ResNet Revisited: An Empirical Discovery on Storage Efficient Image Classification Models

Hao Tu
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
h4tu@ucsd.edu

Abstract

Since the introduction of AlexNet, modern deep learning for image classification has trended toward the pursuit of increasingly complex and parameter-dense models for higher accuracy (Krizhevsky et al, 2012), presenting significant barriers to entry due to the substantial computational and disk resources required. This trend has not only resulted in a gatekeeping effect for capable researchers with limited resources but also limits the scope of applications that can benefit from the latest advancements in artificial intelligence. However, highly popularized projects like tinyML with its adaptation in IoT has benefited thousands of small businesses and millions of users who utilize deep learning on affordable devices daily (Ma et al, 2024). Recognizing the critical need for democratizing access to deep learning technologies, this study introduces a novel approach to model structure discovery focused on storage efficiency. Through empirical analysis, I systematically explore various convolutional neural network architectures. Using the classic ResNet as a starting point, I identify configurations that enhance model accuracy while maintaining the model size. Applied to a customized CIFAR10 dataset, our experiments identify the model architectures that provide competitive accuracy all within a parameter budget that adds less than 1% to the original model size. The findings underscore the potential of model structure discovery as a pathway to more accessible and sustainable AI solutions, as well as presenting the practicality of deploying advanced deep learning models in resource-constrained environments.

1 Introduction

In the ever-evolving landscape of deep learning, the quest for the next breakthrough often leads us down a path of 'the bigger the better'. This phenomenon was especially prevalent during the explosive growth of ResNet-inspired architectures. Most state of the art models were so large and parameter-dense that they're pushing the limits of what our hardware can handle. Those that focus on 'doing more with less' were few and far between.

While giants like ResNet have dominated the arena with their impressive performance, a closer look reveals a curious trend: not all progress hinges on pumping up the parameter count. SqueezeNet, for instance, merges techniques like delayed downsampling and fire modules, similar to ResNet's bottleneck approach that reduces channels for added depth, as well as an innovative mix of 1x1 and 3x3 convolutions (Iandola et al, 2016). Or consider the strides made by MobileNet that utilizes depthwise convolution with a pointwise convolution to significantly reduce parameter count, which have pushed the envelope in balancing model size with accuracy, advocating for a leaner and a more efficient approach to image classification and object detection (Howard et al, 2017). These innovations raise a crucial question — how do these techniques individually contribute to a model's success, especially when we're trying to keep the scale of disk space consumption in check? This

brings us to the heart of what this paper aims to uncover. By focusing on a base model characterized by two residual blocks followed by two fully connected layers, I systematically explore variations of key techniques, such as delayed downsampling and the selective use of fully connected layers, depthwise convolution, bottleneck residual block, among others. By doing so, I aim to contribute to a more nuanced comprehension of model architecture optimization with limited resources, proving success in advancing deep learning is possible within a constrained environment utilizing storage efficient model design.

2 Method

2.1 Dataset and Optimization Strategy

In this study, I employed a predetermined set of consistent variables across all experiments, including dataset, optimization method, scheduling, regularization, amount of training epochs, and the size of each training batch. Those variables have been carefully chosen based on common best practices as well as my personal experiences on fine-tuning basic CNN models for the regular CIFAR-10 dataset.

2.1.1 Dataset Customization and Augmentation

Beyond the conventional use of the CIFAR-10 dataset, I introduce a customized version that applies focused augmentation on the following four categories: birds, cats, deer, and dogs. The choice of these categories stemmed from historical data indicating that most models, regardless of their complexity, struggle with accurately classifying images within these groups. To address this problem I apply horizontal flips, random rotations, and color jittering to introduce variety to the original dataset.

2.1.2 Mini-batch training and Optimizer

To introduce stochasticity into the training process, I opted for a mini-batch size of 4, a choice that aligns with the dataset's size and complexity. It is also crucial to select an optimizer that can universally perform stochastic gradient descent but also (1) introduces moving averages that help the model accelerate the movement through flat regions of the loss landscape (2) overcomes the saddle points efficiently with accumulated momentum. The most effective optimizer that achieves both of those is Adam, which has proven to be the default optimization strategy to most training in state of the art research (Kingma and Ba, 2014).

