ST5229 Deep Learning in Data Analytics Group Project

ALENIN DANILA, LI TSZ KA, LU SIQI, WANG CHING

I. Abstract

This project examines the ResNeSt deep learning architecture, a model that builds on previous works like ResNet, ResNeXt, and SE-Net. We explore its core innovation: the Split-Attention block, which cleverly combines grouped convolutions (cardinality) with an attention mechanism across different feature map splits (radix). Through experiments on the CIFAR-10 dataset, including parameter variations (ablation) and comparisons with other networks, we assessed ResNeSt's performance. A transfer learning task was also conducted to test its feature extraction strength compared with ResNet-50. Our findings show that ResNeSt offers competitive accuracy and in particular robust feature learning capabilities which is suitable for downstream tasks, although its design adds complexity compared to simpler residual networks.

II. Introduction and Literature Review

Deep learning has emerged as a novel approach to machine learning. It employs transformation and graph technologies to build multi-layer learning models, and it has obtained outstanding performance across multiple applications, including image classification, audio and language processing, etc.[1] In the field of image classification, Convolutional Neural Networks (CNNs) are one of the most common methods in this domain. The key components are convolutional layers, pooling layers, and fully connected layers. Convolutional layers extract spatial features and channel relationships. Pooling layers perform spatial downsampling to reduce feature map dimensions, decreasing computational complexity while avoiding overfitting. Fully connected (FC) layers transform the high-level features into class probabilities.[2]

LeNet, the first CNN model, was developed by Yann LeCun in the late 1980s. It contains two convolutional layers, two pooling layers with average pooling method, and two fully connected layers.[3] It demonstrated that CNNs can learn hierarchical features from raw pixels. However, LeNet could only use limited depth due to computational constraints at that time, and its simple structure could not provide sufficient performance for complex image patterns. Therefore, AlexNet was created. It introduced the concept of dropout regularization, which involves stochastically deactivating a portion of neurons during each training iteration. ReLU activation function was introduced to mitigate the vanishing gradient problem. These improvements not only reduce computational complexity but also prevent overfitting.[4]

With the improvement of computational resources, extremely deep networks became the trend. However, this might lead to severe vanishing gradient, where gradient signals exponentially diminish as they propagate through deep networks during backpropagation, and degradation problem, where adding more layers made the network perform worse. Residual Networks (ResNet), introduced in 2015, aim to solve this issue by implementing identity shortcut connections, enabling the network to learn residual mappings.[5] These connections skip over some layers, adding the block's input directly to its output. This creates a residual block. Instead of learning the output directly, the network learns the difference (the residual) between the input and the desired output. This simple idea had a huge impact, helping solve the vanishing gradient and degradation problems by making it easier for gradients to flow through the network. ResNet allowed training networks with hundreds or even over a thousand layers, getting much better accuracy and making residual learning a standard technique for later CNNs.

Around the same time, the Inception architecture (GoogLeNet), developed around 2014-2015[6], explored using multiple processing paths in parallel to make networks more efficient and better at learning features. Instead of processing data through one path, Inception modules used several convolutional filters (e.g. 1x1, 3x3, 5x5) and a pooling path, all working in parallel within a single block. The outputs from these paths were then concatenated. This "split-transform-merge" approach let the network capture features at different sizes at the same time and break down the work, improving results without adding too much computation or too many parameters. Inception shows that how wide the network is and how its parts are arranged are also important to learning good features, not just how deep it is.

Combining ideas from ResNet (residual learning) and Inception (multiple paths), ResNeXt (Aggregated Residual Transformations) was developed in 2017[7]. ResNeXt introduced cardinality as another way to improve models, meaning the number of identical processing paths running in parallel within a residual block. It used grouped convolutions in a structured way following the split-transform-merge idea. The input channels were divided into several groups (the number of groups is the cardinality), each group went through the same set of operations (e.g. 1x1 - 3x3 - 1x1

convolutions), and their outputs were concatenated together before the residual connection was applied. ResNeXt showed that adding more paths (increasing cardinality) improved accuracy more effectively than just making the network deeper or wider, getting better results with similar complexity to ResNet. It provided a clear, organized way to use multiple paths inside a residual block.

