RWTH Aachen University

Ruei-Bo Chen

2022/08/15

ResNet18

Motivation

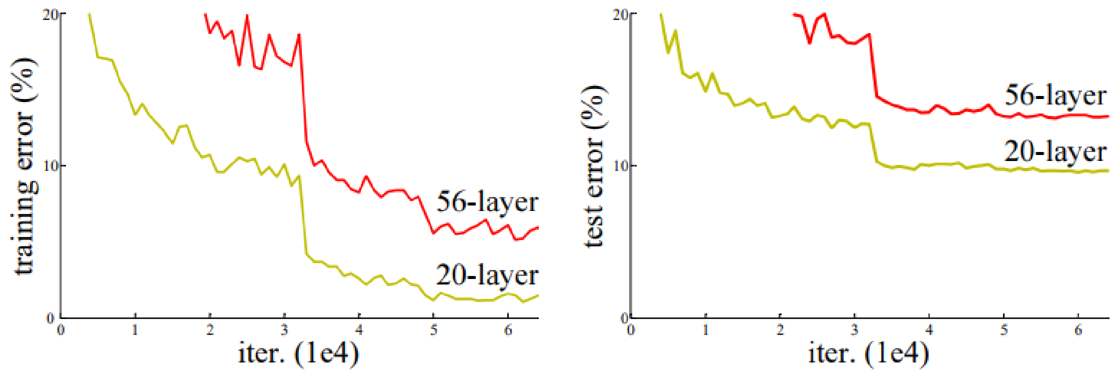


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

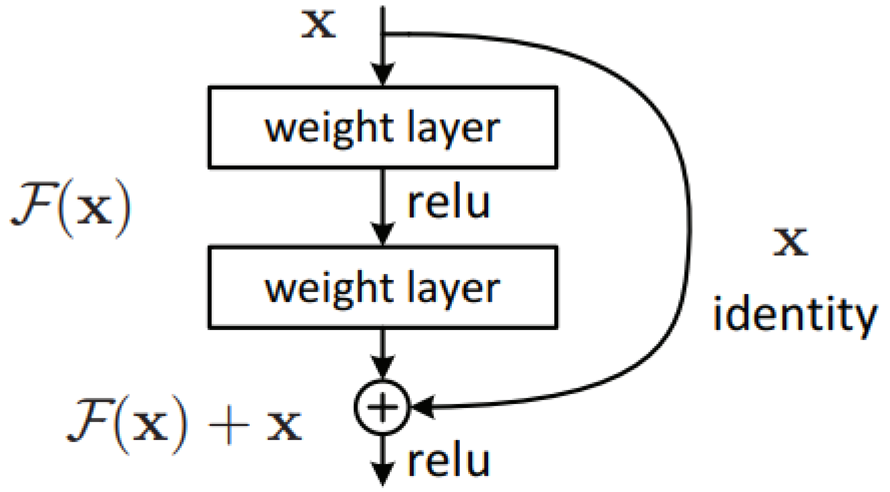


Figure 2. Residual learning: a building block.

ResNet18

Motivation

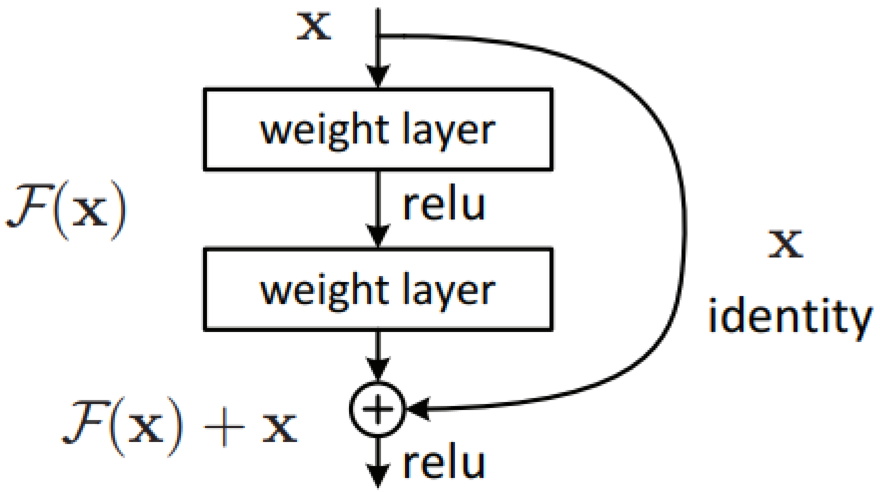


Figure 2. Residual learning: a building block.

$$H(x) = F(x) + x$$

$$F(x) = H(x) - x$$

We want $F(x)$ to be 0 to make this block an identity mapping. If the network can learn identity mapping, then no matter how many layers we add, they will not affect the convergence.

