Optimization and Deployment Analysis of Split-Attention Network (ResNeSt)

Xirui Fu
xf620@nyu.edu

Haodong Ge
hg2363@nyu.edu

Abstract

In this work, we first implement 50-Layer Split-Attention Network (ResNeSt50) with original bottleneck structure on classification task using CIFAR10 dataset. To attempt optimizing the model, we innovatively implement split-attention technique on basic block structure. For comparison between two structures, both ResNeSt50 models are trained at different GPU types.Then, a transfer learning of both models to the CIFAR100 dataset is done to further verify comparison results. For the deployment analysis, the winner model from the previous part is converted and deployed in both Pytorch and Caffe2 frameworks. After model inferences, two frameworks are compared from aspects such as test accuracy, inference time and model sizes.

1 Background & Motivation

1.1 ResNeSt Framework

After being created in 2015, The ResNet (residual neural network)(He et al., 2015) has become one of the most important and popular models in recent years. One simple number is able to show its popularity: the original paper is cited over 70K times. Despite the fact that more fresh image classification models were invented in previous years, the importance of ResNet and its variants has never been ignored. Like Zhang et al. (2020) mentioned, "Most downstream applications such as object detection and semantic segmentation are still using ResNet variants as the backbone network because of its simple and modular structure." Thus, in 2020, The ResNeSt model is created on the basis of the ResNet and split-attention idea. There are several major advantages of the ResNeSt model compared to others. Firstly, like Synced (2020) mentioned, the ResNeSt maintains the overall same structure as the ResNet structure. Thus, it can be directly replaced as a backbone network in downstream tasks. Besides, a more direct advantage is that ResNeSt simply outperforms other similar networks like ResNeXt(Xie et al., 2017) and SENet(Hu et al., 2019). Especially, the ResNeSt-50 achieves a top-1 accuracy on ImageNet with significant difference with the runner-up. Therefore, we decided to focus on the possibility to further optimize the ResNeSt-50 model. Normally, the bottleneck structure is used in deeper ResNet models to reduce the number of parameters and matrix multiplications. The tradeoff effect between two structures at 50 layers depth remains uncertain (More detailed explanation and comparison at 3.2.1). Based on the background mentioned above, we decided to implement ResNeSt-50 on both structures for clear comparison.

1.2 Deployment Analysis

During the past decades, smartphones have evolved with unbelievable speed. Thanks to these technology breakthroughs, those mobile applications and functions that people can hardly imagine in the last decade are appearing with an explosion scale. Some examples would be real-time image classification, AR interaction (Pokemon Go) and AI auto conversation. Traditionally, deep learning models rely on GPUs and huge memory sizes and can hardly be implemented on mobile devices. However, along the trend, mobile DL frameworks like Caffe2, CoreML are created to make mobile inference applicable in the industry. The tradeoff between two frameworks is obvious. The mobile DL framework tends to have smaller size and shorter inference time. However, at the same time, the model performance will be possibly compromised after compressing. Through the deployment analysis, we are able to have a better understanding of the difference between DL frameworks and hardware platforms. The analysis result can help people on balance and application scenarios of dif-

ferent frameworks. More broadly, it will facilitate the integration of DL frameworks in industry and establishment of cross-platform DL ecosystem.

2 Solution Diagram

As mentioned above, this project is composed of two cohesive but relatively independent parts: ResNeSt optimization and deployment analysis.

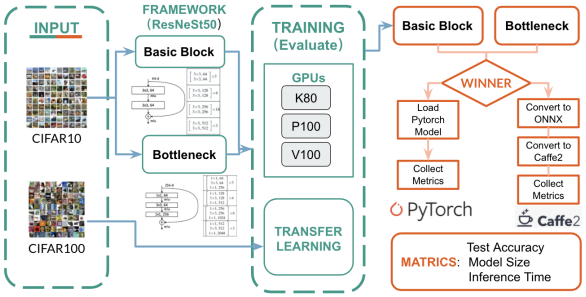


Figure 1: Solution diagram

Before digging into each part of the problem, the solution diagram of the whole picture is necessary to assist understanding the whole context and connection among different parts. Generally, the first part of the project can be summarized to be a contrast between two ResNeSt buildings, structured evaluated by test accuracy in different configurations. On the other hand, the second part is comparison between traditional and mobile DL frameworks measured by accuracy, time and size.