While ResNeXt assigns each channel the same weight and treats every feature with equal importance, Squeeze-and-Excitation Networks (SE-Net) were developed in 2017-18[8] and can be seen as a lightweight mechanism to improve CNN performance. SE-Net introduced small "Squeeze-and-Excitation" (SE) blocks that could easily be added to other networks. First, an SE block uses global average pooling (GAP) to summarize spatial information for each channel. Then, it uses two fully connected layers and a sigmoid function to learn weights for each channel. These weights adjust the importance of each channel in the feature map, boosting useful features and reducing the impact of less useful ones. Paying attention to how channels relate to each other gave significant improvements with little extra computation. SE-Net can be plugged into existing architectures and help residual learning by focusing on inter-channel relationships. SE-Net won the ILSVRC 2017 competition (SENet-154 reached top-1 accuracy), showing how important attention mechanisms could be.

Taking attention ideas further, SK-Net (Selective Kernel Networks) was developed in 2019[9], based on ideas from SE-Net. SK-Net created a flexible attention system that lets the network choose information from convolution kernels of different sizes within a single layer. An SK block sends the input through several parallel paths with different kernel sizes. The outputs are combined using an attention method, which uses channel information generated in a way similar to that of SE-Net (GAP followed by FC layers) to decide how much weight to give each path's output. This lets the network adjust its view (receptive field size) based on the input content for each channel. SK-Net showed attention could do more than just re-weight channels; it could also choose how to combine information from different processing paths, hinting at the more advanced attention methods used later.

In this report, we focus on a new ResNet model, which can be referred to as *Split-Attention Network* (ResNeSt). It was introduced by Zhang et al. in 2020. It is a recent evolution that combines ResNeXt and SE-Net, incorporating the cardinality concept and the attention mechanism into a split-attention block. In these blocks, the residual connection is inherited from ResNet, the input is split into multiple cardinal groups as in ResNeXt, and an attention mechanism similar to SE-Net is used to reweigh the features within each group before aggregation.[10]

III. Methods

Here we will explain the central component of ResNeSt - the *Split-Attention* block, which is composed of the *featuremap group* and the *split attention* operations. We take a historical approach here, since the methods adopted in ResNeSt follow closely those of its predecessors.

A. Featuremap Groups

1) Grouped Convolution and Cardinality: ResNeSt makes use of grouped convolution, which is the idea of splitting the input into multiple branches for convolutional transformation before merging them back to give a single output. The input is often split evenly across all branches, with each branch involving the same set of convolutions. The outputs of all branches are then concatenated or summed up to form the block output. A major advantage of such operation is an improvement in computational efficiency. Consider the dimension of input to be split in a grouped convolution. The input size is n_i and the output size is n_o. A full convolution will lead to a computational cost of $\mathcal{O}(n_i n_o)$. Suppose the inputs are split into g groups of convolutions. The computational cost will then be reduced to $\mathcal{O}(g(n_i/g)(n_o/g)) = \mathcal{O}(n_i n_o/g)$. Namely, there is a g-time reduction in computational cost. In addition to that, grouped convolution also allows parallel distribution between multiple GPUs without a heavy cost to memory, since each GPU can run on its own [10].

ResNeSt uses the grouped convolution architecture of ResNeXt [7], which is a multi-branch extension of ResNet. Here the inputs are split across channels into g groups, where g is called the cardinality of the grouped convolution. Despite the computational efficiency gain, there is a downside to the grouped convolution approach - the interdepency among channels across groups is lost. ResNeXt tries to fix this by including 1×1 convolutions before and after each grouped convolution. See the right figure in Fig. 1 for the implementation. Since the 1×1 convolutions are implemented over all input or output channels, they can capture the interdependency over all of them. However, if the input or output dimensions are left unchanged, the computational cost for each 1×1 convolution will be $\mathcal{O}(c^2)$, where c is the number of input or output channels. To reduce computational cost, a bottleneck is formed by reducing the channel dimension of the main convolutions to $b < c$. This in turn has the effect of reducing the computational cost of the grouped convolution to $\mathcal{O}(b^2/g)$ [11]. The bottleneck design is also helpful in training. It forces the network to learn a more compressed and concise representation of the image in the channel dimension, and reduces the model complexity by keeping the number of parameters at bay when increasing the depth of the network [5].