2.1.3 Training Parameters

The Adam optimizer is configured with a learning rate of 5e-4, and a weight decay of 1e-4, the latter serving as a mathematical equivalent to L2 regularization according to PyTorch official documentation. Given the CIFAR-10 dataset's property of inducing overfitting in models easily, even those with relatively simple architectures, the inclusion of weight decay helps mitigate this tendency. Furthermore, I opt for a ReduceLROnPlateau learning rate scheduler with patience of 4 epochs, a strategic choice designed to adjust the learning rate based on the validation accuracy of each epoch. This scheduler is included as a safeguard against potential stagnation in learning; however, given the short span of 10 epochs per experiment and my chosen initial learning rate, it is anticipated that this mechanism will not be triggered, and the learning rate will remain at 5e-4.

2.2 Model Architecture

In this exploration, I anchor the experiments on a base model derived from the ResNet architecture, tailored to include two blocks followed by two fully connected layers, as detailed in Table 1. This configuration employs max pooling after each of the two blocks before routing the data through to the fully connected layers. With precisely 136,380 learnable parameters, the base model stands at a modest size of 0.5218MB. The core challenge is to iteratively integrate a variety of architectural innovations inspired by SqueezeNet, MobileNet, DenseNet, and ResNet into models. These models aim to remain within a 2% margin of the base model's size, yet strive to significantly enhance accuracy or reduce inference time per image.

Table 1: Base ResNet Architecture

Operation	Description	Output Size
Input	CIFAR-10 Image	$3 \times 32 \times 32$
ResBlock 1	3 to 10 channels, stride 1	$10 \times 32 \times 32$
ReLU Activation		
Max Pooling	Kernel Size 2, Stride 2	$10 \times 16 \times 16$
ResBlock 2	10 to 20 channels, stride 1	$20 \times 16 \times 16$
ReLU Activation		
Max Pooling	Kernel Size 2, Stride 2	$20 \times 8 \times 8$
Flatten		$20 \times 8 \times 8$ to 1280
Fully Connected Layer 1	1280 to 100 neurons	100
Batch Normalization		
ReLU Activation		
Fully Connected Layer 2	100 to 10 neurons	10 (output)

2.2.1 Building Blocks

There are several building blocks that form the foundation of the model variations: SkipConn2d (figure 1), ResBlock (figure 2), FireResBlock (figure 3), DepthwiseResBlock (figure 4), and BottleneckResBlock (figure 5). A pivotal addition across all these blocks is the integration of skip connections that deviate from the original implementation in SqueezeNet and MobileNet. This ensures that each block is not only leveraging the unique features presented by depthwise separation and fire modules but also maintains a consistent basis for comparison through the use of skip connections. Further, to counteract covariate shift, I implemented batch normalization after every convolutional layer, following the principles presented by the foundational batch normalization paper (Ioffe and Szegedy, 2015).

2.2.2 Implementation of Model Variations

Building upon the defined blocks, I present the implementation of model variations, each designed to test a specific architectural hypothesis. For ease of reference and clarity in our comparative analysis, we adopt a versioning scheme to name each model variation:

1. Base ResNet Model with Delayed Max Pooling (Model V1): Utilizes the original architecture but introduces a delay in the max pooling operation
2. DenseNet-Inspired Model (Model V2): By incorporating one skip connection into the base ResNet model, effectively transforming it into a DenseNet configuration due to our model's two-block structure
3. Squeeze-Inspired ResNet Model (Model V3): Replaces the standard ResBlock with FireResBlock, adding four additional layers without increasing the parameter count, thanks to the squeezing nature of these blocks
4. Bottleneck ResNet Model (Model V4): Swaps the ResBlock for BottleneckResBlock, a variant similar to the FireResBlock but without channel-wise concatenation
5. Depthwise ResNet Model (Model V5): Inspired by MobileNet, this model replaces the ResBlock with DepthwiseResBlock, allowing for up to 8 layers with a limitation of 20 channels each for a taller architecture
6. Efficient ResNet Model (Model V6): Simplifies the architecture to one fully connected output layer following global average pooling across feature maps, significantly reducing parameter count and enabling the addition of both height and width to the model

Each model is carefully curated to test the effects of architectural decisions on the base model's performance, without straying far from its original size and parameter count as shown in Table 2. Through this systematic approach, we aim to uncover insights into how each building block influences a model's accuracy and inference speed.

