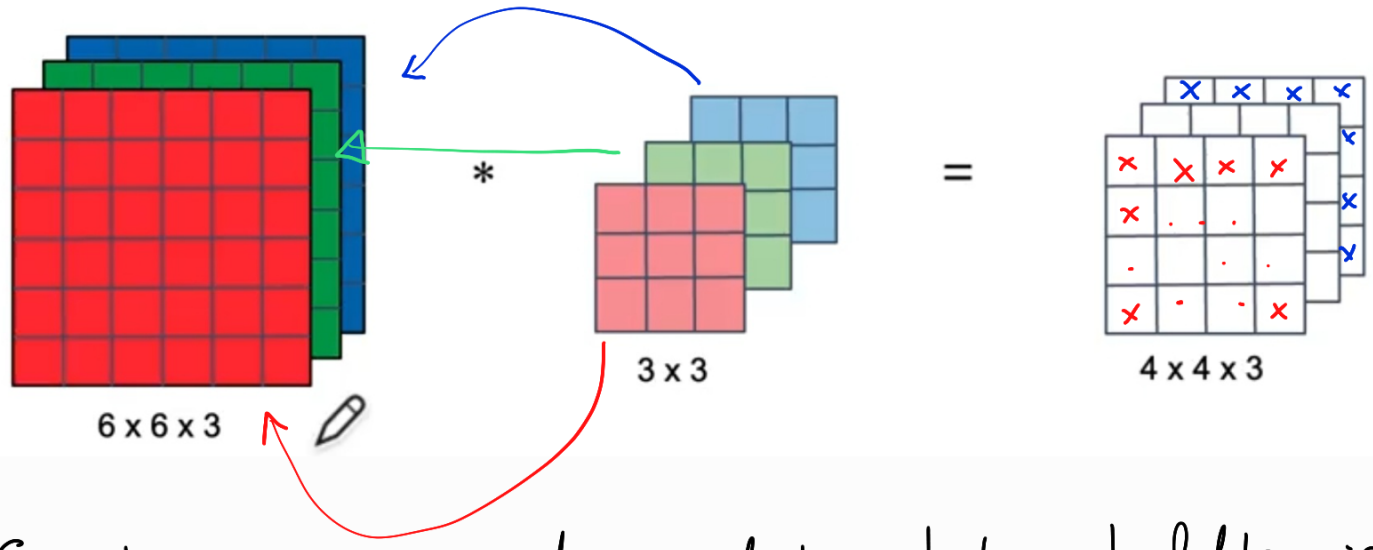
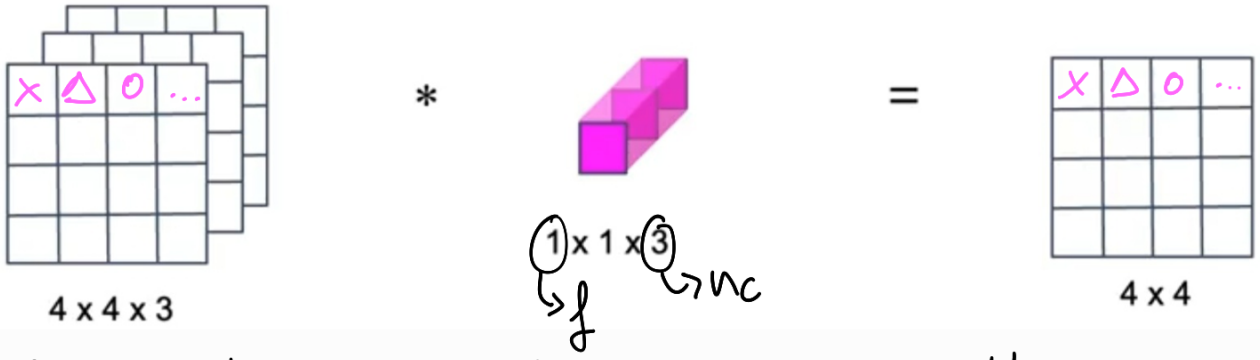
Depthwise Convolution



So, they are normal convolutions, but each filter is different, and applied to a specific layer.