ResNeSt [10] adopts a variant of the above configuration. Instead of an overall 1×1 convolution before the grouped convolution with input channel size c and output channel

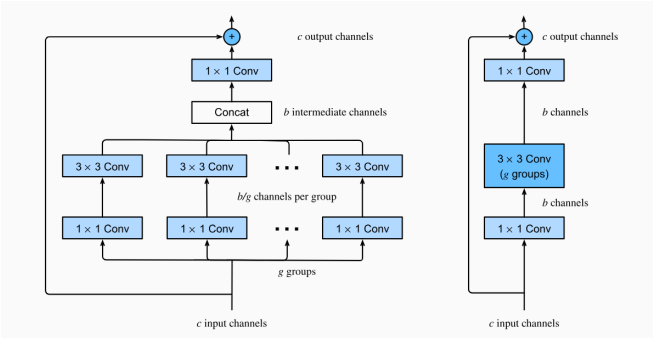

Fig. 1: The block architecture for ResNeXt [11]. The right figure shows the construction based on grouped convolution sandwiched between 2 1×1 convolutions. The left figure shows the modified but equivalent version adopted by ResNeSt.

size b, the 1×1 convolution is moved inside each branch, keeping the input channel size as c while reducing the output channel size to the branch size b/g. This implementation is shown in the left figure of Fig. 1. Note that the two implementations are computationally equivalent.

2) Radix Featuremap Groups: The Inception block [6] introduced the idea of transforming the same input through multiple different parallel branches before concatenating them into a single output. The branches use kernels of different sizes to vary the receptive fields and capture spatial correlations of different scales. The Selective Kernel (SK) convolution in SKNet [9], a predecessor of ResNeSt, adopts the same idea. Unlike the grouped convolution, the branches consist of convolutions with the same input. Moreover, they represent transformations over different spatial resolutions rather than over different channels. Such branches in the ResNeSt block are termed radix featuremap groups with radix number r [10]. When $r = 1$, ResNeSt is equivalent to SENet. When $r = 2$, ResNeSt is equivalent to the simplest form of SKNet. (Fig. 2)

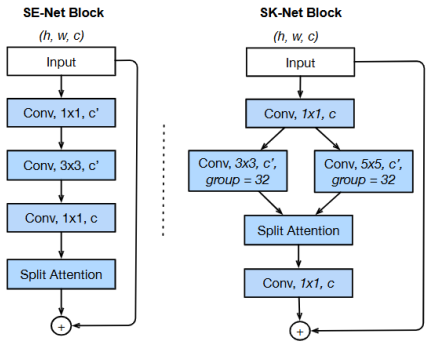

Fig. 2: Left: The SENet block equivalent to $r = 1$ in ResNeSt. Right: The SKNet block equivalent to $r = 2$ in ResNeSt [10].

3) ResNeSt Specifics: The above two types of branching are combined to form a ResNeSt block. The input is divided into K cardinal groups, which are subdivided into R radix splits, so the total number of featuremap groups is $G = KR$. A bottleneck 1×1 convolution is applied at the beginning of each featuremap group. Note that in the ResNeSt paper [10], the author did not vary the kernel sizes in different radix splits. This arrangement goes against the original motivation of SKNet [9]. Nevertheless, varying kernel sizes can easily be implemented in ResNeSt.

B. Split Attention

1) Squeeze-and-Excitation (SE) in SENet: The SE mechanism in SENet [8] is in essence the split attention used in ResNeSt in its simplest form. Here, there are no radix or cardinal featuremap groups. The mechanism is about which channel to pay attention to. In the *squeeze* operation, a global average pooling is applied to each channel to "squeeze" the spatial information into a channel descriptor, which provides the contextual summary statistics of the spatial features in each channel. Mathematically, for a SE block input $\mathbf{U}$ with a size $H \times W \times C$ where H, W and C are the height, width and the number of channels respectively, the channel descriptor $\mathbf{s} \in \mathbb{R}^C$ is

$$s_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{U}_c(i, j) \ ,$$

where c is the channel index. The *excitation* operation is about assigning a weight to each channel using the channel descriptor $\mathbf{s}$. A large weight *excites* that particular channel. The weight is calculated by a bottleneck mechanism to limit model complexity as well as to force the algorithm to learn a more concise and generalizable pattern across channels. Firstly, through a full-connected layer with learnable weights $\mathbf{W} \in \mathbb{R}^{\frac{C}{r} \times C}$, $\mathbf{s}$ is transformed to a smaller size C/r, where the hyperparameter r is the reduction ratio. The ReLU function δ, which provides a gating mechanism, is then applied to give a compact feature $\mathbf{z}$.