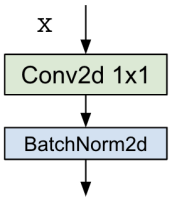

Figure 1: SkipConn2d

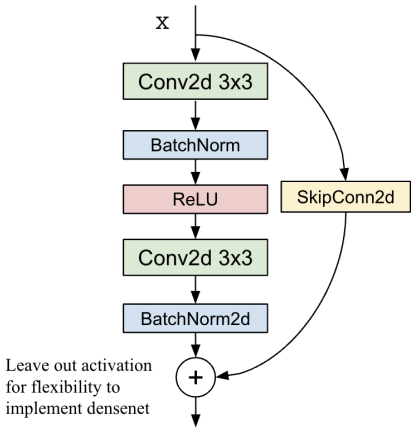

Figure 2: ResBlock

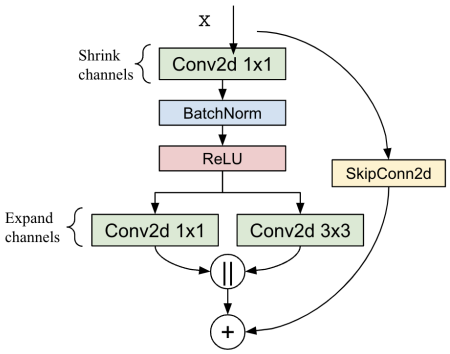

Figure 3: FireResBlock

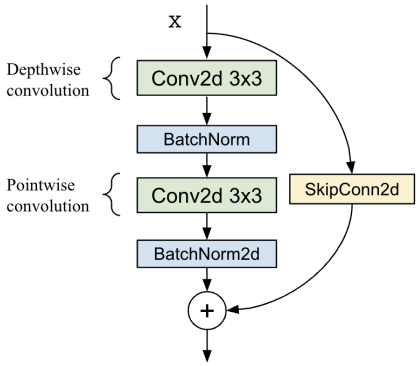

Figure 4: DepthwiseResBlock

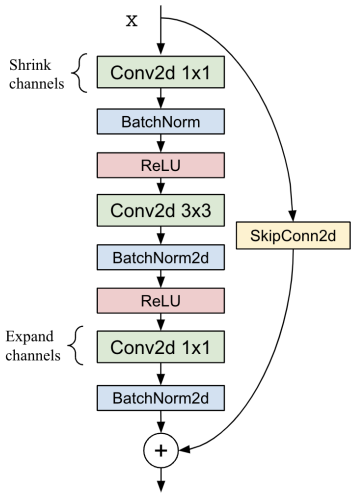

Figure 5: BottleneckResBlock

Table 2: Comparison of Model Parameters and Size

Model	Parameters	Size (MB)	% Change in Size
Base	136,380	0.5218	-
V1	136,430	0.5222	$8.55 \cdot 10^{-2}\%$
V2	136,500	0.5224	0.12%
V3	136,354	0.5229	0.22%
V4	136,536	0.5239	0.41%
V5	137,066	0.5271	1.02%
V6	136,960	0.5268	0.97%

3 Experiment

3.1 Baseline Performance with ResNet

To establish a baseline for our experiments, I plotted the training and validation loss of the base ResNet model as shown figure 6. This curve tells us that the predetermined settings for the model are appropriate, as the training and validation loss converge at a similar rate. One might argue that the model is slightly overfitting, as the validation loss is slightly higher than the training loss. However, given the small margin between the two, it is safe to assume that the model is generalizing well to the dataset, which is shown in the accuracy curve in figure 7. After 10 epochs, this model was able to achieve an accuracy of 73.09% on the validation set with an average inference time of 0.408 ms per image. These two metrics serves as a benchmark for the subsequent model variations.

