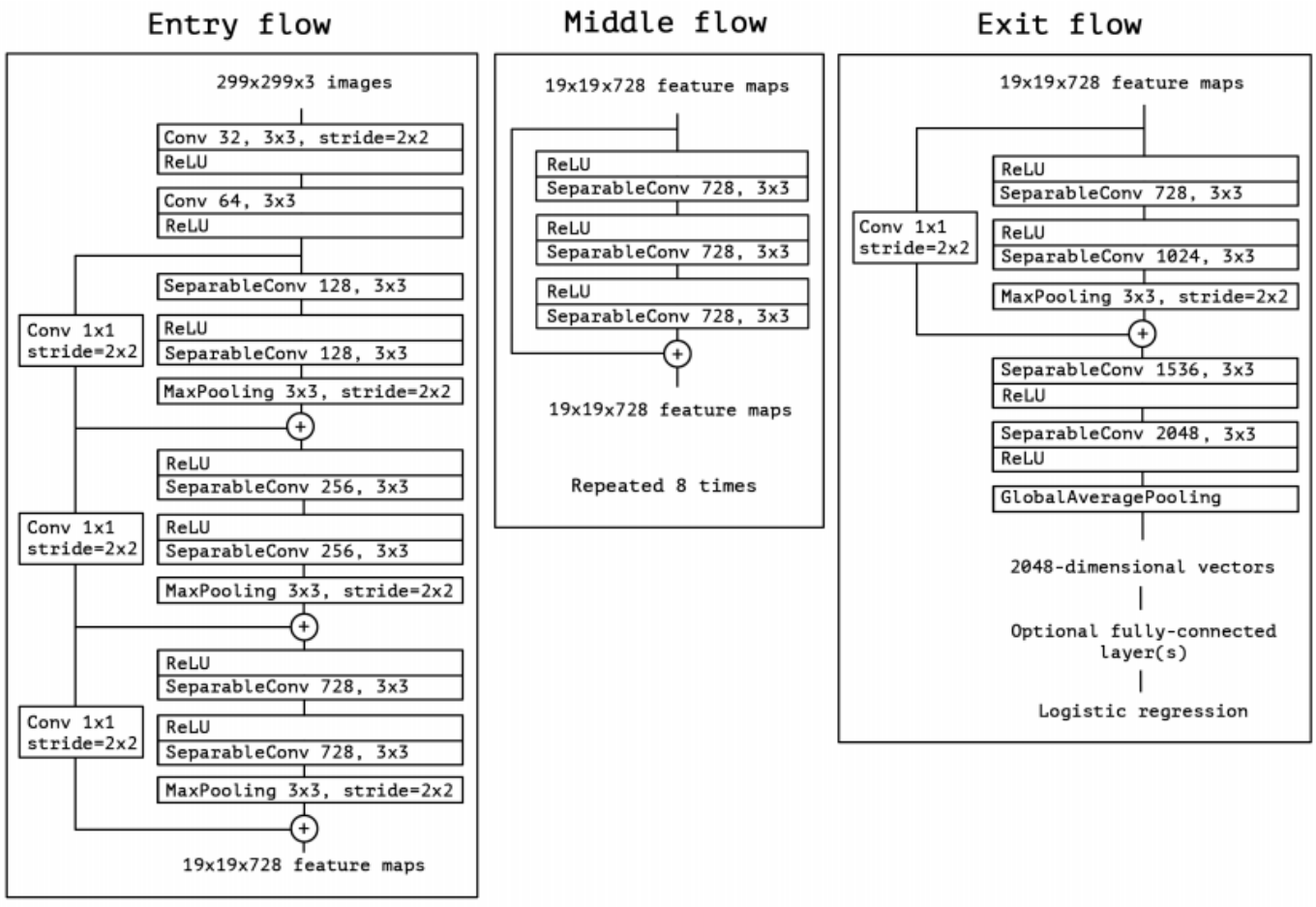
K_06 Understanding of XCeption - EN

Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. It was developed by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads them to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions.

On this page

What does it look like?

The data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization.



As in the figure above, **SeparableConv** is the modified depthwise separable convolution. We can see that SeparableConvs are treated as Inception Modules and placed throughout the whole deep learning architecture.

And there are residual (or shortcut/skip) connections, originally proposed by ResNet, placed for all flows.

Xception architecture has overperformed VGG-16, ResNet and Inception V3 in most classical classification challenges.

How does XCeption work?

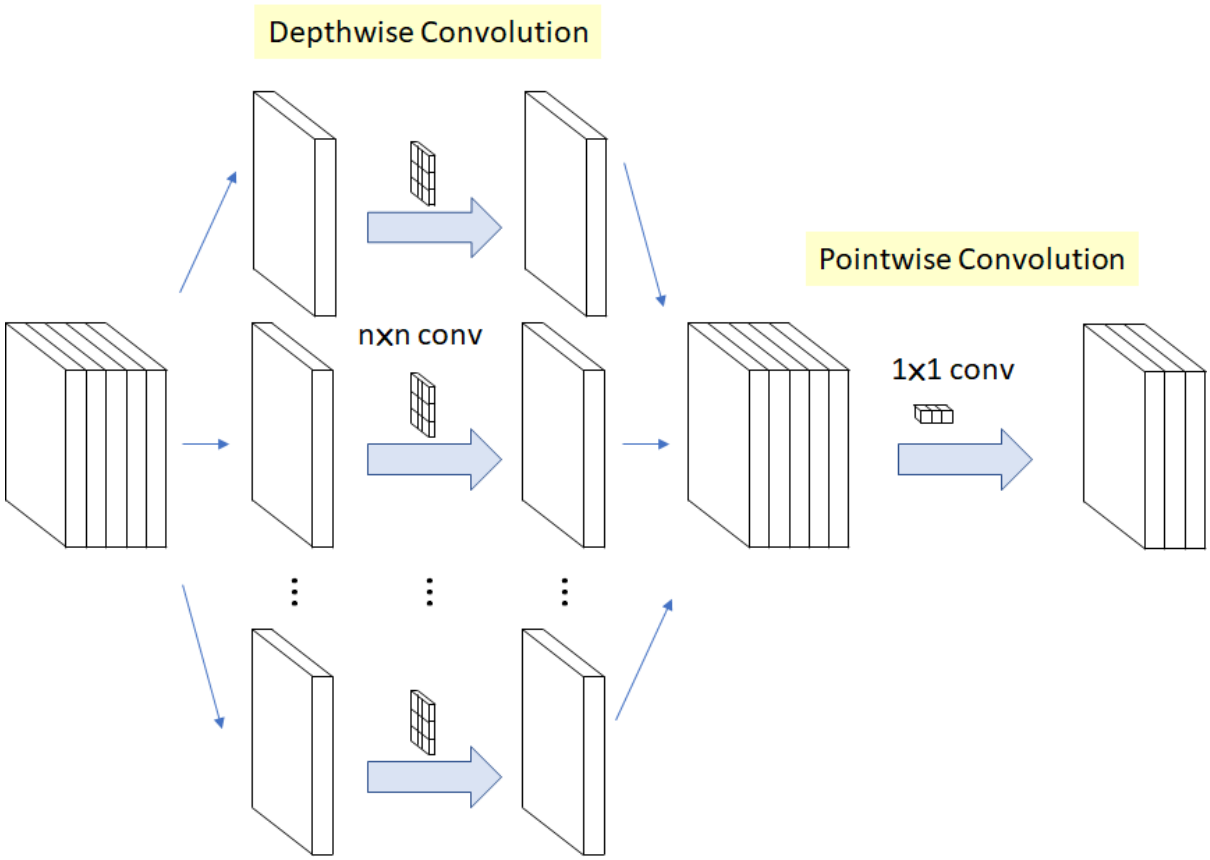
XCeption is an efficient architecture that relies on two main points :

- Depthwise Separable Convolution
- Shortcuts between Convolution blocks as in ResNet

Depthwise Separable Convolution

Depthwise Separable Convolutions are alternatives to classical convolutions that are supposed to be much more efficient in terms of computation time.