$$\mathbf{z} = \delta(\mathbf{W}\mathbf{s}) \ .$$

Secondly, the compact feature $\mathbf{z}$ is transformed back to the original size C by the trainable weights $\tilde{\mathbf{W}} \in \mathbb{R}^{C \times \frac{C}{r}}$ with a sigmoid activation σ to obtain the final weights $\mathbf{a} \in \mathbb{R}^C$ for each channel c.

$$\mathbf{a} = \sigma(\tilde{\mathbf{W}}\mathbf{z}) \ .$$

The output of the squeeze-and-excitation block $\mathbf{V} = (\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_C)$ where

$$\mathbf{V}_c = a_c \mathbf{U}_c \ .$$

This mechanism is a form of self-attention as no input from a target is involved.

2) Selective Kernel in SKNet: As the immediate predecessor of the split attention mechanism in ResNeSt, the Selective Kernel (SK) convolution in SKNet uses a channel-wise attention mechanism similar to that of SENet [9]. However, each radix featuremap group has its own weights for the channels, so selection is essentially done over radix groups. To this end, the outputs from the

convolutional transformations of all radix branches are first fused together through an element-wise summation. The output then undergoes a global average pooling and the a bottleneck fully connected transformation as before. There are some differences in the case of SKNet. Firstly, the compact feature $\mathbf{z}$ is obtained through an additional batch normalization $\mathcal{B}$.

$$\mathbf{z} = \delta(\mathcal{B}(\mathbf{W}\mathbf{s})) \ .$$

Secondly, the dimension reduction is carried out with an additional hyperparameter L which controls the minimum value of the output channel dimension d.

$$d = \max(C/r, L) \ .$$

Lastly, the final weights are obtained for every radix branch i and channel c through a softmax function.

$$\mathbf{a}_{ic} = \frac{\exp(\tilde{\mathbf{W}}_{ic}\mathbf{z})}{\sum_{j=1}^{R} \exp(\tilde{\mathbf{W}}_{jc}\mathbf{z})} \ .$$

The output of the SK convolution block is the weighted sum of $\mathbf{U}$ across radix branches i for each channel c.

$$\mathbf{V}_c = \sum_{i=1}^{R} a_{ic}\mathbf{U}_{ic} \ .$$

3) Split Attention in ResNeSt: ResNeSt uses virtually the same split attention mechanism in SKNet. Here, it is emphasized that the channel weights for a radix group i can be different for each cardinal group k. (Fig. 3) The final weights $\mathbf{a}_{kic}$ are obtained through the sigmoid function as in SENet when the total number of the radix featuremap groups $R = 1$, and the softmax function as in SKNet when $R > 1$.

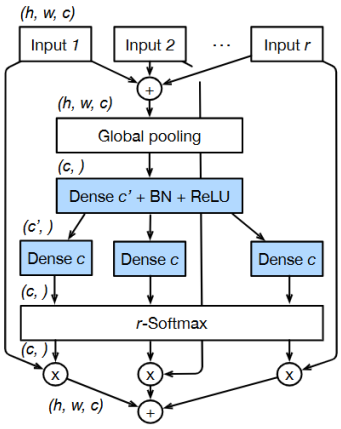

Fig. 3: Split attention in ResNeSt [10]. Dense c refers to a fully connected layer with c output channels. $c' < c$ is chosen to form a bottleneck.

In terms of block architecture, there are two different implementations [10]. In the *cardinality-major implementation* (Fig. 4), radix splits within the same cardinal group reside next to each other. The split attention mechanism is applied across radix splits within each cardinal group. The outputs are then concatenated. In the *radix-major implementation* (Fig. 5), radix splits with the same radix number but different cardinality are placed next to each other. The outputs from the same radix branch of each cardinal group are concatenated first before applying split attention across all radix branches. Note that a SKNet setup using the ResNeXt architecture backbone is similar to the radix-major implementation [9]. The ResNeSt authors used the radix-major version in the real implementation.