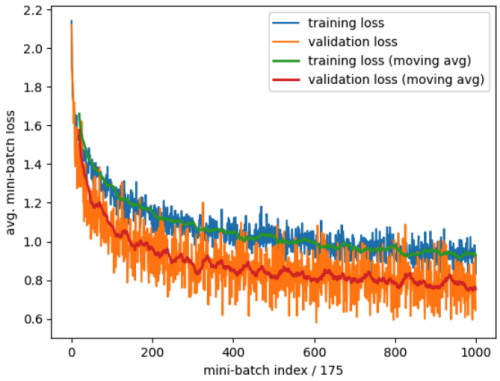

Figure 6: Training and Validation Loss of Base ResNet Model

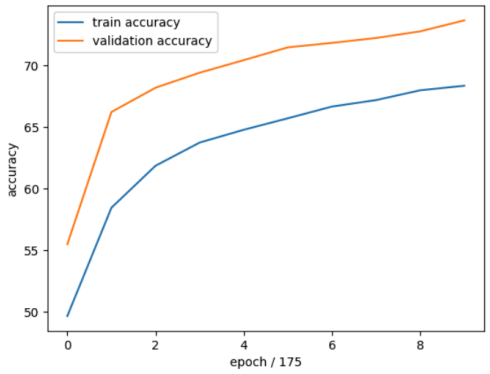

Figure 7: Training and Validation Accuracy of Base ResNet Model

3.2 Model V1 - Base ResNet with Delayed Max Pooling

3.2.1 Motivation

Model V1 tests the impact of delayed max pooling, inspired by SqueezeNet's approach to preserving spatial information for potentially improved training results (Iandola et al, 2016). This model isolates the effect of delayed downsampling, without the fire module, to evaluate its standalone contribution to performance.

3.2.2 Results

Contrary to expectations, Model V1 demonstrated a validation accuracy of 72.78%, which is a decrease from the base model's 73.09%. With a model size slightly increased to 0.522 MB and an average inference time per image of 0.755 milliseconds, the results suggest no significant improvement, potentially indicating a decrease in efficiency.

3.2.3 Observations

The experiment revealed a critical insight: reducing channel counts in early layers to delay downsampling, within the tight parameter constraints of our model, did not lead to the expected performance gains. This suggests that in compact models, preserving channel density early on might be more crucial for maintaining or improving accuracy than preserving feature map size.

3.3 Model V2 - DenseNet-Inspired Model

3.3.1 Motivation

The motivation behind Model V2 is to investigate the impact of incorporating a DenseNet-like skip connection into a compact ResNet framework, evaluating its potential to enhance model performance through improved feature propagation in a small-scale network (Huang et al, 2017).

3.3.2 Results

Model V2 reports a validation accuracy of 73.71%, slightly surpassing the base model's 73.09%, with an incremental model size increase to 0.5224MB. The average inference time slightly rose to 0.414 milliseconds from the base model's 0.408 milliseconds.

3.3.3 Observations

The experiment reveals that integrating a dense connection offers a modest accuracy improvement with minimal impact on the model size and a slight increase in inference time. This suggests that dense connections can be beneficial for small networks, enhancing performance while maintaining a balance between efficiency and computational cost.

3.4 Models V3 & V4 - Channel Efficiency via Convolutional Transformations

3.4.1 Motivation

Both model V3 and V4 aim to assess the impact of compressing and then expanding channel densities through 3x3 convolutions, with V3 utilizing a concatenation of 1x1 and 3x3 convolutions for expansion (Iandola et al, 2016), while V4 adopts a sequence of 1x1 convolutions to shrink, followed by 3x3, then another 1x1 convolution to expand (He et al, 2016). This comparative exploration seeks to understand how these nuanced approaches to channel manipulation influence overall model performance, accuracy, and efficiency.

3.4.2 Results

- Model V3 (FireResBlock) presented a model size of 0.523MB, achieving a validation accuracy of 70.39%, with an average inference time of 0.781 milliseconds.
- Model V4 (BottleneckResBlock) slightly increased in size to 0.5239MB, with a reduced validation accuracy of 70.10% and a longer average inference time of 1.192 milliseconds.

3.4.3 Observations

V3, despite its slightly better accuracy, demonstrates the potential of fire module-based transformations to maintain more efficient inference times while minimally impacting the model size. Conversely, V4's approach, while similar in premise, results in a slight decrease in accuracy and a noticeable increase in inference time, suggesting that the method of channel expansion and compression significantly affects both the computational cost and model efficacy. In short, although FireResBlock may offer a more balanced compromise between model size, accuracy, and efficiency than the sequential expand-collapse-expand strategy of the BottleneckResBlock, both models do not show improvements over the base model, indicating that in a constrained environment, the cost of width may not be justified by the height gains.