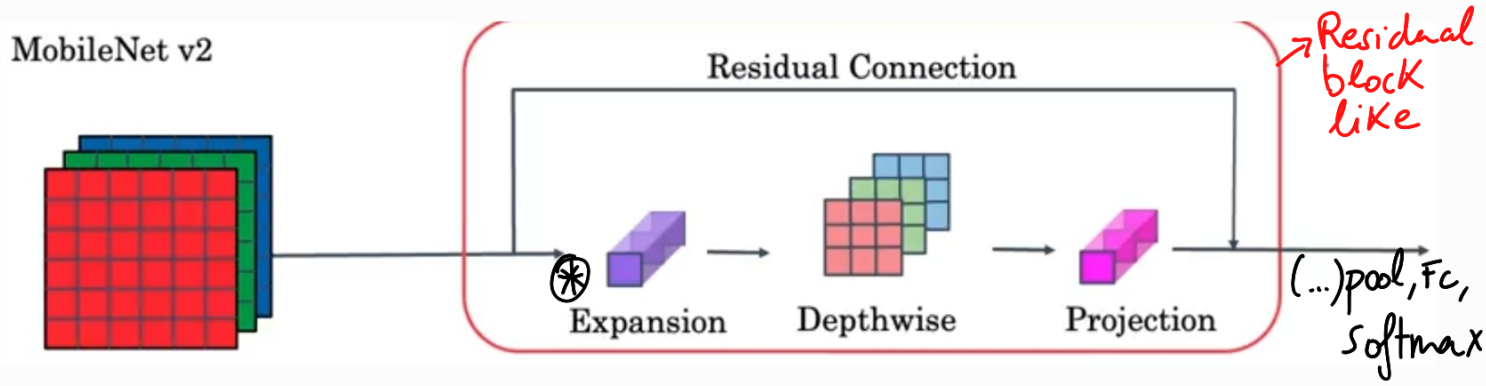
Then, we take the result, and compute a pointwise convolution:

Pointwise Convolution

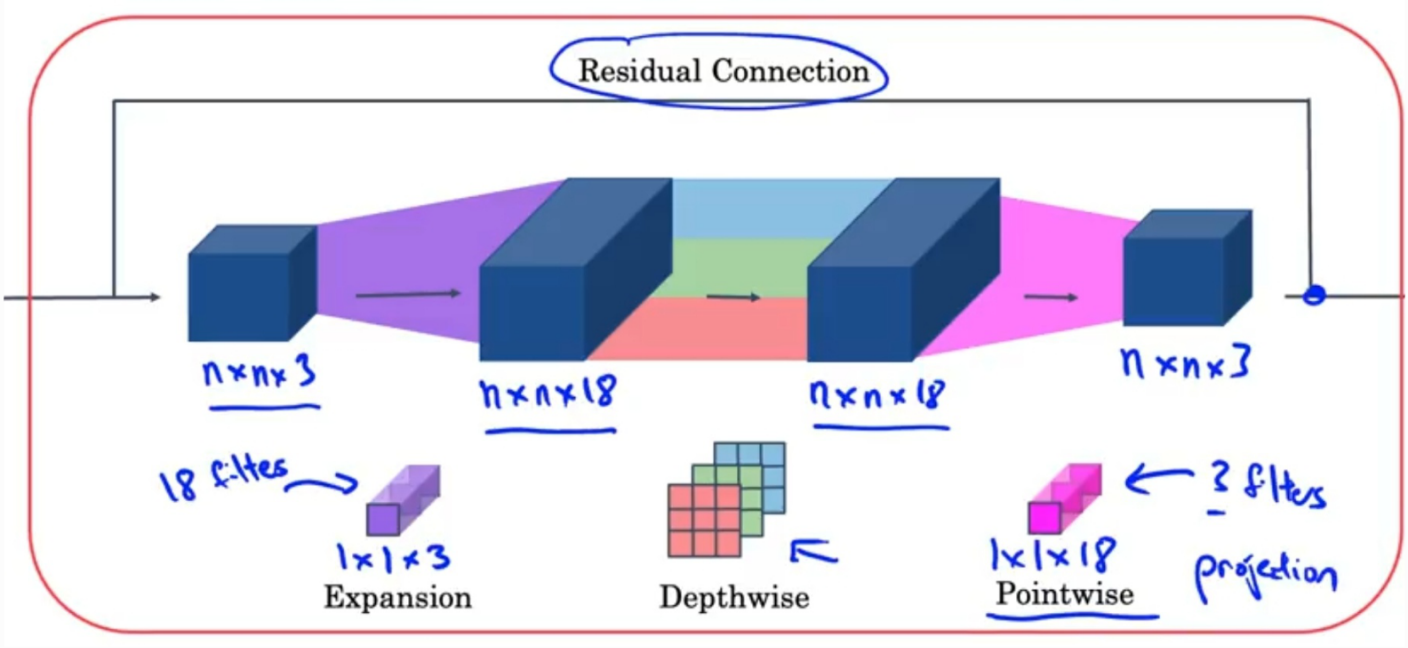


Generally, the computational cost of this method, is about $\frac{1}{\text{number of channels in output}} + \frac{1}{(\text{number of filters})^2} \sim 0'1 \cdot$ normal conv.

$\downarrow$ n_c $\qquad$ $\downarrow$ f^2



It repeats the block, until it makes a prediction with the [POOL + FULLY CONNECTED LAYER + SOFTMAX]



This architecture accomplishes:

- By expanding, it allows the neural network to learn a richer function.
- However, on a mobile device, there is less memory & power, so projecting is key. That way, we can perform a richer set of computations, while also keeping a small memory usage.

What you should remember:

- MobileNetV2's unique features are:
  - Depthwise separable convolutions that provide lightweight feature filtering and creation
  - Input and output bottlenecks that preserve important information on either end of the block
- Depthwise separable convolutions deal with both spatial and depth (number of channels) dimensions

3.1 - Inside a MobileNetV2 Convolutional Building Block

MobileNetV2 uses depthwise separable convolutions as efficient building blocks. Traditional convolutions are often very resource-intensive, and depthwise separable convolutions are able to reduce the number of trainable parameters and operations and also speed up convolutions in two steps:

1. The first step calculates an intermediate result by convolving on each of the channels independently. This is the depthwise convolution.
2. In the second step, another convolution merges the outputs of the previous step into one. This gets a single result from a single feature at a time, and then is applied to all the filters in the output layer. This is the pointwise convolution, or: **Shape of the depthwise convolution X Number of filters.**