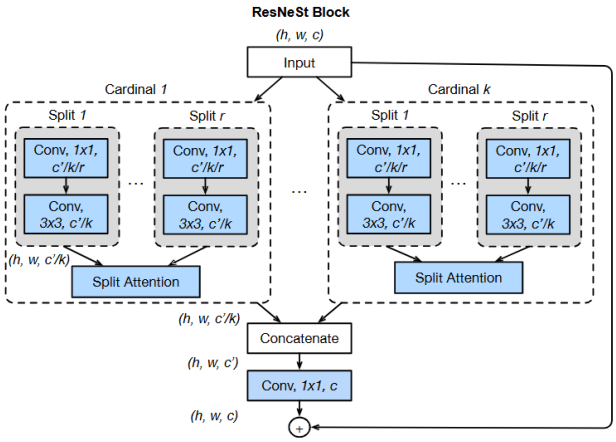

Fig. 4: The cardinality-major implementation in ResNeSt [10].

IV. Experiments

We conducted three experiments, mirroring those in the referenced article, to investigate the architectural properties of ResNeSt. First, we performed an ablation study on ResNeSt variants with varying parameters (radix, cardinality, and network width), analogous to Table 3 in [10]. Second, we conducted a comparative study between ResNeSt and other networks to assess its effectiveness in balancing accuracy and latency. Finally, we implemented a transfer learning task to evaluate the capability of the pretrained ResNeSt and ResNet-50 as a backbone for a specific vision task.

A. Ablation Study

To examine the impact of different parameters on the network, we employed a minimal network architecture. While the original article used a [3,4,6,3] layer configuration for ResNeSt-50, we opted for a [1,1,1,1] configuration, corresponding to ResNeSt-14 according to the article's naming convention, calculated as 14=2+3×(1+1+1+1), excluding the final three output layers. This results in 17 total layers. This represents the smallest feasible network given that the source code requires at least one layer per block (setting the parameter to 0 is therefore not an option). Similar to the original study, we began with ResNet-14 (where the article used a radix block implementation by default; we also experimented with standard residual blocks by setting radix=0). We then compared this baseline with ResNeSt-14 variants featuring different radix, cardinality, and network width. Following the article's notation, we

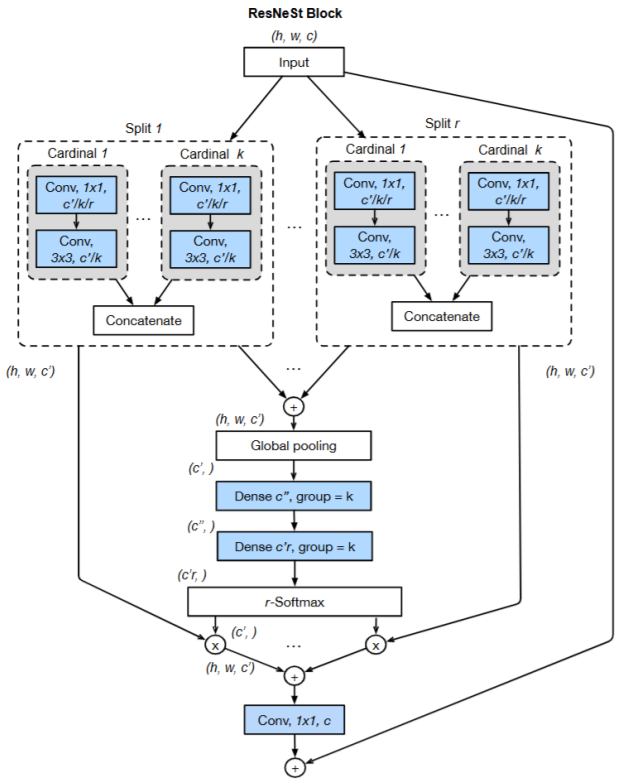

Fig. 5: The radix-major implementation in ResNeSt [10].

named our variants in the format ResNeSt-14-1s1x64d, where 's' denotes radix, 'x' cardinality, and 'd' network width. '0s' indicates the use of a standard residual block, akin to ResNet-D.

We would like to discuss more on the difference between the ResNet-14-0s1x64d and ResNeSt-14-0s1x64d variants, as this distinction is not explicitly detailed in [10]. We highlight three key architectural differences. First, the stem layers differ significantly. ResNet-14 employs a single 7x7 convolutional layer with 64 output channels, a stride of 2, and padding of 3, followed by batch normalization, a ReLU activation, and a 3x3 max pooling layer. In contrast, ResNeSt-14 utilizes a sequence of three 3x3 convolutional layers, each followed by batch normalization and a ReLU activation. These layers are as follows: 3 input channels to 32 output channels (stride 2, padding 1), 32 input channels to 32 output channels (stride 1, padding 1), and finally, 32 input channels to 64 output channels (stride 1, padding 1). Second, ResNeSt-14 incorporates an average pooling layer with a kernel size equal to the stride before the 1x1 convolutional layer. Third, in the first residual block of ResNeSt-14, a 3x3 average pooling layer with a stride of 1 and padding of 1 is applied before the second convolutional layer within that block.

