obvious thing that you can ask yourself is what comes after resnet? Right?
So let's look at the zoo of more ideas. And this is really just going to give you, you know, the tip of the iceberg.

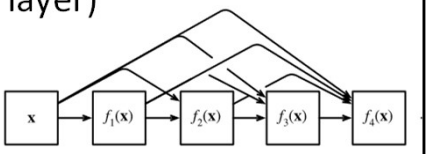
DenseNet

- Huang et al., 2016 https://arxiv.org/abs/1608.06993
- ResNet combines x and $f(x)$
- DenseNet uses higher order 'Taylor series' expansion

$$x_{i+1} = [x_i, f_i(x_i)]$$
$$x_1 = x$$
$$x_2 = [x, f_1(x)]$$
$$x_3 = [x, f_1(x), f_2([x, f_1(x)])]$$

- Occasionally need to reduce resolution (transition layer)





https://d2l.ai/

ResNet went from parameterizing f of x equals zero to f of x equals x.
This is a simple function.
But you can essentially get like a higher order Taylor series type expansion.
So what you do is basically, you define x_i plus one, to be the concatenation of x_i, and a phi of xi.
So, as a result, that vector will keep on growing, and it'll basically have increasingly higher order terms in the x expansion.
They start with x_1 is x, x_2 is x and so on.

Basically, x_3, x_4 would be this term plus f of all of the previous.

Actually, quite surprisingly, if you train resnet really well, or resnext really well, it outperforms dense net.

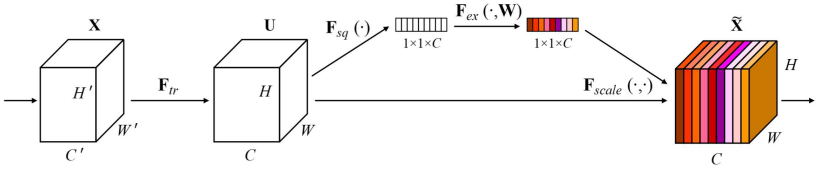
So how did they manage to do really well?

Well, their training implementation was better.
So sometimes, it's not the network, but how you train it that lets you win benchmarks.
This is the thing that isn't always clear to people when they read papers.

Okay, so densenet, it's kind of useful, but not that much.

Squeeze-Excite Net

- Hu et al., 2017 https://arxiv.org/abs/1709.01507



- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

Here's one that's actually a little bit more exciting.
It's called squeeze-excite or SE-NET.
And this uses something called attention.
So attention is essentially a mechanism where, rather than taking averages over a bunch of vectors, we're using a separate function to gate how that average should be computed.

What squeeze excit does is if you think about the various channels, maybe there's a cat channel, and there's a dog channel, and maybe there's a dinosaur channel?
Now, if you knew that you're recognising a cat, well, what would you do?
You would weight that cat channel higher and you ignore the rest, right?
But that's kind of stupid, because how would you know that you are recognising a cat until we have actually recognised the cat?

Like, once I know the answer, the question becomes a lot easier.

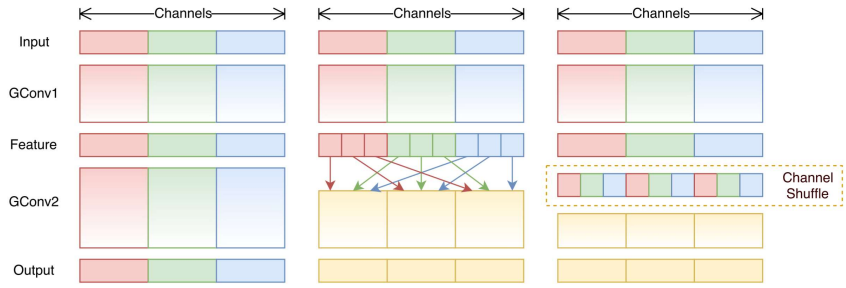
The other thing is that the information transfer that I have in convolutional networks is kind of slowish.
We have maybe a three by three convolution, another three by three, then we pool and so on.
So it can take like 4,5,6 layers until the information from this corner percolate to the

other corner.

Maybe if there is a bowl of milk here, the chances that there's a cat over there is much higher.
So I would know that from the context.
So my cat detector can use the fact that there's a bowl of milk to infer that there is a cat.
What you could actually do is you could take very simply a product of the entire image in a per channel basis, with some other vector.
And so now you get channel many numbers out of it.
So this is a very simple object. It's fairly cheap to do compared to all the convolutions and everything.
Finally you use those numbers and the softmax over them to rewrite your channels.
So therefore, if this very cheap procedure tells me, well, there's a good chance that there's a cat somewhere, I can now up weigh the cat channel.
So they're currently actually the best ones in the model zoo.

ShuffleNet

- Zhang et al., 2018 https://arxiv.org/abs/1707.01083



- ResNext breaks convolution into channels
- ShuffleNet mixes by grouping (very efficient for mobile)

Beyond ResNet SE-Net the question came up can we do something a little bit more structured, or, more sparse, structured with our networks.

ResNet breaks up the channels into subgroups, and then within each subgroup of the channels, you do your stuff, and then you need to combine.
Now, that's not necessarily very good, because you may end up getting those very long stovepipes, where the features only mix within each of those, but not across them.

You got rid of the cross channel mixing in order to get faster computation and everything.

Now, one way to bring it back is if you just go and reshuffle things in between convolutions.
And so in this case, if we have three channels, well, we go and basically pick, one from the red greens and blues and turn that into a new block.
And then I essentially intertwine things in a meaningful way.
That gives you a little bit more accuracy. So, shuffle net, is what you get out of that.

On a mobile phone you don't want to have a lot of computations and there this approach can be very efficent.
ResNet is one of those cases where you can get high accuracy for a small number of

comparatively small number of computations.

Things to explore

- AutoML (find best model architecture automatically Google Cloud AutoML)
- Hypernetworks (a network that proposes the weights for another network), also neural processes
- Networks with memory, e.g. kanerva machine
- Almost no new basic architectures accepted nowadays (see https://nips.cc/virtual/2020/public/cal_main.html NeurIPS 2020 programme, focuses on meta findings)

Summary

- **Inception**
  - Inhomogeneous mix of convolutions (varying depth)
  - Batch norm regularization
- **ResNet**
  - Taylor expansion of functions
  - ResNext decomposes convolutions
- **Model Zoo**
  - DenseNet, ShuffleNet, Separable Convolutions, …

https://in.pinterest.com/pin/556124253981912489/

So to summarise a little bit, we talked about inception, and resnet. And the key point in Inception was essentially that you can mix and match different types of convolutions and you can use batch norms.
Resnet usees this idea of a Taylor expansion and decomposes convolution.
So it's basically separable convolutions, but with a bit more control.
And then there's this entire slew of additional things that you can do.
SE-Net and shuffle net are the more interesting parts there.

What do we learn from that

- Deeper is not necessarily better if the function space is not regularised
- ResNet is the workhorse of Deep learning (for now. Do you have a better idea that hasn't been tried yet? Let me know but look on arXiv first!)
- Lot's of variations have been proposed but it often boils down to how you train a network and for what purpose.