3.5 Model V5 - Depthwise Separable Convolutions

3.5.1 Motivation

Model V5 delves into the efficiency of depthwise separable convolutions, a technique inspired by MobileNet, aiming to streamline parameter usage by applying filters on a per-channel basis (Howard et al, 2017). This approach hypothesizes that reducing parameter count through depthwise operations could allow for a significant increase in the network's depth, therefore increasing the model's capacity to learn complex features.

3.5.2 Results

Model V5 presents a validation accuracy of 68.03%, with a model size of 0.527MB and an average inference time per image of 1.029 milliseconds.

3.5.3 Observations

The shift to depthwise separable convolutions in Model V5 led to a more complex network within the parameter budget. However, this complexity did not translate to improved accuracy, which saw a decrease compared to the base model. The results indicate that while depthwise separable convolutions offer a path to increasing network depth without a proportional increase in parameters, the sacrifice on cross channel interactions may not be justified in a compact model.

3.6 Model V6 - Streamlining with Global Average Pooling

3.6.1 Motivation

Model V6 introduces a significant architectural modification by eliminating one of the fully connected layers, alongside its batch normalization and activation function, opting instead for a singular linear layer directly following global average pooling. Inspired by the original ResNet architecture, the modification seeks to test the hypothesis that simplifying the network's end stages—specifically by reducing the parameter-heavy fully connected layers can allow for an expanded network architecture ("taller" and potentially "wider") within the same parameter budget, as shown in Table 3 (He et al, 2016).

3.6.2 Results

The restructuring in Model V6 yielded an impressive validation accuracy of 80.67%, marking a substantial improvement over the base model. This was achieved with a model size of 0.527MB and an average inference time per image of 0.848 milliseconds.

3.6.3 Observations

The results from Model V6 underscore the considerable impact of fully connected layers on the parameter budget of CNNs. By removing one such layer and adopting global average pooling to directly feed into one fully connected output stage, Model V6 demonstrates that it is possible to significantly enhance model accuracy while maintaining a tight control over model size and computational demand. The discovery is grounded on the realization that additional fully connected layer in our base model consumes a large amount of parameters (exactly 128,100), which contrasts with the approach of global average pooling leading to only 500 parameters for the final classification stage. With the network having both width and depth, we can see from figure 8 and figure 9 that not only the model converges faster but also generalizes better to the validation set.

3.7 Model Alpha - Combining Global Average Pooling with Dense Connections

3.7.1 Motivation

As a result of the experiment, the concluding hypothesis is that the effect of shrinking channel density and preserving feature map size are not as beneficial given the parameter budget. Based on this insight, I propose a hybrid model (Table 4) that combines the global average pooling from Model V6 with the dense connections from Model V2. This model, named Model Alpha, aims to leverage the