Original Depthwise Separable Convolution



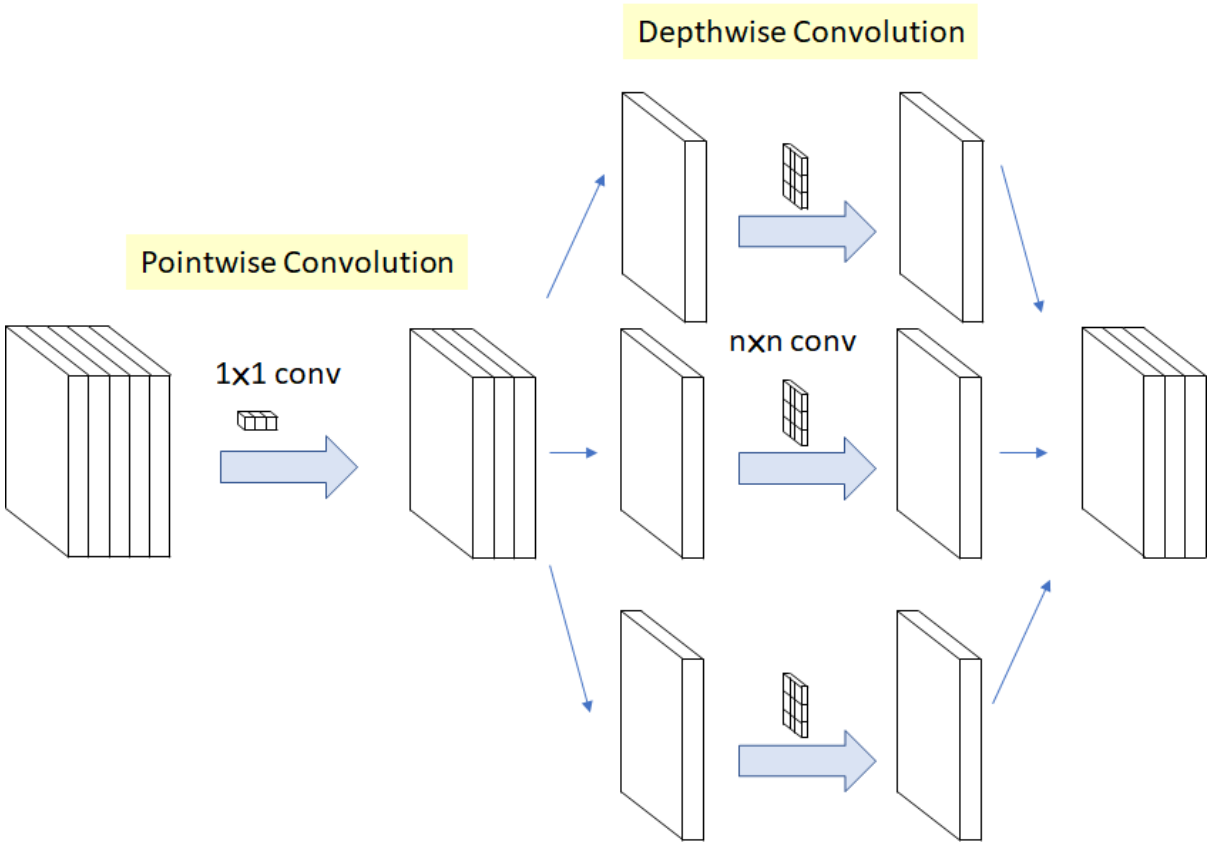
Original Depthwise Separable Convolution

The original depthwise separable convolution is the depthwise convolution followed by a pointwise convolution.

- Depthwise convolution is the channel-wise $n \times n$ spatial convolution. Suppose in the figure above, we have 5 channels, then we will have 5 $n \times n$ spatial convolution.
- Pointwise convolution actually is the 1×1 convolution to change the dimension.

Compared with conventional convolution, we do not need to perform convolution across all channels. That means the number of connections are fewer and the model is lighter.

Modified Depthwise Separable Convolution in Xception



The Modified Depthwise Separable Convolution used as an Inception Module in Xception, so called “extreme” version of Inception module (n=3 here)

The modified depthwise separable convolution is the pointwise convolution followed by a depthwise convolution. This modification is motivated by the inception module in Inception-v3 that 1×1 convolution is done first before any $n\times n$ spatial convolutions. Thus, it is a bit different from the original one. ($n = 3$ here since 3×3 spatial convolutions are used in Inception-v3.)

Two minor differences:

- The order of operations: As mentioned, the original depthwise separable convolutions as usually implemented (e.g. in TensorFlow) perform first channel-wise spatial convolution and then perform 1×1 convolution whereas the modified depthwise separable convolution perform 1×1 convolution first then channel-wise spatial convolution. This is claimed to be unimportant because when it is used in stacked setting, there are only small differences appeared at the beginning and at the end of all the chained inception modules.
- The Presence/Absence of Non-Linearity: In the original Inception Module, there is non-linearity after first operation. In Xception, the modified depthwise separable convolution, there is NO intermediate ReLU non-linearity.

Intro to Xception

Xception-The Extreme Inception! Sounds cool and Xtreme! But “Why such a name??”, one might wonder! One obvious thing is that the author Francois Chollet (creator of Keras) had been inspired by the Inception architecture. He tells about how he sees the Inception architecture in his abstract, which I’ve quoted below.

We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation

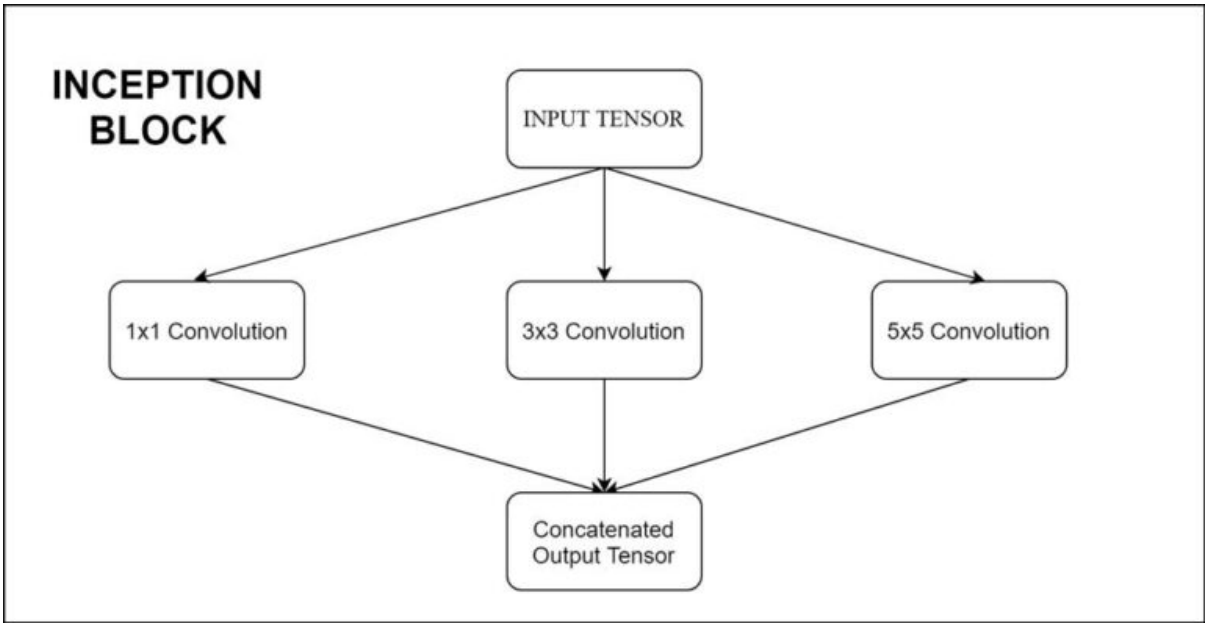
-Francois Chollet in the Xception paper

Another new deep learning term for today, “depthwise separable convolution”. Now, I know that most of you would’ve guessed that it’s some kind of layer. But for those who have never heard it, fret not. I’ve got you covered in this post. We’ll go deeper about what’s a depthwise separable convolution and how it’s used to build the Xception model. And also more on how it builds upon the Inception hypothesis. As always, I’ll try to throw in more illustrations to make the details clear. Let’s dive in!

