Let me start to answer the question proposed by Yang Sizhe and share our reproduction result. If you have any questions, type it in the chat channel and we will answer it later.

Before talking about the result, let me briefly go through the general architecture of Efficient Net. As you can see from the graph, Efficient Net is just stacks of multiple Mobile Convolution Modules with different sizes, and here we call it as MBConvolution for simplicity. Acutally this architecture is not selected manually, instead it is selected via Reinforcement Learning which is called Network Architecture Searching.

To conclude, there are 2 main parts for our reproduction result. First is to determine the optimal combination of depth, width and resolution for specifc computation resouces. Second is to compare the performance of Efficient Net with other 2 convolutional neural networks. Here we focus on the CNN baseline, which is given in our lecture and the famous ResNet152.

For the first part, from the motivation discussed earlier, it suggests us to uniformly scale up those 3 dimensions. However, the question is, how to explicitly determine these 3 scaling factors? Efficient Net proposes the ‘Compound Scaling Scheme’ to determine the optimal scaling factors.

To help you guys understand this concept better, we do requrire some extra knowledge on the relation between scaling factors and corresponding computation costs. So, let’s get started. Imagine this scenario, the input size is H\_input times W\_input times C\_input and the kernel size is K times K times C\_input and there are totally C\_output number of kernels. Then, after convolution operation, the output size should be H\_output times W\_output times C\_output. If we focus on only one pixel in the output, it actually requires C\_input times K\_square operation, which is equal to number of parameters in one kernel. Therefore, the total computation cost is exactly proportional to this term. The first 2 terms correspond to number of computation for one pixel in the output and last 3 terms correspond to number of pixels.

Based on this result, firstly, we can draw the conclusion that, if we enalrge the width by beta and resolution by gamma, the total computational cost will be proportional to beta square times gamma square larger. Secondly, if we enlarge the depth by alpha, then the computational cost will be proportional to alpha times. Therefore, if we enlarge these 3 factors by alpha, beta, gamma respectively, then the computation costs will be alpha times beta square times gamma square larger. This is the result we want to show. I hope all of you guys can remember this formula since we will use it in next slide.

OK, now, we can jump into the ‘Compond Scaling’ Scheme. Recap that, our first question is like, if our resources are N times larger, how to allocate for these 3 scaling factors? Compond Scaling Scheme says that we can solve this by the following 2 steps. Step 1, let us assume twice more resources are availabe. As we illustrate in the previous slide, we are required to solve the following optimization problem. We maximize the validation accuracy given the constraint that alpha times beta square times gamma square approximately equals to 2. This is the formula we show before. To solve this numerically, we do grid search on Efficient Net baseline.

We conduct experiments as follows. The dataset we use is CIFAR-10 with total classes. We run 100 epochs to make the algorithm converge. Grid Search Result shows that we should choose alpha equals to 1.2, beta equals to 1.1, gamma equals to 1.15.

Step 2 is, Now we achieve the optimal scaling factors and in the first question, we have N times more computation resources. In this setting, the optimal scaling factors can be determined as follows. (stop 3 sec)

Here, phi is called uniform scaling parameter since it uniformly increases these 3 scaling factors. With the knowledge in previous slide, you can easily verify that, the computational cost is exactly enlarged by approximately N times!

To conclude, this ‘Compound Scaling Scheme’ decomposes the large intractable grid search problem into 2 parts, one uniform scaling parameter phi and one small tractable grid search problem. Actually it achieves the trade off between tractability and loyalty to the original problem. I think the idea is pretty cool.

Then we come to the second part. We do comparison between Efficient Net and other CNN architectures including the CNN baseline given in our lecture and the famous ResNet152. In this figure, you can see that, ResNet152 outperforms the CNN baseline a little bit and our Efficient Net greatly outperforms ResNet152.

Next, let me show our model extension result.

We mainly do the 2 things, one is to apply Efficient Net on CIFAR 100 dataset. The other is to imporve the Efficient Net architecture and make comparisons.

For the first part, we compare the EfficientNet and CNN given in our lecture on CIFAR100 dataset. You can see from the graph that, Efficient Net model also behaves much better on the CIFAR100 dataset.

For the second part, to modify the architecture, we should see the limitation first. The main issue of Efficient Net is, it will be slow due to the first 2 modules in MBConvolution. Therefore, we use a simple convolution operator to replace them. The new unit is called Fused-MBConvolution.

The comparison with respect to model performance and model efficiency can be shown in this graph. As for Efficient Net version 2, its accuracy is slightly better but it will be much more efficient.

In the last part of our presentation, we want to make a summary.

Actually EfficientNet starts from the skip connection of ResNet. And in order to improve perforance, it introduces 2 things. Firstly, it modifies the architecture by utilizing MBConvolution unit. Secondly, it creatively proposes Compound Scaling scheme to enlarge the model in a reasonable and clever way. To make some extension, we firstly test its performance on other datasets. Then, we look for the EfficientNet Version 2 paper to see what kinds of modification can be made to improve the current model.

In this project, the dataset we use is CIFAR10 and CIFAR100. All our codes are implemented in colab and its GPU, and the library we use in Torch since its scalability is much better than Tensorflow from my personal perspective. All the codes are sent to the github and the link is attached as follows. If you guys are interested, you can clone and play with it.

That is all for our presentation. Hope this will help you a little bit. Thank you for your attention.