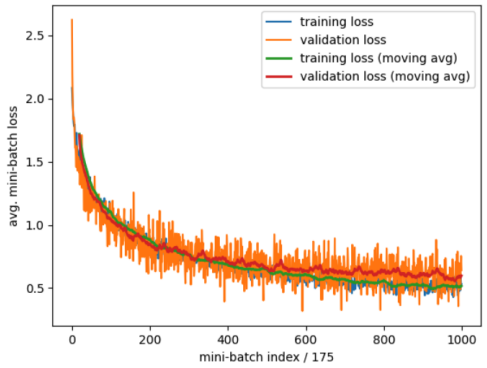

Figure 8: Training and Validation Loss of Model V6

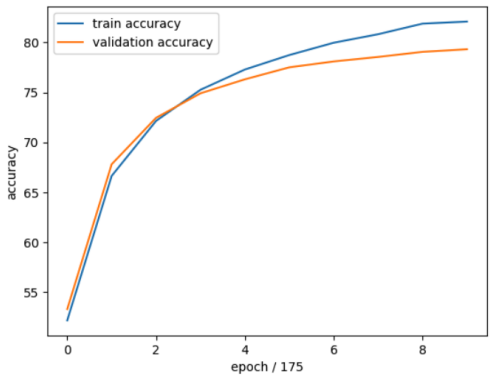

Figure 9: Training and Validation Accuracy of Model V6

Table 3: Model V6 Architecture

Layer Type	Operation	Output Size
Input	CIFAR-10 Image	$3 \times 32 \times 32$
ResBlock 1	ResBlock(3, 10, 1) + ReLU	$10 \times 32 \times 32$
Pooling 1	MaxPool2d(2, 2)	$10 \times 16 \times 16$
ResBlock 2	ResBlock(10, 20, 1) + ReLU	$20 \times 16 \times 16$
Pooling 2	MaxPool2d(2, 2)	$20 \times 8 \times 8$
ResBlock 3	ResBlock(20, 30, 1) + ReLU	$30 \times 8 \times 8$
ResBlock 4	ResBlock(30, 40, 1) + ReLU	$40 \times 8 \times 8$
ResBlock 5	ResBlock(40, 50, 1) + ReLU	$50 \times 8 \times 8$
ResBlock 6	ResBlock(50, 50, 1) + ReLU	$50 \times 8 \times 8$
Flatten	View	50×64
Global Avg Pool	Mean(dim=-1)	50 (channels)
Fully Connected	Linear(50, 10)	10 (output classes)

benefits of both techniques to achieve a higher accuracy while maintaining a similar model size and inference time.

Table 4: Model Alpha Architecture

Layer	Operation	Output Size
Input	CIFAR-10 Image	$3 \times 32 \times 32$
ResBlock 1	ResBlock (3, 10, 1) + ReLU	$10 \times 32 \times 32$
Pooling 1	MaxPool2d(2×2)	$10 \times 16 \times 16$
ResBlock 2	ResBlock (10, 20, 1) + SkipConn2d + ReLU	$20 \times 16 \times 16$
Pooling 2	MaxPool2d(2, 2)	$20 \times 8 \times 8$
ResBlock 3	ResBlock (20, 30, 1) + Multiple SkipConns + ReLU	$30 \times 8 \times 8$
ResBlock 4	ResBlock (30, 40, 1) + Multiple SkipConns + ReLU	$40 \times 8 \times 8$
ResBlock 5	ResBlock (40, 40, 1) + Multiple SkipConns + ReLU	$40 \times 8 \times 8$
ResBlock 6	ResBlock (40, 50, 1) + Multiple SkipConns + ReLU	$50 \times 8 \times 8$
Flatten	View	50×64
Global Avg Pool	Mean (dim=-1)	50 (channels)
Output	Linear (50, 10)	10 (output classes)

3.7.2 Results

Model Alpha showcases a notable advancement in model performance, achieving a validation accuracy of 81.04%. This represents a significant improvement over the base model and all other variants explored in this study. The model operates with 131,970 parameters and maintains a

compact size of 0.512MB, demonstrating efficient use of the parameter budget. However, the average inference time per image is 1.400 milliseconds, indicating a moderate increase in computational demand compared to the base model and some earlier variants.

3.7.3 Observations

Model Alpha illustrates a successful balance between model depth, parameter efficiency, and computational performance. The reduction in model size, alongside an increase in accuracy, underscores the effectiveness of strategic architectural choices in achieving high performance within tight parameter constraints. However, the extended inference time suggests a trade-off, where the gains in accuracy and parameter efficiency come at the cost of slightly slower inference speeds. This raise a concern for adapting dense connection in a model that is already deep and wide, as the additional connections may lead to a more complex network that requires more computational resources to process.

4 Conclusion

Drawing conclusions from the model comparisons shown in Table 5, it becomes evident that conventional practices such as reducing channel density while maintaining feature map resolution, a strategy often effective in larger, more complex networks, do not necessarily translate well to lightweight models. This is highlighted by the performance of Models V3 and V4, which, despite the implementation of such strategies, did not achieve significant gains in accuracy and experienced longer inference times.