Experiments environment is set on an Nvidia T4 GPU in Google Colab. Due to our limited computational resources, we used the CIFAR-10 dataset with a batch size of 256 instead of ImageNet, and without image augmentation. The original optimizer settings is retrained as an SGD optimizer with a weight decay of 0.0001 and 0.9 of momentum. Training was limited to 20 epochs, and we applied a dropout rate of 0.2, consistent with the article, to mitigate the significant overfitting observed in our initial trials. The results are summarized in the following table.

Variant	Parameters	Latency (ms) / Throughput (img/sec)	Accuracy
ResNet-14-0s1x64d	8.03M	0.1223 / 8177	58.08%
ResNet-14-1s1x64d	8.21M	0.1311 / 7624	58.68%
ResNeSt-14-0s1x64d	8.06M	0.1716 / 5826	62.96%
ResNeSt-14-1s1x64d	8.23M	0.1282 / 7797	61.21%
ResNeSt-14-1s1x40d	5.44M	0.1435 / 6967	61.22%
ResNeSt-14-1s2x64d	13.63M	0.1307 / 7646	62.95%
ResNeSt-14-2s1x64d	8.58M	0.1545 / 6471	62.64%

TABLE I: Comparison of ResNet and ResNeSt-14 variants in terms of parameters, latency, throughput, and accuracy.

From the table, we can draw the following conclusions: Regarding latency and throughput, with similar network architecture and number of parameters, ResNet and ResNeSt achieved comparable results. Employing the radix block appeared to enhance the throughput on ResNeSt but slightly decrease it on ResNet. Increasing radix or cardinality, or decreasing network width in ResNeSt, generally led to a slight reduction in throughput. In terms of accuracy, all ResNeSt variants outperformed the ResNet baseline. The use of the radix block seemed to have a slightly negative effect on ResNet's accuracy, potentially due to random variation. For ResNeSt, using a radix of 1 resulted in a decrease in accuracy compared to the standard block; however, this loss was mitigated by using a radix of 2. This suggests a complex relationship between accuracy and radix, or it could again be attributable to random fluctuations. Altering the network width did not noticeably influence accuracy, while increasing cardinality from 1 to 2 improved performance.

Hypothesize that the interplay between network architecture/hyperparameters and performance metrics (throughput/accuracy) is intricate, and our study did not encompass all possible combinations. Given the relatively small performance differences observed, the possibility of random variation contributing to the results can not be ignored. Furthermore, the substitution of CIFAR-10 for ImageNet, which differs significantly in image size, dataset size, and number of classes, may have influenced the observed trends.

B. Accuracy-Latency Trade-off with Other Networks

The original article investigates the accuracy-latency trade-off by comparing ResNeSt with EfficientNet [12], operating under the premise that more complex networks generally exhibit higher latency and accuracy. To explore this relationship in our context, we experimented with various hyperparameter configurations of ResNeSt alongside

EfficientNet (b0 to b3) and a smaller baseline network, ResNet-18 [5]. The results of this comparison are presented in the following table:

Network	Parameters	Latency (ms) / Throughput (img/sec)	Accuracy
ResNeSt-14	8.58M	0.1808 / 5530	61.68%
ResNeSt-32	16.56M	0.1761 / 5678	59.04%
ResNeSt-50	25.45M	0.1894 / 5277	59.45%
ResNeSt-101	46.14M	0.2670 / 3745	58.18%
ResNet-18	11.18M	0.1247 / 8016	58.55%
EfficientNet b0	4.02M	0.1450 / 6668	50.72%
EfficientNet b1	6.53M	0.1692 / 5908	46.57%
EfficientNet b2	7.72M	0.1703 / 5871	43.23%
EfficientNet b3	10.71M	0.2181 / 4584	41.80%

TABLE II: Comparison of network performance and accuracy.

The data in the table reveal an unexpected trend in our experiments: an increase in network complexity did not correlate with an improvement in accuracy. In fact, larger ResNeSt and EfficientNet variants exhibited lower accuracy compared to smaller ResNeSt-14. Given that the anticipated positive correlation between accuracy and latency was not observed in our findings, we are unable to make a meaningful trade-off plot similar to Figure 1 in [10].