3 ResNeSt Optimization

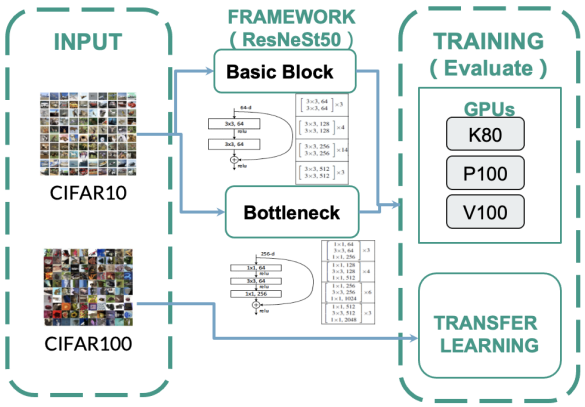


Figure 2: Diagram of ResNeSt Optimization

3.1 ResNeSt Framework

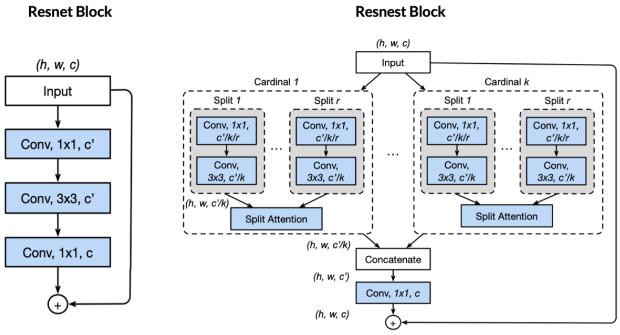


Figure 3: Resnet block and ResNeSt block

Based on the Resnet architecture, ResNest combines the channel-wise attention strategy with multi-path network layout. Figure 3 shows the structures of the two models.

In Resnest blocks, the feature can be divided into several groups, and the number of featuremap groups is given by a cardinality hyperparameter K. In each group, a new radix hyperparameter R is introduced to indicate the number of splits within a cardinal group.

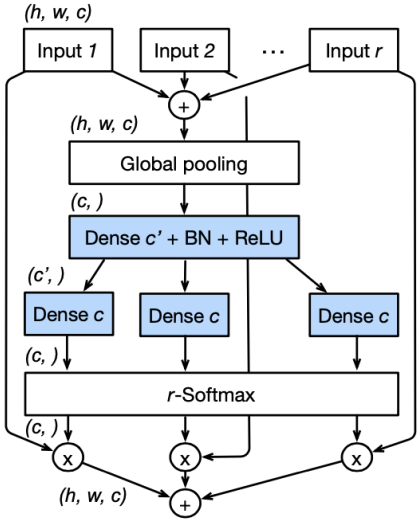


Figure 4: Split attention within a cardinal group

In the split attention, global average pooling across spatial dimensions first gathers the global contextual information. A weighted fusion of the cardinal group representation is then aggregated using channel-wise soft attention, where each featuremap channel is produced using a weighted combination over splits.

Compared with Resnet block, Resnest block is much more complex, since it divides the input into

several groups and in each group implements the split attention technique. However, there are still some similarities between these two models. The Resnest block keeps the convolution layers and kernel size same as the Resnet block.

To summarize, Resnest block maintains the overall structure of Resnet, but introduces featuremap group and split attention to better learn the features of images.

3.2 Basic Block

3.2.1 Basic Block vs Bottleneck

img: basic block and bottleneck structure

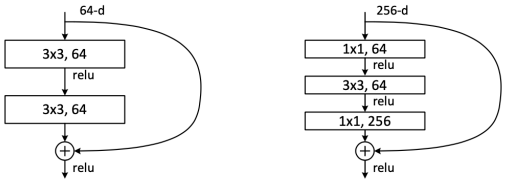


Figure 5: Left: Basic block, Right: Bottleneck

The bottleneck architecture is used in very deep networks due to computational efficiency.

The bottleneck block is composed of 1x1, 3x3, and 1x1 convolution layers, instead of 2 3x3 convolution layers. The 1x1 layers are responsible for reducing and increasing the dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions.

The basic block with larger filters is able to learn more features but has more parameters to train, while the bottleneck need less time to train but may have lower accuracy.

The tradeoff between these two structures changes along with the depth of the model. Basically, deep networks prefer bottleneck due to good computational efficiency. However, the tradeoff is unclear at 50 layers, since this depth is also fine for the basic block.

3.2.2 Basic Block with Split Attention

For the bottleneck, the first 1x1 convolution layer is used to reduce dimension for computation while the second 1x1 convolution layer is used to increase dimension for shortcut.

In the basic block, since there is no need for dimension reduction, the kernel size of the first 1x1 layer can be converted to 3x3 and the second 1x1 layer can be dropped.

Besides, the structure of SE-Net indicates that the output of split attention can be direct input of the shortcut.

Therefore, applying the split attention to the basic block just needs to change the kernel size of first layer and number of filters of the second layer.