The Inception Hypothesis

The Inception-style architecture was introduced in the “Going Deeper With Convolution” paper. The authors called the model introduced in the paper as GoogLeNet, which used the Inception blocks. It was a novel and innovative architecture and it still is. Also, it got much attention as many architectures at that time were stacks of more and more layers to increase network capacity. Inception, on the other hand, was more creative and slick!

Rather than just going deeper by simply adding more layers, it also went wide. We’ll see what I mean by “wide” shortly. The Inception blocks take in an input tensor and perform a combination of convolution and pooling in parallel.

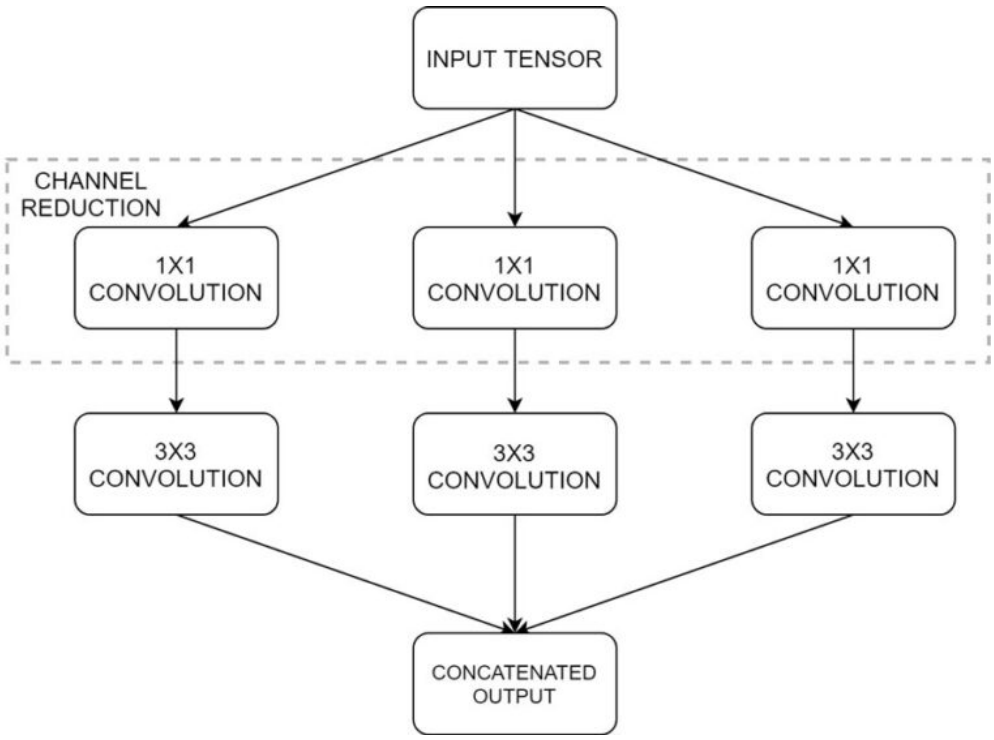


For people who have seen or read the Inception papers, you might find that this is not exactly like an Inception block. Yeah, you're right! I've just illustrated it this way so that everybody gets a rough idea of what it does. You can call it "the naive version of inception" as the authors called it.

Now, the actual Inception block is a little bit different in terms of the number of convolutions, their size, and how they're layered. But this naive illustration conveys what I meant by "wide" before. The block performs convolution with different filter sizes in parallel. And the output tensors are concatenated along the channel dimension i.e stacked one behind the other.

Going a Bit Deeper Towards Xception

Now that you've seen the parallel convolution block, I shall go a bit deeper about the block above. The input tensor which is processed by the different convolution would be of shape (BatchSize, Height, Width, Channels). In the Inception block, the input tensor's channel dimension is reduced using 1×1 convolution before applying 3×3 or 5×5 convolutions. This reduction in channel size is done to reduce the computation when feeding this tensor to the subsequent layers. You can find this concept explained in great detail in the 1×1 convolution article.

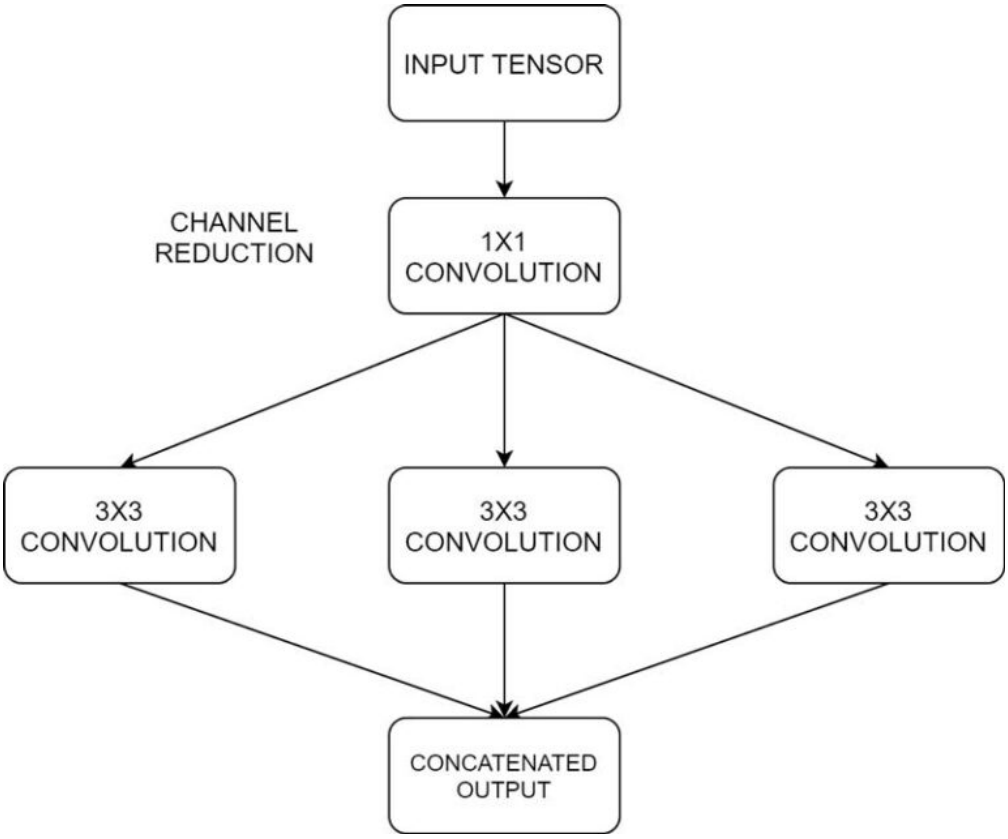


Again, this is just another depiction to understand and is not drawn to be the same as the original Inception block. You can give or take some layers, maybe even wider, and make your own version. The input tensor is individually processed by the three convolution towers. And the three separate output tensors are concatenated along the channel dimension. The GoogLeNet uses multiple Inception block and a few other tricks and tweaks to achieve its performance. I believe, now you get the idea of what an Inception block does.

Next, we shall remodel the above block to make it “**Xtreme**”!