Why don't we just learn $H(x)$?

ResNet18

Motivation

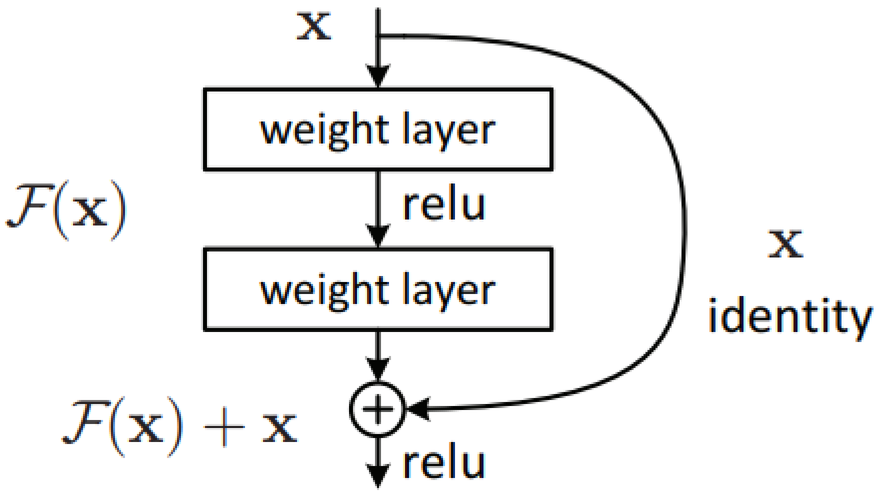


Figure 2. Residual learning: a building block.

$$H(x) = F(x) + x$$

$$F(x) = H(x) - x$$

Why don't we just learn $H(x)$?

For example:

$x = 2.9$, if $H(x) = 3.0 \rightarrow F(x) = 0.1$

$x = 2.9$, if $H(x) = 3.1 \rightarrow F(x) = 0.2$

$$\Delta H = \frac{3.1 - 3.0}{3.0} = 3.3\%,$$

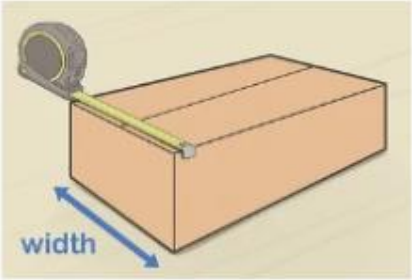
$$\Delta F = \frac{0.2 - 0.1}{0.1} = 100\%$$

The larger the variance is, the easier the network learns.

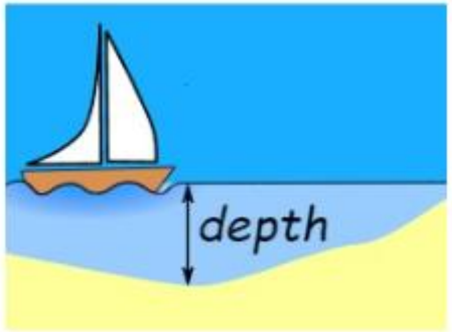
ResNeXt50

Motivation

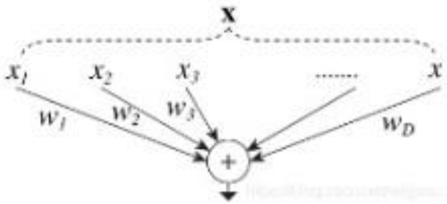
Want to increase the accuracy?