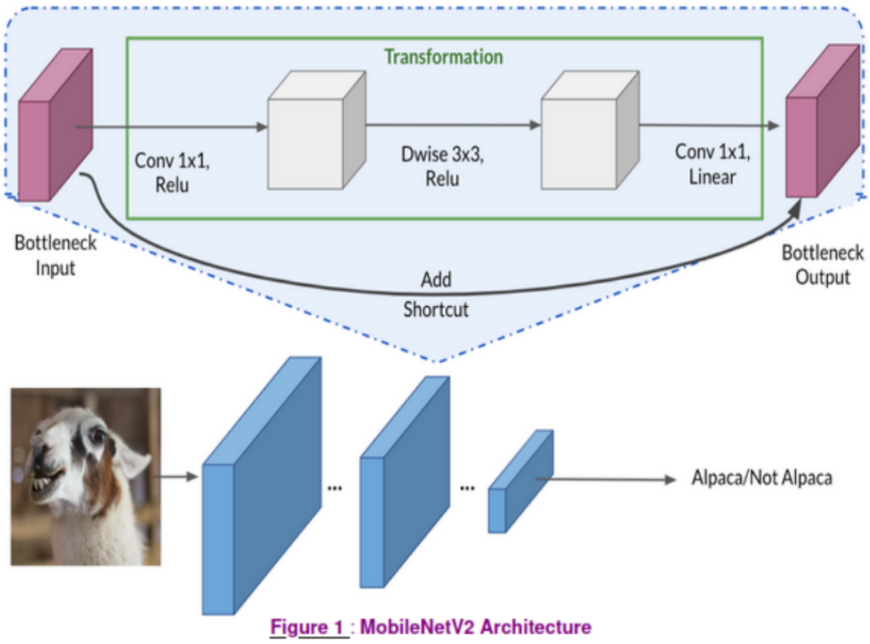


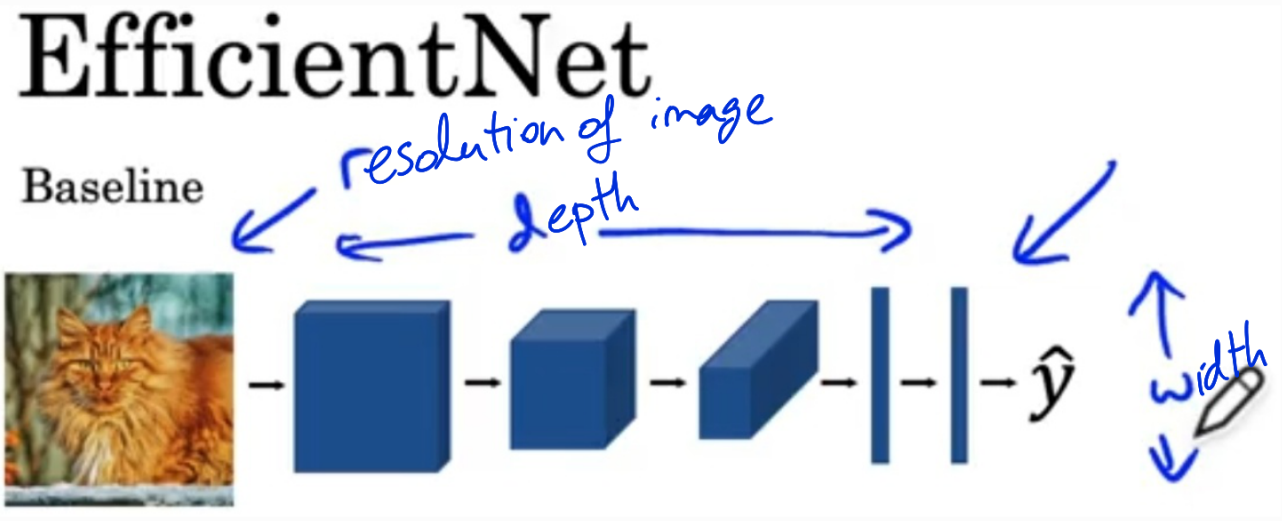
Figure 1 : **MobileNetV2 Architecture**
This diagram was inspired by the original seen here.

Each block consists of an inverted residual structure with a bottleneck at each end. These bottlenecks encode the intermediate inputs and outputs in a low dimensional space, and prevent non-linearities from destroying important information.

The shortcut connections, which are similar to the ones in traditional residual networks, serve the same purpose of speeding up training and improving predictions. These connections skip over the intermediate convolutions and connect the bottleneck layers.

Let's try to train your base model using all the layers from the pretrained model.

Similarly to how you reused the pretrained normalization values MobileNetV2 was trained on, you'll also load the pretrained weights from ImageNet by specifying `weights='imagenet'`.



To scale your model and achieve greater performance:
- Use higher resolution image
- Increase depth
- Increase width of layers

} A combination is called compound scaling

Which configuration to use?