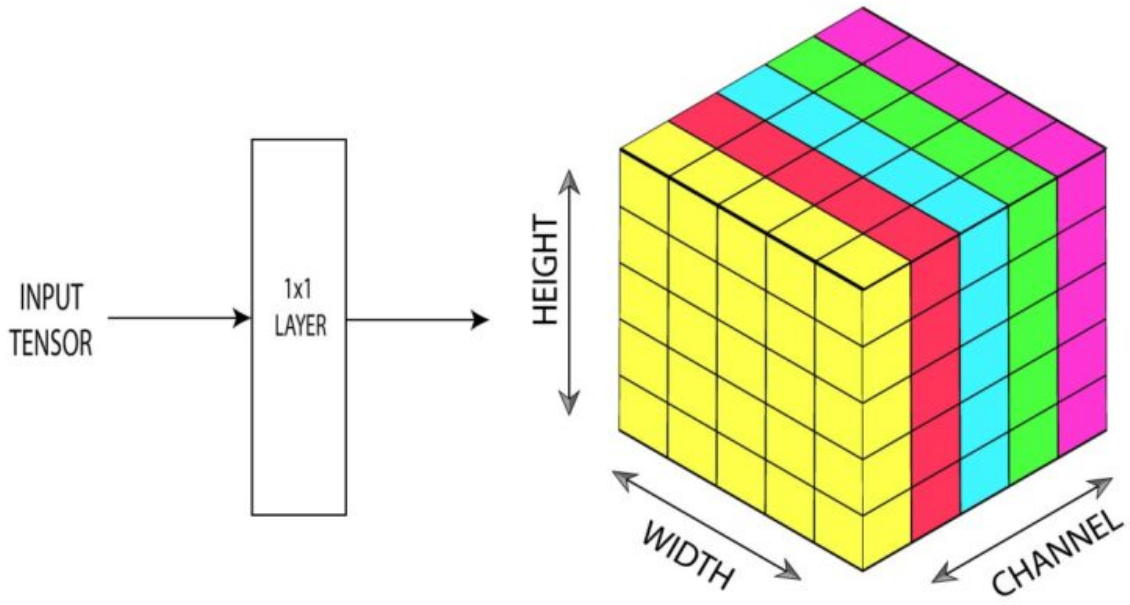
Making it Xtreme

We’ll replace the three separate 1×1 layers in each of the parallel towers, with a single layer. It’ll look something like this.



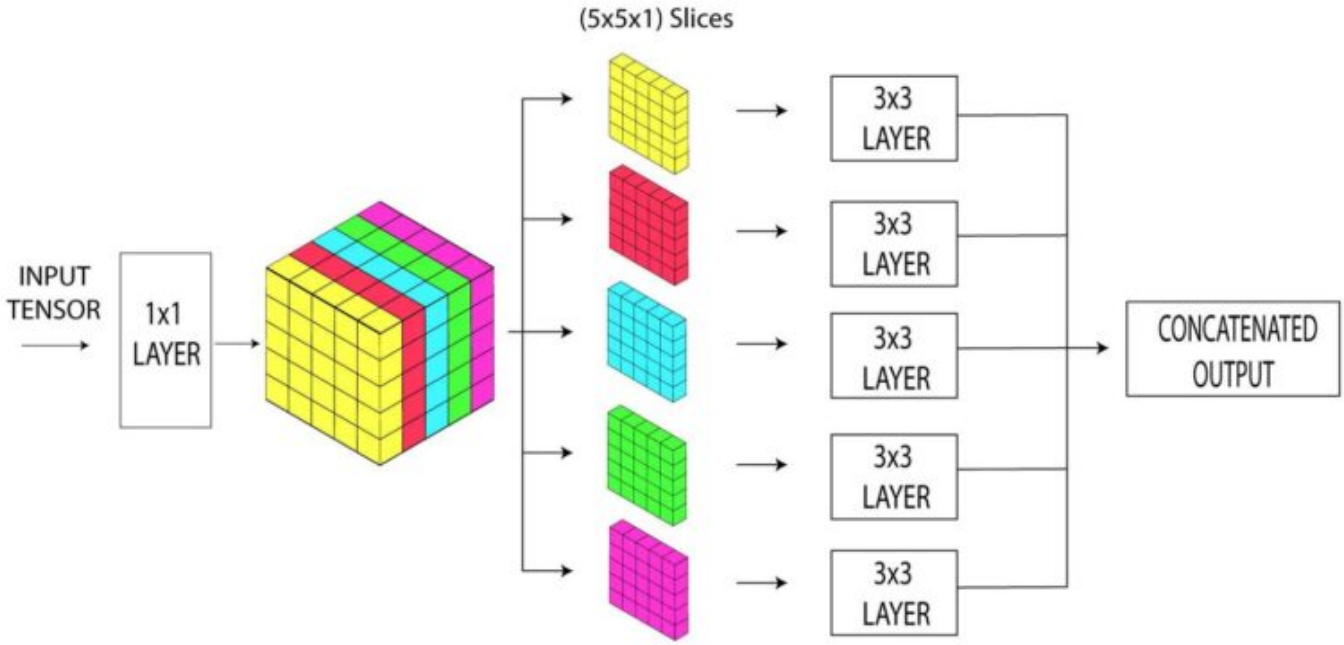
Inception block with a common 1×1 layer

Rather than just passing on the output of the 1×1 layer to the following 3×3 layers, we'll slice it and pass each channel separately. Let me illustrate with an example. Say, the output of the 1×1 layer is of shape $(1\times5\times5\times5)$. Let's not consider the batch dimension and just see it as a $(5\times5\times5)$ tensor. This is sliced along the channel dimension as shown below and fed separately to the following layers.



Slicing of the output tensor of 1×1 convolution along the channel dimension

Now, each of the slices is passed on to a separate 3×3 layer. This means that each 3×3 block will have to process a tensor of shape $(5\times5\times1)$. Therefore, there'll be 5 separate convolution blocks, one for each slice. And each of the convolution blocks will just have a single filter.



Each channel slice is fed to a separate 3×3 layer with just one filter each

This same process scales up to bigger input tensors. If the output of the pointwise convolution is $(5\times5\times100)$, there'd be 100 convolution blocks each with one filter. And all of their output would be concatenated at last. This way of doing convolution is why it's named EXTREME as each channel of the input tensor is processed separately.

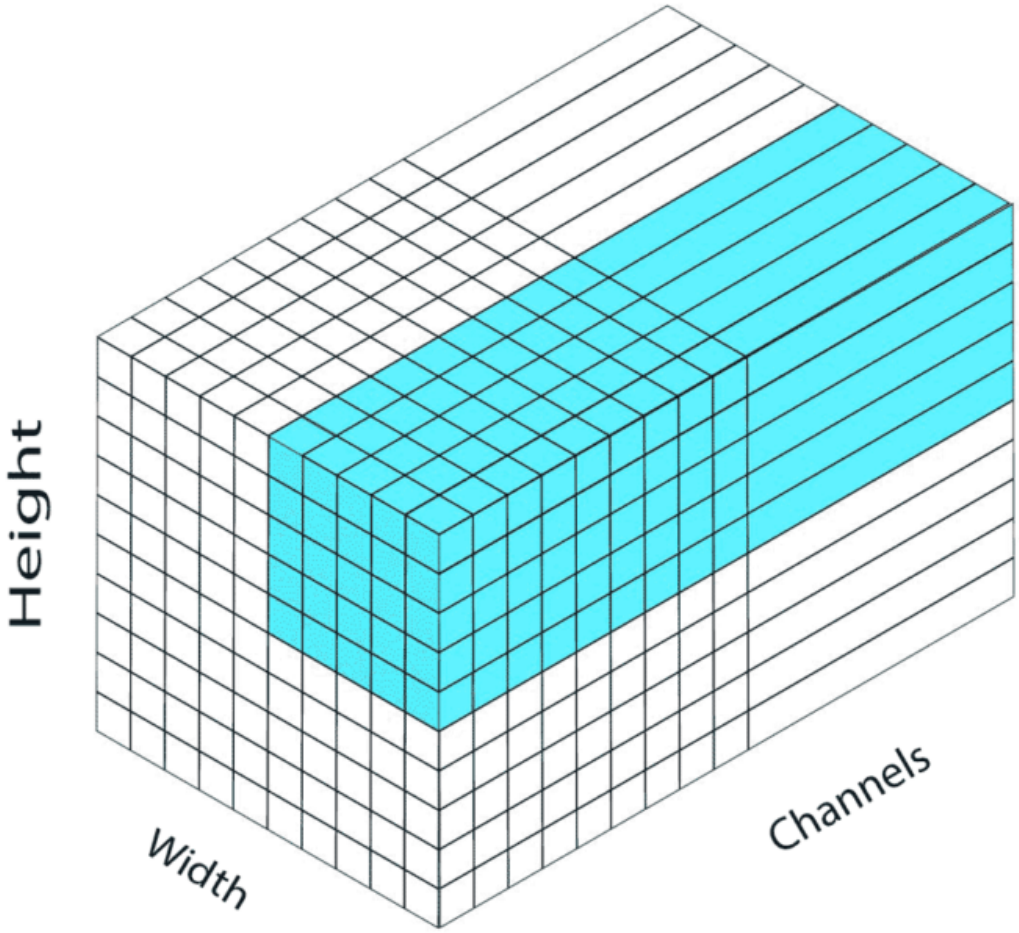
We have seen enough about the idea of Inception and an Xtreme version of it too. Now, let's venture further to see what powers the Xception architecture.