We attribute this deviation from the expected behavior to several factors. First, the replacement of the ImageNet dataset by CIFAR-10 most likely played a significant role. CIFAR-10, with its smaller image size and fewer classes, might not sufficiently challenge the representational capacity of these larger networks within the limited 20 training epochs. Second, 20 epochs may be insufficient for the larger networks to converge and learn the effective features in the dataset. Finally, the optimal performance of complex networks like EfficientNet often necessitates extensive hyperparameter tuning, particularly for the optimizer and the regularization strategies, which was beyond our computational capabilities for this study. Besides, the latency and throughput difference is only obvious for large network.

Despite the absence of a clear accuracy-latency trade-off in our results, we can still conclude that, across all tested model sizes, the ResNeSt variants consistently achieved higher accuracy than the EfficientNet models. Although the model complexities of ResNeSt as seen in the number of parameters are significantly higher in most cases, ResNeSt-14 and EfficientNet b2 have relatively comparable number of parameters and the former still performs better in terms of accuracy. This suggests that, even with limited training and a smaller dataset, the architectural innovations in ResNeSt provide a benefit in terms of accuracy compared to EfficientNet.

C. Transfer Learning

In order to evaluate the feature extraction capabilities of ResNeSt, we conducted a fine-tuning experiment comparing ResNeSt-50 with ResNet-50. Both models were initialized with weights of the model pretrained on the ImageNet dataset. For our downstream task, we utilized the Hotdog dataset from the textbook Dive into Deep Learning (D2L) [11]. In accordance with the method in the D2L, we modified the final fully connected output layer of both models to accommodate the change in the number of classes from 1000 (ImageNet) to 2 (Hotdog).

Standard normalization and data augmentation techniques is applied to the Hotdog dataset. The models were fine-tuned using the SGD optimizer with a small learning rate of 5×10^{-5} for the layers present in the pretrained model and a learning rate that is 10 times as large for the final fully connected layer. The weight decay was set to 0.001. The model was trained for a total of 5 epochs.

Following the fine-tuning process, we evaluated the performance of both models on the validation set. ResNeSt-50 achieved a validation accuracy of 0.950, while ResNet-50 attained an accuracy of 0.907. This result suggests that ResNeSt-50 serves as a more effective feature extractor compared to ResNet-50, underscoring the potential of ResNeSt architectures as robust backbones for various downstream vision tasks.

V. Conclusions

In this report, we explore the ResNeSt architecture and its primary innovation. The Split-Attention block integrates the cardinality concept from ResNeXt with an attention mechanism inspired by SE-Net. Experiments of model comparison were conducted. They demonstrate that ResNeSt submodels exceed the ResNet baseline on CIFAR-10, with ResNeSt-14-2s1x64d achieving 62.64% accuracy compared to ResNet-14's 58.08%. Scaled-down experiments using CIFAR-10 could not fully capture the information with limited computational resources, but we can still conclude that ResNeSt models have better performance than EfficientNet models in general. Transfer learning methods were used to justify the feature extraction capabilities of the ResNeSt model, and it achieved 95% accuracy on the Hotdog dataset versus ResNet-50's 90.7%. The results indicate that despite the extra architectural complexity and computational resource requirements of Split-Attention blocks, ResNeSt represents an outstanding improvement in deep learning, especially for transfer learning applications where robust feature extraction is critical, making it a strong choice for backbone networks in various deep learning tasks.

VI. Relevance, Significance, and Limitations

ResNeSt is notable for successfully integrating distinct architectural components into one effective system. Its key innovation, the Split-Attention block, combined multi-path execution similar to ResNeXt with channel attention inspired by SE-Net. This approach resulted in significant performance improvements: ResNeSt achieved leading results on various

benchmarks, often matching or exceeding contemporary models like EfficientNet.[12] ResNeSt became a preferred backbone for object detection and segmentation due to its strong performance and manageable computational requirements. The research community quickly adopted ResNeSt[13], driven by its advantageous balance of high accuracy and practical processing speed.