Table 5: Model Comparisons

Model	Validation Accuracy	Inference Time (ms)	Size (MB)	Parameters
Base Model	73.09%	0.408	0.522	136,380
Model V1	72.78%	0.755	0.522	136,430
Model V2	73.71%	0.414	0.5224	136,500
Model V3	70.39%	0.781	0.523	136,354
Model V4	70.10%	1.192	0.5239	136,536
Model V5	68.03%	1.029	0.527	137,066
Model V6	80.67%	0.848	0.527	136,960
Model Alpha	81.04%	1.400	0.512	131,970

The results suggest that for compact models, the benefits of preserving spatial resolution at the cost of channel density do not outweigh the cost, especially when considering the balance between accuracy and inference speed. Model V5's attempt to deepen the network architecture within a parameter-restricted environment also faced challenges, resulting in the lowest accuracy among the variants. This further underscores that strategies successful in scaling down larger models may require adaptation to be effective in smaller-scale architectures.

In contrast, Model Alpha achieved the highest validation accuracy of 81.04% with the fewest parameters, signifying a well-tuned balance of model dimensions and capacity. However, the model's increased inference time suggests that the additional complexity introduced by dense connections may not be as beneficial in a model that is already deep and wide, especially in application limited by computational resources such as embedded systems or mobile devices.

The scatter plot shown in figure 10 captures these dynamics, with Model Alpha distinctly positioned as the accuracy leader but with long inference time, while Model V6 offers a robust combination of high accuracy and efficiency. With different applications requiring varying trade-offs between accuracy, model size, and inference speed, the insights from this study provide a roadmap for designing storage-efficient models that can be tailored to specific use cases, as well as highlighting the potential for architectural innovations to drive advancements in the field of compact deep learning models.

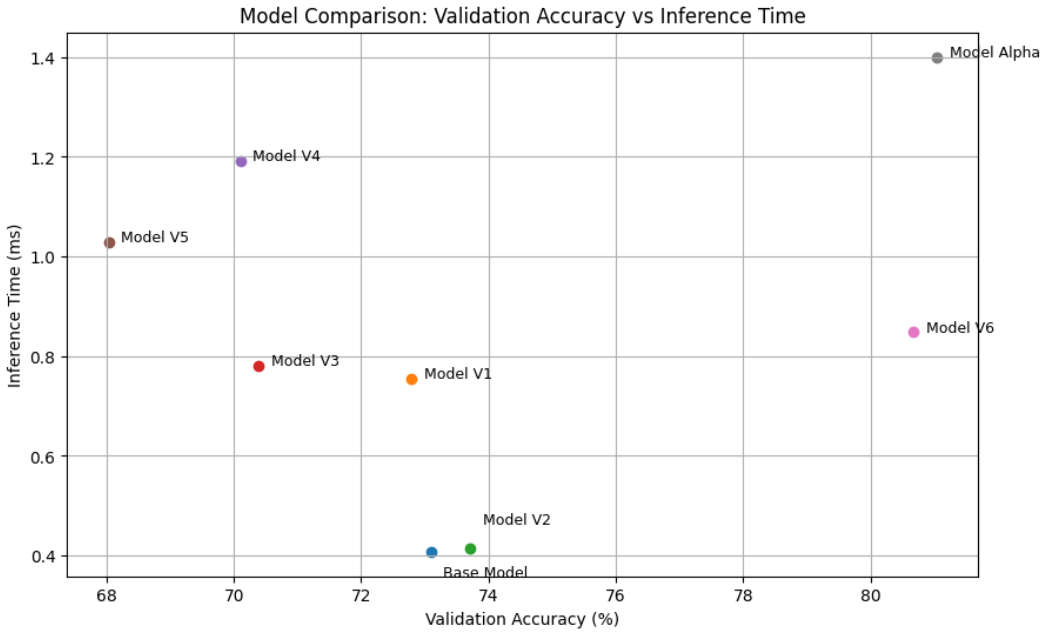

Figure 10: Accuracy and Inference time plot of size-equivalent model variations

References

[1] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[2] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[3] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[4] Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." arXiv preprint arXiv:1602.07360 (2016).

[5] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. pmlr, 2015.

[6] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[7] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

[8] Perez, Luis, and Jason Wang. "The effectiveness of data augmentation in image classification using deep learning." arXiv preprint arXiv:1712.04621 (2017).

[9] Schizas, Nikolaos, et al. "TinyML for ultra-low power AI and large scale IoT deployments: A systematic review." Future Internet 14.12 (2022): 363.