Depthwise Separable Convolution: That which powers Xception

The depthwise separable convolution layer is what powers the Xception. And it heavily uses that in its architecture. This type of convolution is similar to the extreme version of the Inception block that we saw above. But differs slightly in its working. Let's see how!

Consider a typical convolution layer with ten 5×5 filters operating on a $(1\times10\times10\times100)$ tensor. Each of the ten filters is of shape $(5\times5\times100)$ and slides over the input tensor to produce the output. Each of the 5×5 filters covers the whole channel dimension (the entire 100) as it slides

over the input. This means a typical convolution operation encompasses both the spatial (height, width) and the channel dimensions.



A 5×5 convolution filter sliding over a $(10\times10\times100)$ tensor

If you're not familiar with convolutions, I suggest you go through How Convolution Works?

A depthwise separable layer has two functional parts that split the job of a conventional convolution layer. The two parts are depthwise convolution and pointwise convolution. We'll go through them one by one.

Depthwise Convolution

Let's take an example of a depthwise convolution layer with 3×3 filters that operate on an input tensor of shape $(1\times5\times5\times5)$. Again, let's lose the batch dimension for simplicity as it doesn't change anything and consider it as a $(5\times5\times5)$ tensor. Our depthwise convolution will have five 3×3 filters one for each channel of the input tensor. And each filter will slide spatially through a single channel and generate the output feature map for that channel.

As the number of filters is equal to the number of channels of the input, the output tensor will also have the same number of channels. Let's not have any zero paddings in the convolution operation and keep the **stride** as 1.

$$H = \frac{(\text{Height} - \text{FilterSize} + 2*\text{Padding})}{\text{Stride}} + 1$$

$$W = \frac{(\text{Width} - \text{FilterSize} + 2*\text{Padding})}{\text{Stride}} + 1$$

Output Height and Width calculation after applying a convolution

Going by the formula for the output size after convolution, our $(5 \times 5 \times 5)$ will become a $(3 \times 3 \times 5)$ tensor. The illustration below will make the idea clear!

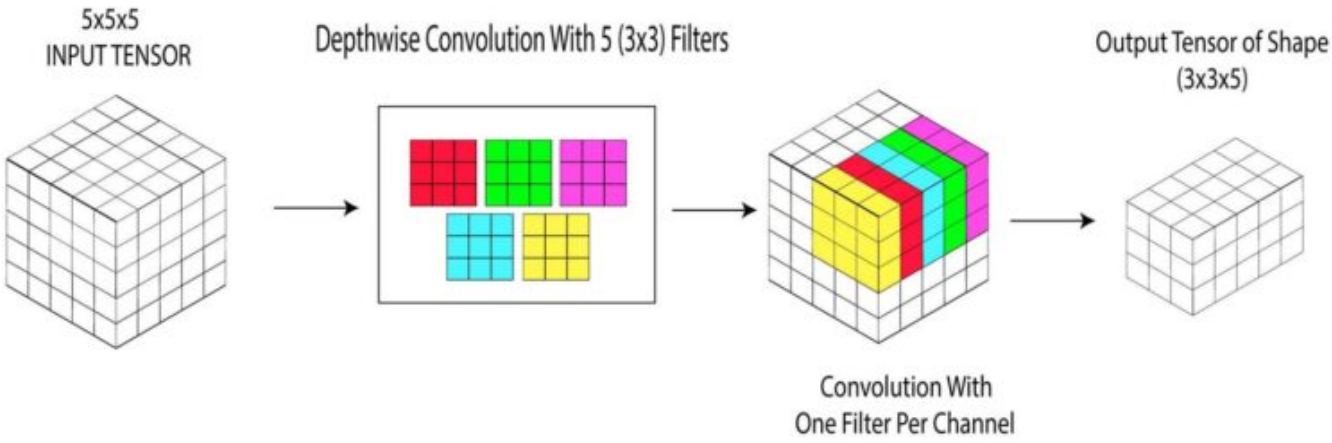


Illustration of Depthwise Convolution Operation

That's Depthwise Convolution for you! You can see that it's almost similar to the way we did the Xtreme convolution in the Inception.

Next up, we have to feed this output tensor to a pointwise convolution which performs cross-channel correlation. It simply means that it operates across all the channels of the tensor.

Pointwise Convolution

The pointwise convolution is just another name for a 1×1 convolution. If we ever want to increase or decrease the depth (channel dimension) of a tensor, we can use a pointwise convolution. That's why it was used in the Inception block to reduce the depth before the 3×3 or 5×5 layers. Here, we're gonna use it to increase the depth. But how?

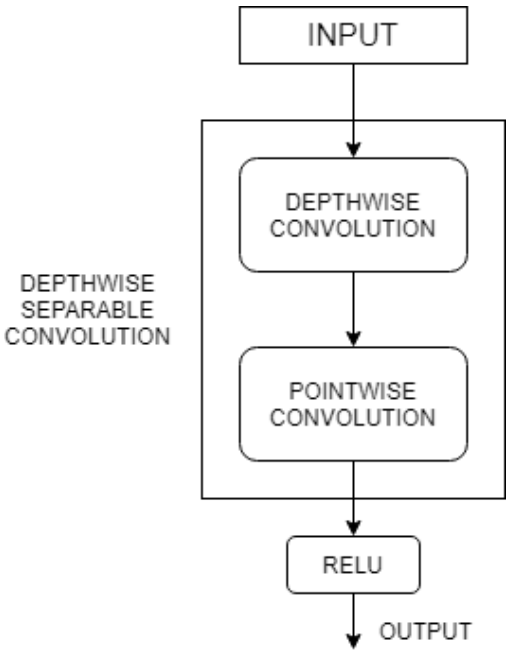
The pointwise convolution is just a normal convolution layer with a filter size of one (1×1 filters). Therefore, it doesn't change the spatial output size after convolution. In our example, the output tensor of the depthwise convolution has a size of $(8\times8\times5)$. If we apply 50 1×1 filters, we'll get the output as $(8\times8\times50)$. And RELU activation is applied in the pointwise convolution layer.

See How Pointwise Convolution Works? for more detailed illustrations and its advantages.

Combining the depthwise convolution and pointwise convolution, we get the **Depthwise Separable Convolution**. Let's just call it **DSC** from here.

Differences between Xception's DSC and the Xtreme Inception

In the Inception block, first comes the pointwise convolution followed by the 3×3 or 5×5 layer. Since we'd be stacking the **DSC** blocks on above the other, the order doesn't matter much. The Inception block applies an activation function on both pointwise and the following convolution layers. But in the **DSC**, it's just applied once, after the pointwise convolution.



The Xception author discusses the effect of having activation on both the depthwise and pointwise steps in the **DSC**. And has observed that learning is faster when there's no intermediate activation.