change width



change depth



split-transform-merge

ResNeXt50

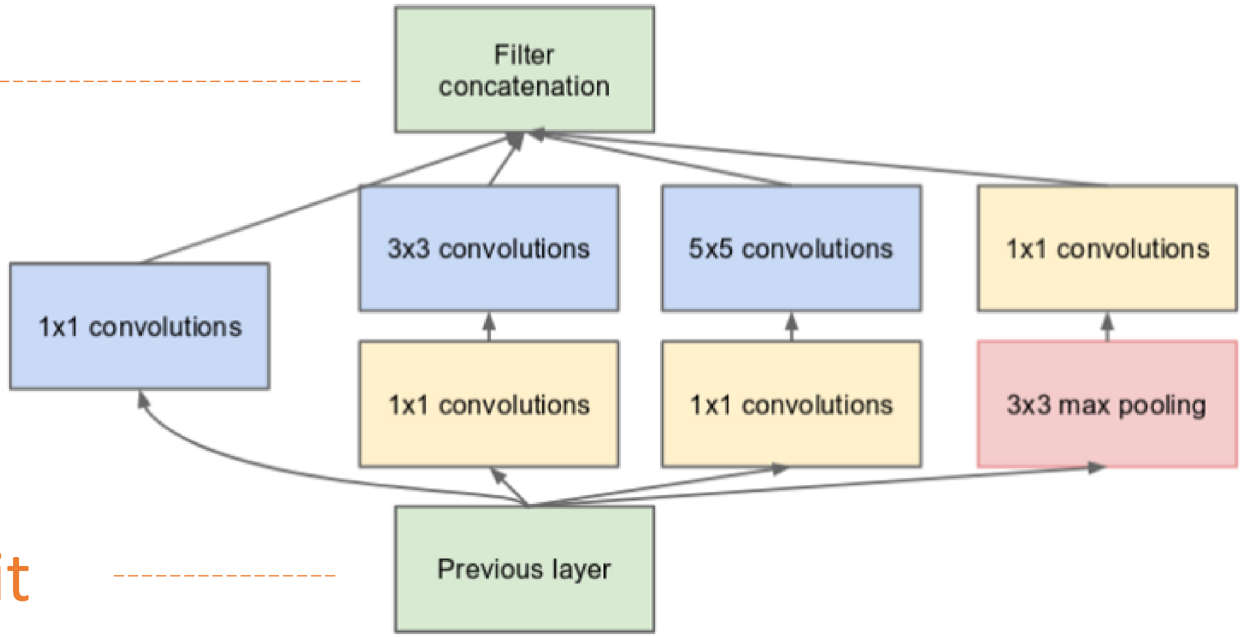
Motivation

Split-Transform-Merge

Merge

Transform

Split



(b) Inception module with dimension reductions

ResNeXt50

Motivation

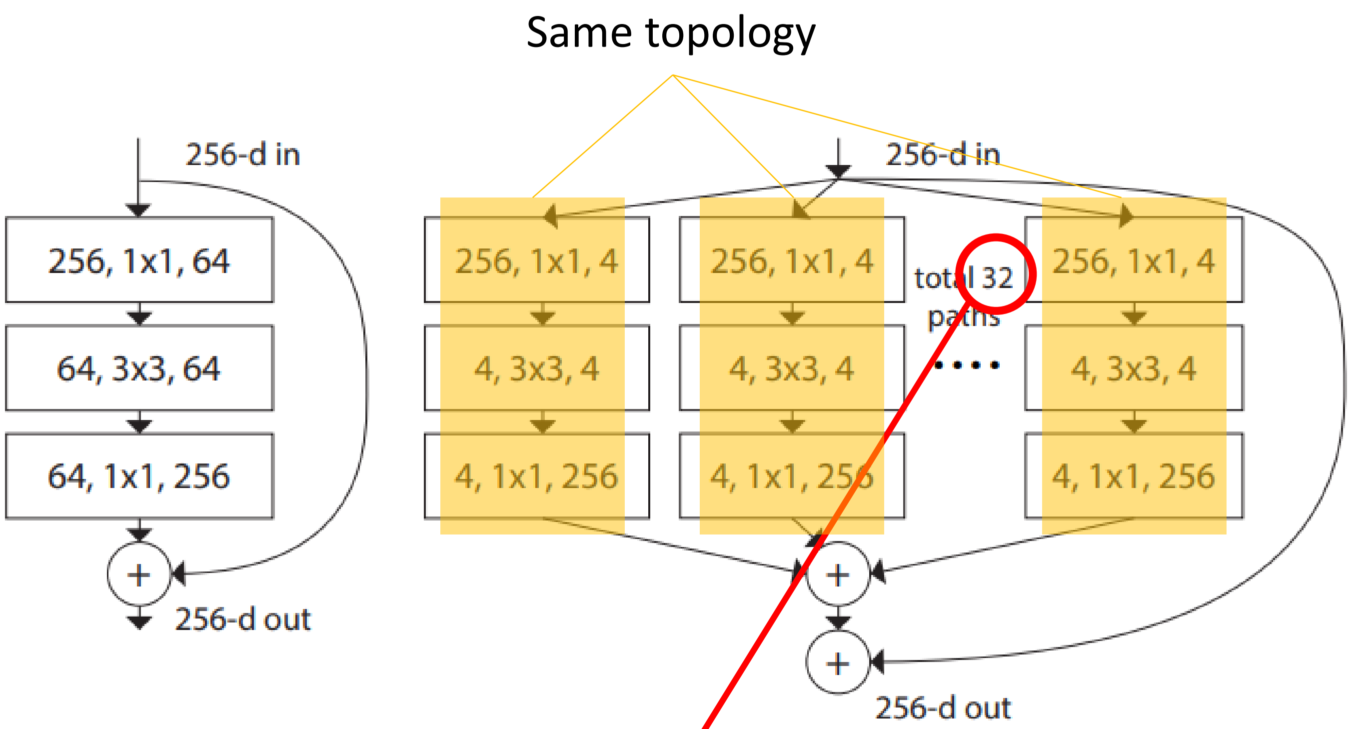


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ResNeXt50

Motivation

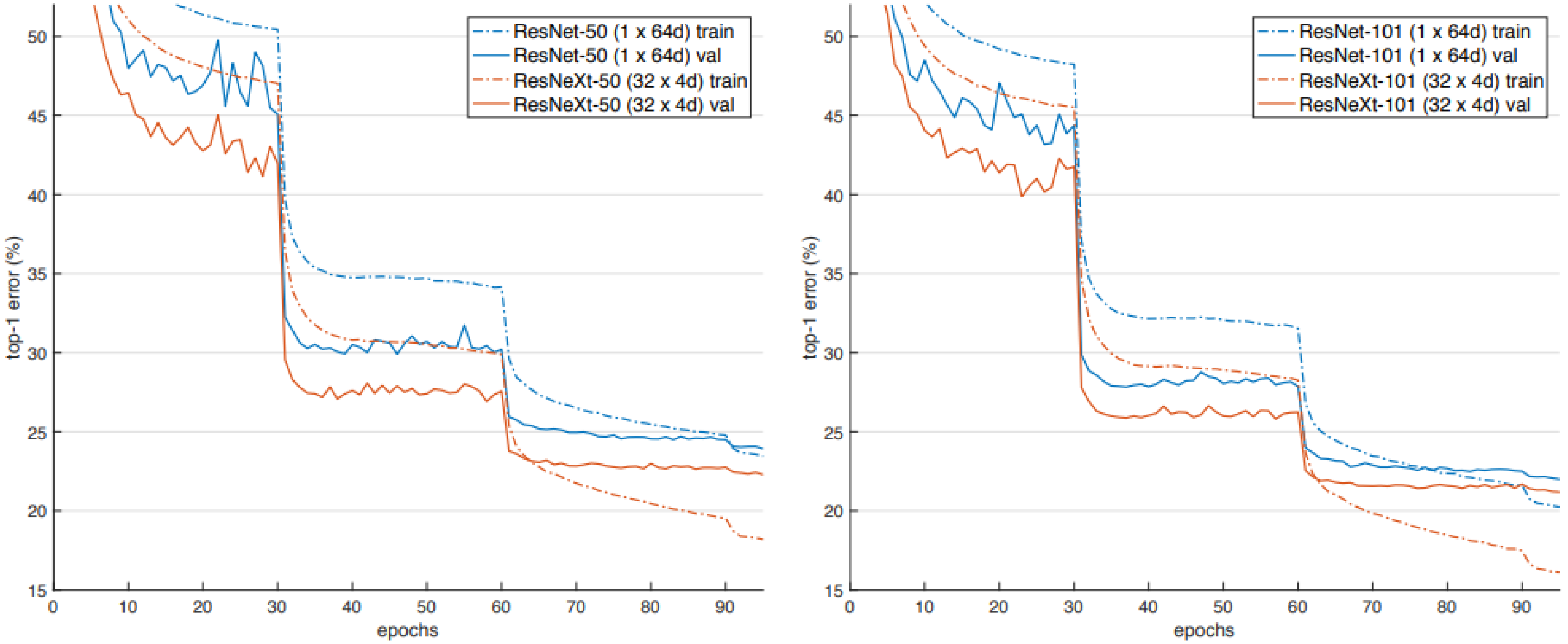


Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (~4.1 billion FLOPs, ~25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (~7.8 billion FLOPs, ~44 million parameters).

ResNeXt50

Motivation

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity (~4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity (~7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

		setting	top-1 err (%)	top-5 err (%)
	1× complexity references:			
	ResNet-101	1 × 64d	22.0	6.0
	ResNeXt-101	32 × 4d	21.2	5.6
	2× complexity models follow:			
Deeper	ResNet-**200** [15]	1 × 64d	21.7	5.8
Wider	ResNet-101, wider	1 × **100**d	21.3	5.7
	ResNeXt-101	**2** × 64d	20.7	5.5
	ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