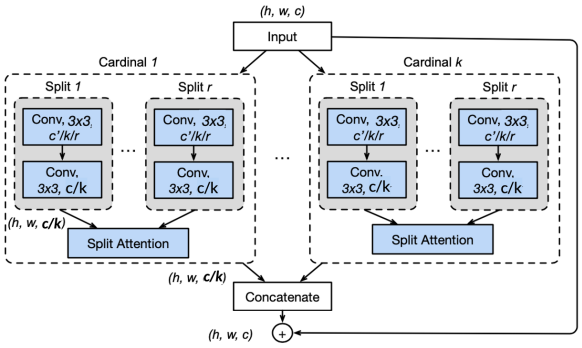


Figure 6: Basic block with split attention

3.3 Implementation Details

In the optimization of Resnest, the two architectures, basic block and bottleneck, are compared and realized by PyTorch on Google Cloud Platform.

3.3.1 Training ResNeSt50 on CIFAR10

Models are trained with CIFAR10 on various GPU types like K80, P100 and V100. During training, the data like test accuracy and training time are collected.

3.3.2 Transfer Learning on CIFAR100

Transfer learning is achieved with CIFAR100 to compare the two structures. This is mainly to explore the performances on different datasets.

The implementation details, like hyperparameters, for ResNeSt50 training and transfer learning are listed in Table 1.

	ResNeSt50 training	Transfer learning
Epoch	250	200
Batch size	128	64
Learning rate	0.1	0.1
LR scheduler	gamma=0.1, milestone=[80, 210]	gamma=0.1, stepsize=60
Criterion	crossEntropy	crossEntropy
Optimizer	SGD	SGD

Table 1: Model implementation details

3.4 Experimental Results

Accuracy		
	Basic block	Bottleneck
K80	0.8862	0.8705
P100	0.8923	0.8742
V100	0.8907	0.8765
Average	0.8897	0.8737
Transfer Learning	0.6323	0.6152

Table 2: Test accuracy

In terms of accuracy (Table 2), basic block outperforms bottleneck on various GPU types as well as transfer learning.

Training Time (s/epoch)		
	Basic block	Bottleneck
K80	320	149
P100	36	31
V100	25.5	22

Table 3: Training time for one epoch

However, from Table 3, the bottleneck structure is more computationally efficient. When training on K80, it only takes half of the time for the basic block, but the gap is getting smaller when training on P100 and V100, with 5 and 3 seconds respectively.

Thus, there is a tradeoff between these two architectures, which is actually a tradeoff between accuracy and computational efficiency. For example, if training on K80, it would be better to choose bottleneck, since it saves much time with little loss of accuracy.

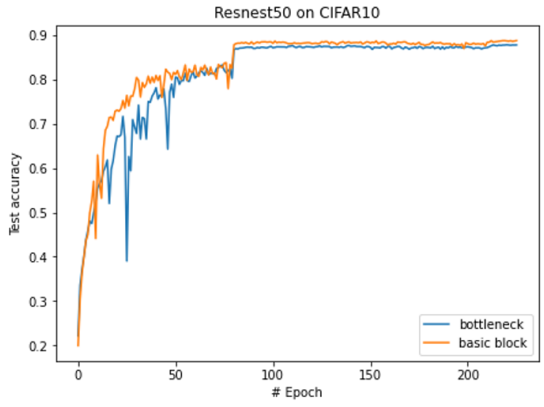


Figure 7: Test accuracy during training

The two plots , ResNeSt50 on CIFAR10 (Figure 7) and transfer learning on CIFAR100 (Figure 8),

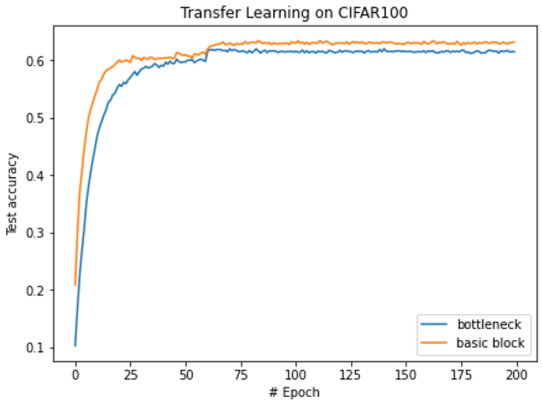
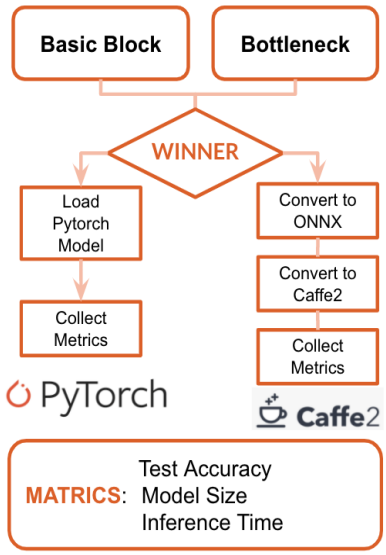


Figure 8: Test accuracy during training

both show that basic blocks ususally have a higher accuracy and converges a little bit faster, since basic blocks with larger filters can learn features better.