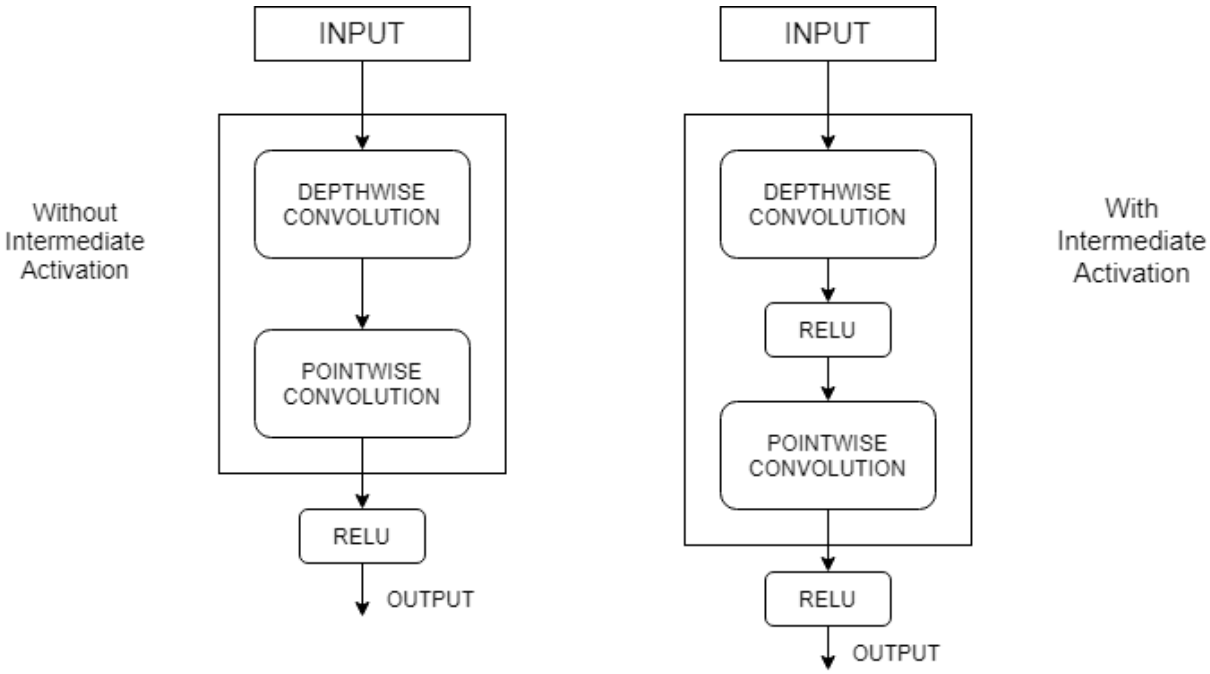


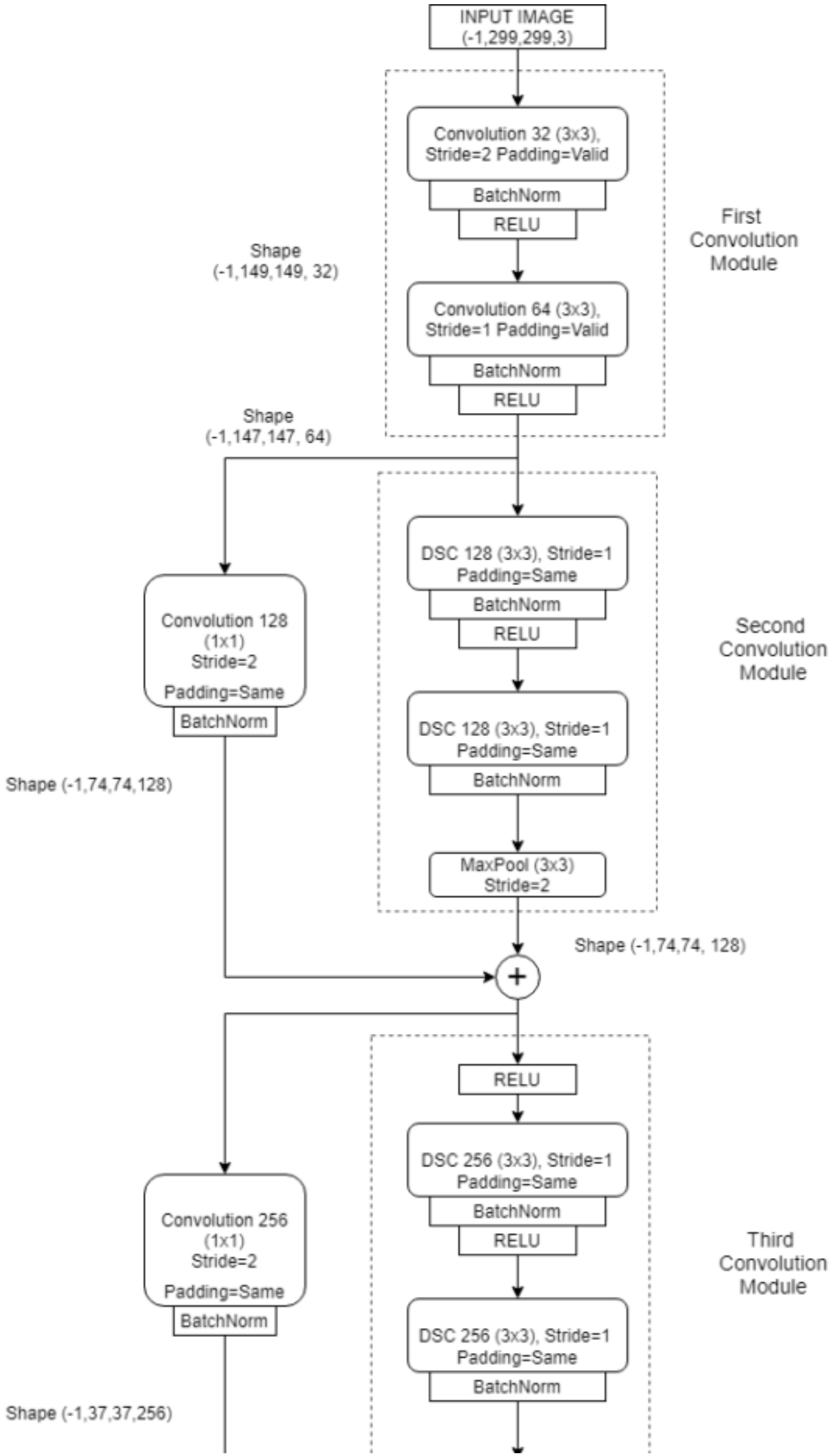
Illustration of DSC with and without having an intermediate activation