ResNeXt50

Motivation

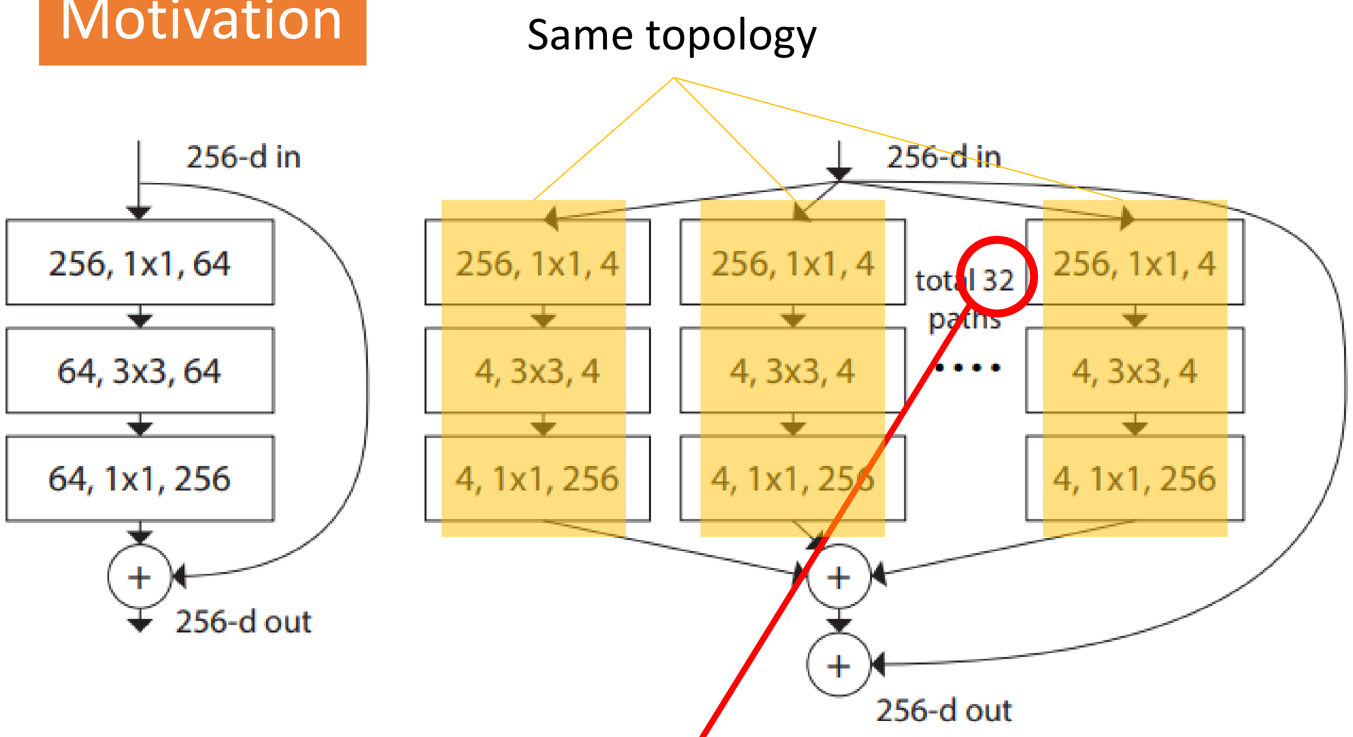


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

1. Combines the greatness of Inception and ResNet. (Allow the model to jointly attend to the information from different feature subspaces)
2. Because the topology of each cardinality is the same, the model reduce the number of hyperparameters, make it more general in other case.
3. Increase the accuracy without increasing the number of parameter.

Comparison

```
model_name = 'resnext'
continued_model = initialize_model(model_name, num_classes, use_pretrained=False)
continued_model.load_state_dict(torch.load('./models/resnext_best/best_weight.pt').state_dict())
continued_model = continued_model.to(device)