Despite all the advantages of ResNeSt, it still has limitations. ResNeSt's design is more complex than standard residual networks. Its Split-Attention blocks include additional tunable hyperparameters, such as the number of splits (radix) and groups (cardinality). Determining the optimal values for these parameters typically requires extra experimentation, which can make the network harder to fine-tune efficiently. The increased complexity within Split-Attention blocks results in higher computational and memory requirements. These blocks involve additional operations such as pooling, attention calculations via fully connected layers, and softmax functions, along with managing multiple parallel feature pathways. Although ResNeSt generally provides good accuracy relative to its computational cost, increasing hyperparameters such as the radix number directly raises processing time and memory consumption.[5][10]While ResNeSt demonstrates effective feature transfer for standard computer vision tasks similar to its training data, its suitability for distinctly different data types may be limited. The architecture was primarily designed for 2D image analysis. Adapting it effectively to other data formats, such as 3D, video, or non-visual data, is difficult and could demand considerable modifications, though not impossible.

VII. Future Directions and Extensions

Attention mechanisms are highly impactful in computer vision [14]. ResNeSt makes an effective use of this approach. Further progress could still be made by refining its architecture and deepening our theoretical understanding. Integrating spatial attention mechanisms alongside the existing channel attention could create more powerful blocks capable of focusing on relevant image regions, benefiting tasks such as semantic and instance segmentation. The Split-Attention concept could also be incorporated into Neural Architecture Search (NAS)[15] to automatically discover optimal configurations or layer-specific attention needs. Exploring hybrid models that combine ResNeSt's convolutional strengths with the global context modeling of Vision Transformers[16] is another promising path. A deeper theoretical investigation into why Split-Attention is effective could lead to a simplified, but equally powerful, block design or inspire entirely new approaches to feature representation and adaptive computation.

ResNeSt's strong foundation presents numerous opportunities for practical application and enhancement. Potential areas include video analysis (using 3D ResNeSt variants for action recognition)[17], medical imaging (3D scans), remote sensing (using channel attention for multi-spectral data), etc. Addressing its computational complexity is crucial. Developing lightweight or efficient variants (e.g., "Mobile-ResNeSt")[18] using techniques like channel reduction, depthwise separable convolutions, or knowledge distillation could make ResNeSt's accuracy accessible to resource-constrained devices.

VIII. Source Code

The source code for the implementation and experiments conducted in this project is published on GitHub at our repo.

IX. Team Member Contribution

Team Member	Contributions
Alenin Danila	Write the Intro & Lit Review, the Significance & Limitations, as well as the Extensions Section in the report.
Li Tsz Ka	Write the Methods Section of the report. Help run a small part of the experiments. Proofread the report.
Lu Siqi	Write the codes and the Experiments part of the report.
Wang Ching	Write the Intro & Lit Review and Conclusions Sections of the report. Help review and run the experiment. Tidy up and standardize the report.

TABLE III: Summary of team member contributions

References

[1] Z. J. H. A. J. A.-D. A. D. Y. A.-S. O. . . F. L. Alzubaidi, L., "Human-level control through deep reinforcement learning," *ournal of big Data*, vol. 8, pp. 1–74., 2021.

[2] Y. B. LeCun, Yann and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] B. L. B. Y. . H.-P. LeCun, Y., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86(11), pp. 2278–2324, 1998.

[4] S. I. . H. G. E. Krizhevsky, A., "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, p. 25, 2012.

[5] Z. X. R. S. . S.-J. He, K., "Deep residual learning for image recognition," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[7] G. R. D. P. T. Z.-. H. K. Xie, S., "Aggregated residual transformations for deep neural networks.," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[8] S. L. . S. G. Hu, J., "Squeeze-and-excitation networks," *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[9] X. Li, W. Wang, X. Hu, and J. Yang, "Selective kernel networks," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 510–519, 2019.

[10] W. C. Z. Z. Y.-L. H. Z. Z. . . S. A. Zhang, H., "Resnest: Split-attention networks," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2736–2746, 2022.

[11] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai.

[12] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019.

[13] MMDetection Contributors, "OpenMMLab Detection Toolbox and Benchmark." https://github.com/open-mmlab/mmdetection, 2019–.

[14] M.-H. Guo, T.-X. Xu, J.-J. Liu, Z.-N. Liu, P.-T. Jiang, T.-J. Mu, S.-H. Zhang, R. R. Martin, M.-M. Cheng, and S.-M. Hu, "Attention mechanisms in computer vision: A survey," *Computational Visual Media*, vol. 8, no. 3, pp. 331–368, 2022.

[15] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[17] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 6299–6308, 2017.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.