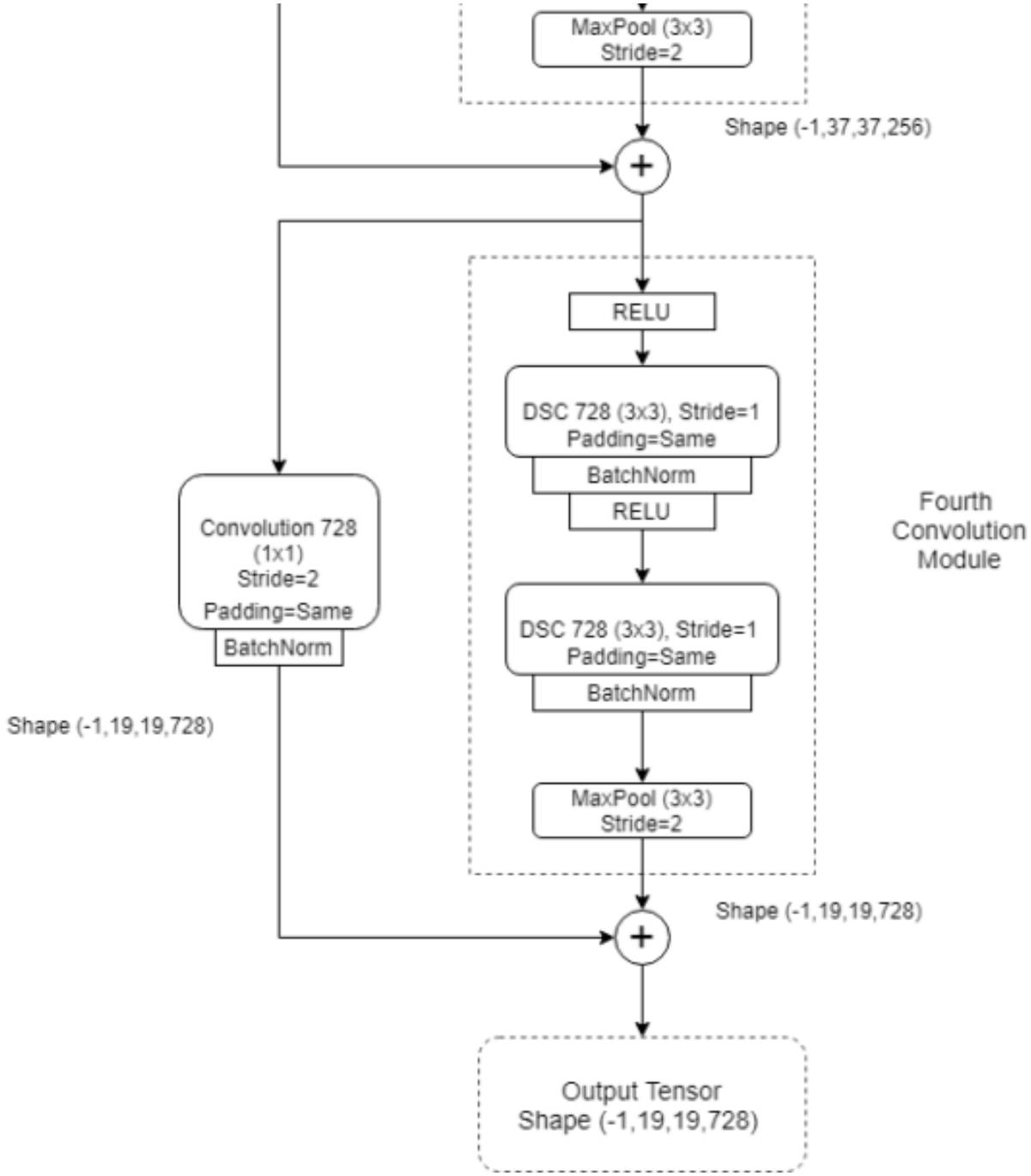
Xception Architecture

The author has split the entire Xception Architecture into 14 modules where each module is just a bunch of DSC and pooling layers. The 14 modules are grouped into three groups viz. the entry flow, the middle flow, and the exit flow. And each of the groups has four, eight, and two modules respectively. The final group, i.e the exit flow, can optionally have fully connected layers at the end.

Note: All the DSC layers in the architecture use a filter size of 3×3, stride 1, and "same" padding. And all the MaxPooling layers use a 3×3 kernel and a stride of 2.

Entry Flow of Xception


INPUT IMAGE
(-1,299,299,3)
Convolution 32 (3x3),
Stride=2 Padding=Valid
BatchNorm
RELU
First
Convolution
Module
Shape
(-1,149,149, 32)
Convolution 64 (3x3),
Stride=1 Padding=Valid
BatchNorm
RELU
Shape
(-1,147,147, 64)
DSC 128 (3x3), Stride=1
Padding=Same
BatchNorm
RELU
Convolution 128
(1x1)
Stride=2
Padding=Same
BatchNorm
Second
Convolution
Module
DSC 128 (3x3), Stride=1
Padding=Same
BatchNorm
Shape (-1,74,74,128)
MaxPool (3x3)
Stride=2
Shape (-1,74,74, 128)
RELU
DSC 256 (3x3), Stride=1
Padding=Same
BatchNorm
RELU
Convolution 256
(1x1)
Stride=2
Padding=Same
BatchNorm
Third
Convolution
Module
DSC 256 (3x3), Stride=1
Padding=Same
BatchNorm
Shape (-1,37,37,256)

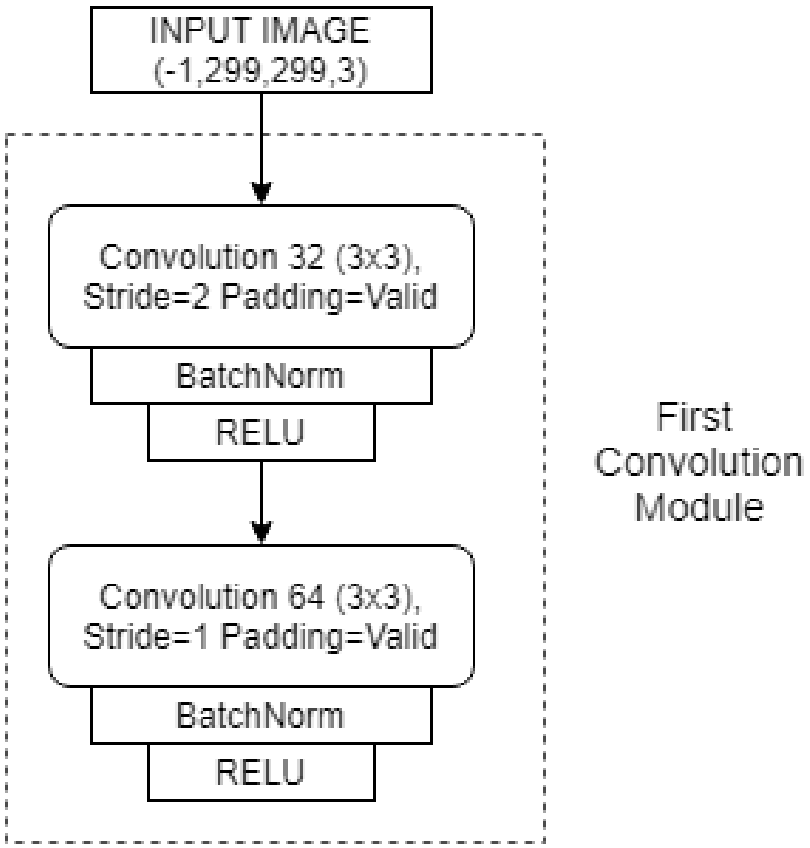


Entry Flow of Xception

The above illustration is a detailed version of the one given in the Xception paper. Might seem intimidating at first but look again, it's very simple.

The very first module contains conventional convolution layers and they don't have any **DSC** ones. They take input tensors of size $(-1, 299, 299, 3)$. The -1 in the first dimension represents the batch size. A negative -1 just denotes that the batch size can be anything.

And every convolution layer, both conventional and **DSC**, is followed by a Batch Normalization layer. The convolutions that have a stride of 2 reduces it by almost half. And the output's shape is shown by the side which is calculated using the convolution formula that we saw before.



First Convolution Module in the Entry Flow

Excluding the first module, all the others in the entry flow have residual skip connections. The parallel skip connections have a pointwise convolution layer that gets added to the output from the main path.