model_testing(continued_model, data_loader)
```

```
Test complete in 0m 42s
Acc: 0.7071
```

```
model_name = 'resnet'
test_model = initialize_model(model_name, num_classes, use_pretrained=False)
test_model.load_state_dict(torch.load('./models/res18_best/res18best_weight.pt').state_dict())
test_model = test_model.to(device)
model_testing(test_model, data_loader)
```

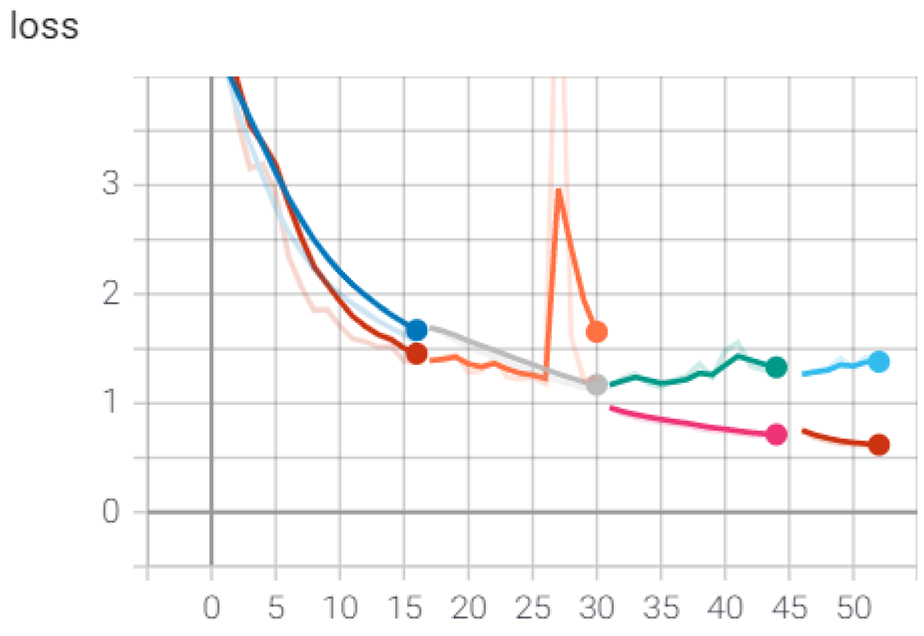
```
Test complete in 0m 19s
Acc: 0.7162
```

In terms of the test result, ResNet18 performs better and faster!

1. ResNet18 is faster because its complexity is smaller than ResNeXt50 naturally.
2. ResNeXt is not trained perfectly. The training overfits in the end. It could still be optimized.