4 Deployment Analysis



4.1 Implementation Details

During the deployment analysis, the major framework and dependencies are pytorch, ONNX and Caffe2. To make a comparison between DL frameworks like Pytroch and TensorFlow with mobile ones like Caffe2 and CoreML, ONNX is one of the most important tools during the process. More precisely, according to its inventor, “ONNX is an open format that lets users move deep learning models between different frameworks.” This open format was initially proposed by Facebook and Microsoft but is now a widely accepted industry standard.

Given the pretrained basic-block ResNeSt-50 model from the previous part in Pytorch form, the

first deployment task is to convert it into an ONNX format and then deploy the exported ONNX model using Caffe2. Ideally, the exported Caffe2 model will be able to run on the model device in systems like IOS. However, due to the limited time, we simulated the inference process of the mobile DL framework using ONNX model on Caffe2 backend. (Nayak, 2021)

More specifically, in result, both models take image with size $[32 * 32 * 3]$ as input. The tensor is then converted into a Float32 numpy array and run through the loaded model in Caffe2.The outputs of the model are actually log probabilities. The actual predict score can be obtained after taking exponential function.

4.2 DEMO: Inference on single image

After framework conversion, both pytorch model and Caffe2 model are ready to be deployed. Before running models on the test sets, a single test image of the ship is fed into models. The predict scores are listed below (Table 4). As the table showed, both models made absolute correct classification with over 99% confidence. The prediction scores of the two models are very close with tiny differences less than 10^{-5}.

Predict #	Predict Class	.pt Model Score	.onnx Model Score
1	Ship	0.994803439	0.99480238
2	Horse	0.002122742	0.002120206
3	Dog	0.001315854	0.00131847
4	Truck	0.0011049229	0.0011057975
5	Frog	0.000377598	0.000377779
6	Bird	0.000216783	0.000216705
7	Airplane	4.08936e-05	4.08939e-05
8	Deer	1.4758e-05	1.4758e-05
9	Automobile	2.767e-06	2.7671e-06
10	Cat	2.422e-07	2.422e-07

Table 4: Prediction scores on single image

4.3 Evaluation: Inference on Testsets

There are two different test sets for model inference. The large test set is originally from CIFAR10 dataset with 10K images. On the other hand, the smaller test set contains 50 manually labeled images that are not included in the CIFAR dataset at all. Both test accuracy and average inference time is recorded during the evaluation process. The results are shown in the table below.

	Test_acc small(%)	Avg_time small(s)	Test_acc large(%)	Avg_time Large(s)	Model_size (mb)
Pytorch	0.9	0.05145	0.8923	0.05242	128
Caffe2	0.88	0.02854	0.8812	0.02878	115

Table 5: Evaluation scores of DL frameworks

According to the results(Table 5), the pytorch model achieves slightly higher accuracy but takes longer inference time on both small and large testsets. On the other hand, the caffe2 model has significantly shorter inference time and smaller model size, but slighted compromised performance due to compressing. Generally, Caffe2 model is lighter and faster, and Pytorch model has more powerful performance.

5 Conclusion and Future Work

5.1 Conclusions

To be concise, several conclusions can be summarized based on the results of evaluation. From the comparison between two kinds of ResNet building blocks, it showed that Basic block achieves slightly higher accuracy (around 1.6%), while bottleneck model takes less time to train. Thus, a tradeoff between accuracy and computational efficiency exists between two structures.

On the other hand, from the comparison between traditional and mobile deep learning framework, it showed that Mobile DL framework like Caffe2 still has advantage on model size and significantly shorter inference time than traditional DL framework like pytorch. However, the model performance will be slightly compromised after compressing.

5.2 Limitation & Future Work

Due to the time and resource limit, many possible generalized and more innovative experiments can be tried in both parts of the project. Firstly, for the model framework part, when we train Resnest50 model, we used best configuration in paper. A better alternative could be tunning

split-attention hyperparameters like cardinality and radix. Besides, the model performance is compared on a limited dataset, for better generalizability, model comparison should be implemented on various and larger dataset like ImageNet.

During the deployment phase, Due to the limited time, only Caffe2 model is practiced in this project with simulation of ONNX model on Caffe2 backend.For better realization of deployment comparison, More mobile DL frameworks should be explored like CoreML, TFlite, etc. Furthermore, the analysis would be more comprehensive if models can be implemented on different hardware platforms like different versions of Iphone and Samsung.

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2019. Squeeze-and-excitation networks.

Sunita Nayak. 2021. Pytorch model inference using onnx and caffe2: Learn opencv.

Synced. 2020. Amazon introduces resnest: Strong, split-attention networks.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks.

Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. 2020. Resnest: Split-attention networks.