Middle Flow of Xception

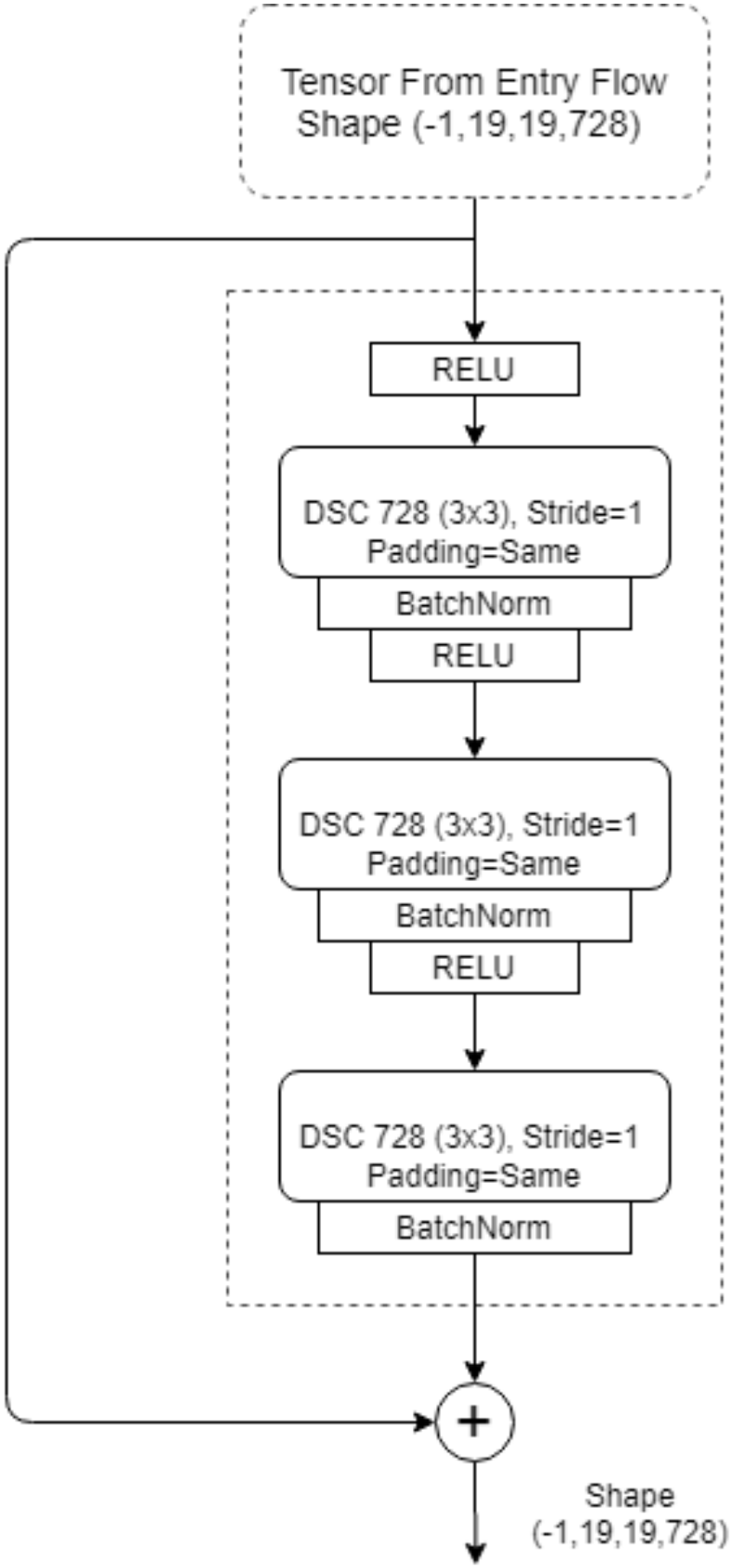
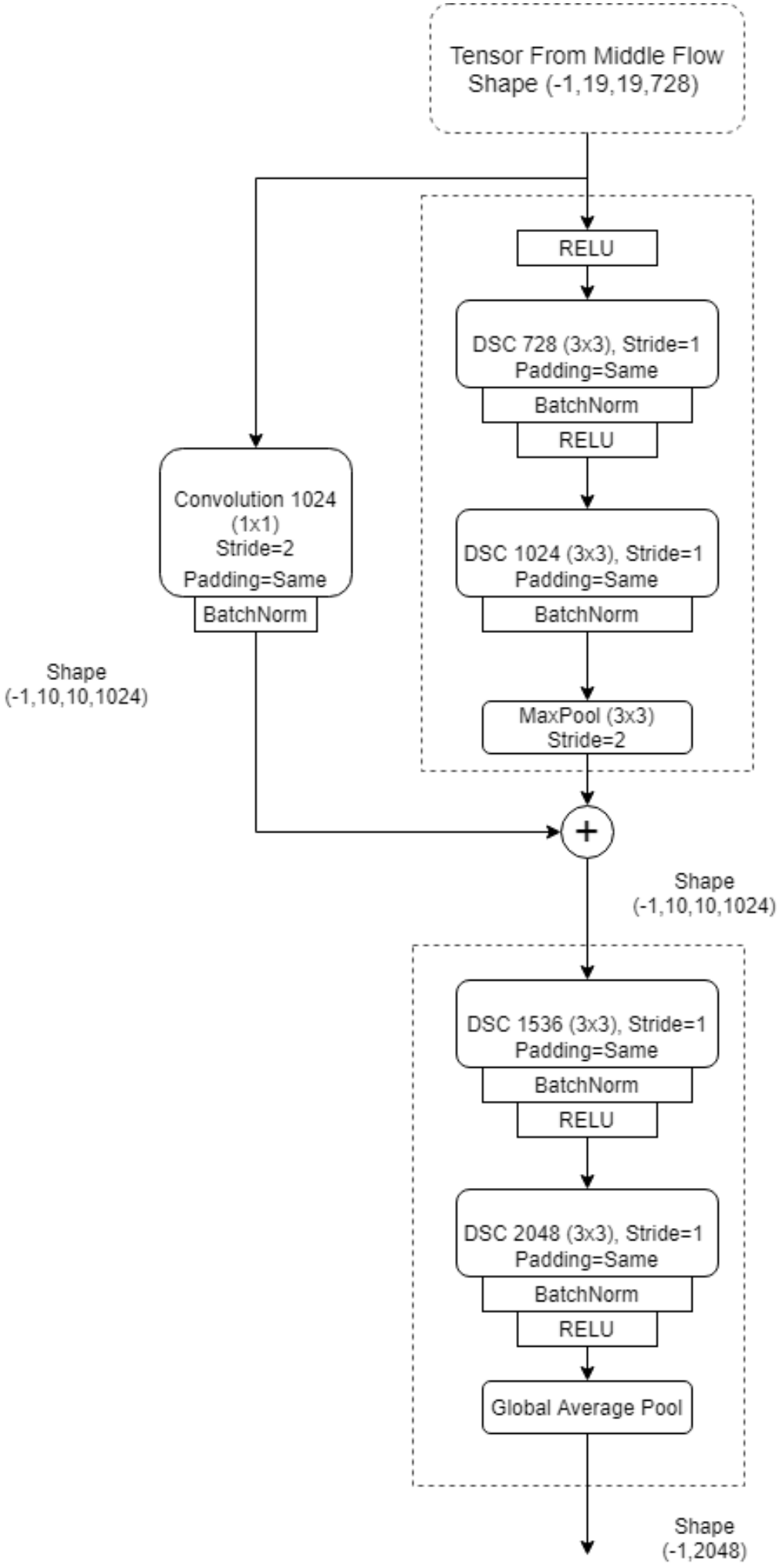


Illustration of middle flow

In the middle flow, there are eight such modules, one after the other. The above module is repeated eight times to form the middle flow. All the 8 modules in the middle flow use a stride of 1 and don’t have any pooling layers. Therefore, the spatial size of the tensor that’s passed from the

entry flow remains the same. The channel depth remains the same too as all the middle flow modules have 728 filters. And that’s the same as the input’s depth.

Exit Flow of Xception


Tensor From Middle Flow
Shape (-1,19,19,728)
RELU
DSC 728 (3x3), Stride=1
Padding=Same
BatchNorm
RELU
DSC 1024 (3x3), Stride=1
Padding=Same
BatchNorm
MaxPool (3x3)
Stride=2
Convolution 1024
(1x1)
Stride=2
Padding=Same
BatchNorm
Shape
(-1,10,10,1024)
+
Shape
(-1,10,10,1024)
DSC 1536 (3x3), Stride=1
Padding=Same
BatchNorm
RELU
DSC 2048 (3x3), Stride=1
Padding=Same
BatchNorm
RELU
Global Average Pool
Shape
(-1,2048)

Exit flow of xception

The exit flow has just two convolution modules and the second one doesn’t have any skip connection. The second module uses Global Average Pooling, unlike the earlier modules which used Maxpooling. The output vector of the average pooling layer can be fed to a logistic regression layer directly. But we can optionally use intermediate Fully Connected layers too.

Last edited by : May 18, 2023, 9:59 a.m.

Comment 0 Feedback

- **Prev** : K_05 Understanding of ResNet - EN
- **Next** : K_07 Understanding of Vision Transformer - EN

TOP