Improve the performance

1. L2-Regularization for overfitting



ResNeXt Overfitting

```
scratch_optimizer  =  optim.Adam(scratch_model.parameters(),  lr=0.001)
```

Add ‘weight_decay=5e-4’ as an argument

Improve the performance

2. Add more data or use stronger and variant data augmentation

E.g., SnapMix

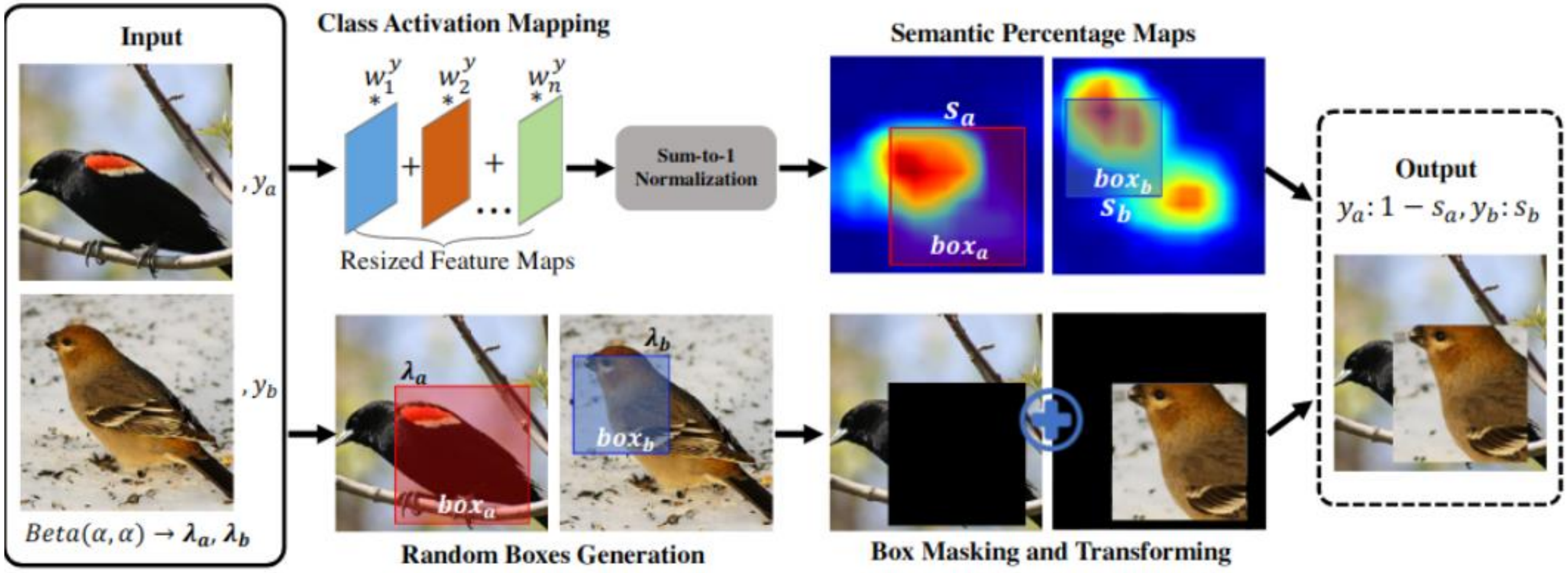


Figure 2: An overview of proposed method.

Improve the performance

3. AutoML for better hyperparameter

Cloud AutoML Vision

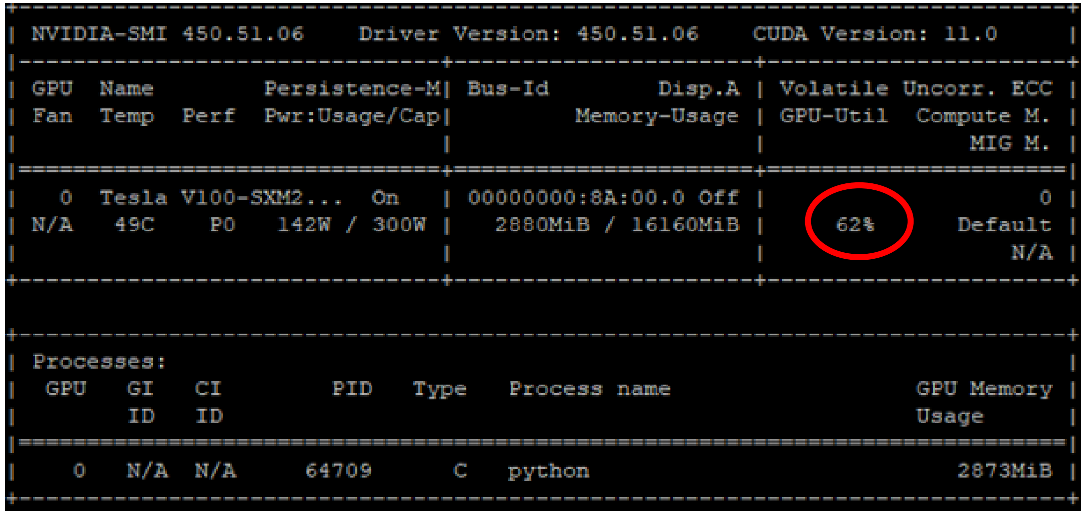
AutoML can assist developers to tune the hyperparameter in an better way.

The most naïve way is to test all of the possible hyperparameters automatically.

We can try to use AutoML to tune ResNeXt. Since its number of hyperparameter is not that large, it would be a good way.

Improve the inference speed

1. GPU Utilization



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:8A:00.0 Off |                    0 |
| N/A   49C    P0   142W / 300W |   2880MiB / 16160MiB |     62%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     64709      C   python                           2873MiB |
+-----------------------------------------------------------------------------+
```

GPU utilization will dramatically affect the inference speed. Bad transfer of data from the host to device can result in low GPU utilization.

1. Utilize prefetch technique to make CPU preprocess the data while waiting for GPU training current batch.

2. Cache those data which don't need to be preprocessed in each epoch in the memory. (Tensorflow has this option, not sure whether Pytorch has a similar function?)

3. Use bottleneck to check which part of the data pipeline is stuck for the longest time.

Improve the inference speed

2. Mixed Precision

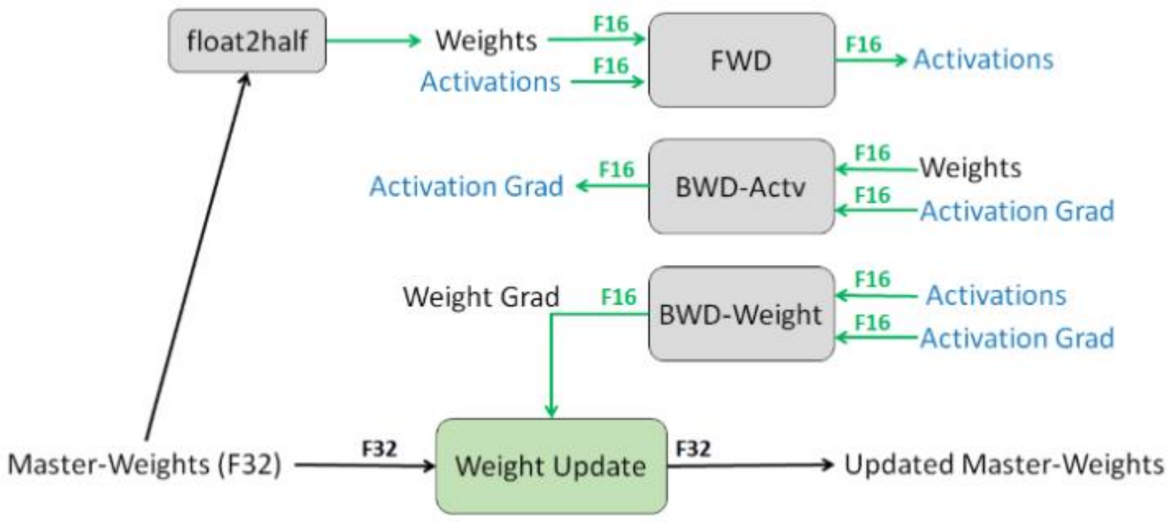


Figure 1: Mixed precision training iteration for a layer.

Use Float16 to replace Float32 during training, saving half of the space and improve the throughput.

1. Pytorch has the option to implement mixed precision conveniently.

2. When enabling mixed precision, remember to adjust the output type of the model layers.