# Efficient Image Classification with Bottleneck-Augmented VGG Networks: A Comparative Study on MNIST and CIFAR-10

Mariia Zimokha
ec24117@qmul.ac.uk
Student ID: 240129639

## 1. Introduction

Deep convolutional neural networks (CNNs) have achieved remarkable success across a range of visual tasks, largely due to their ability to learn hierarchical feature representations from data. Among the most influential architectures, VGG networks demonstrated that stacking small 3×3 convolutional layers in deep, uniform structures could yield strong performance on benchmarks such as ImageNet [12]. However, while effective, VGG architectures suffer from high parameter counts and computational demands, making them impractical for deployment in resource-constrained environments.

Subsequent research focused on improving the efficiency of deep models. Residual Networks (ResNets) [5, 6] introduced skip connections to mitigate the degradation problem, enabling the training of much deeper networks. ResNet further employed bottleneck blocks—compact 1×1–3×3–1×1 structures—to reduce channel dimensionality, significantly decreasing computational cost without compromising representational power.

Recent research has explored the integration of bottleneck into other architectures to achieve a better balance between accuracy and efficiency [1, 3]. Inspired by these developments, this paper investigates whether bottleneck strategies can be effectively adapted to simplify and compress VGG-style networks without introducing the complexities of re-parameterization or multi-branch structures.

This work addresses the following research questions: **(1)** Can bottleneck blocks be effectively integrated into a VGG-style architecture to reduce model size and computational cost without significantly compromising classification accuracy?; **(2)** How does the performance of the bottleneck-augmented VGG (VGG_BN) compare to standard VGG and ResNet architectures across different datasets (MNIST and CIFAR-10)?

To answer these questions, we propose **VGG_BN**, a bottleneck-augmented VGG variant, and conduct comprehensive experiments to evaluate its performance and efficiency.

## 2. Related Work

### 2.1. Residual Networks (ResNet)

Residual Networks (ResNets), introduced by He et al. [5], addressed the degradation problem in very deep networks by introducing skip connections (Figure 1). These connections allow gradients to bypass intermediate layers during backpropagation, enabling the stable training of deep architectures and preserving features from earlier layers.

A key advancement in ResNet was the introduction of the *bottleneck block*, which uses a 1×1–3×3–1×1 structure to reduce and then restore feature dimensions. This design cuts FLOPs by approximately 30% compared to stacking plain 3×3 convolutions [6], and enabled networks with over 100 layers to be trained efficiently.

### 2.2. Visual Geometry Group Networks (VGG)

VGG networks, developed by Simonyan and Zisserman [12], use deep stacks of uniform 3×3 convolutional layers with ReLU activations and with periodic 2×2 max-pooling (Figure 2). VGG-16 (13 convolutional layers) and VGG-19 (16 convolutional layers) became standard backbones due to their simplicity and

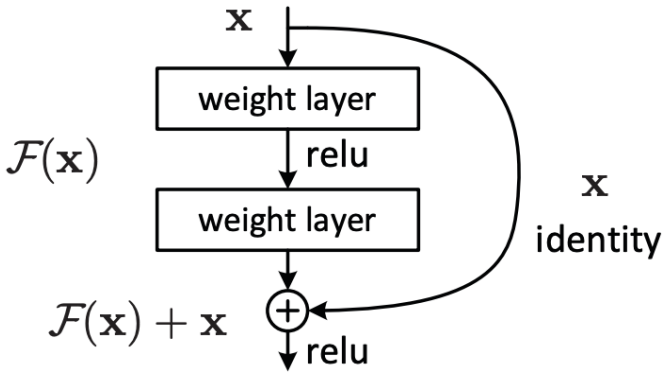



Figure 1. A typical residual block in ResNet, showing the skip connection.

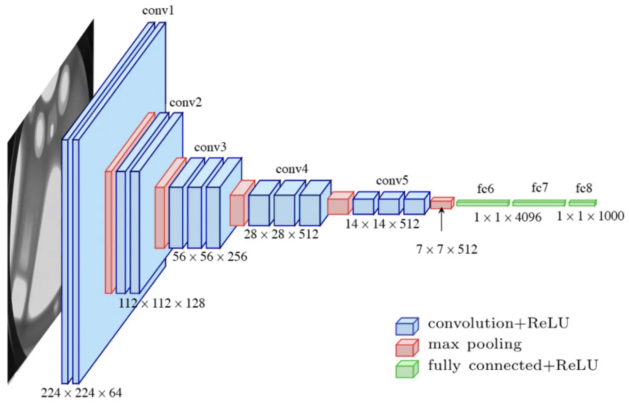



Figure 2. Overview of the VGG-16 convolutional neural network architecture, showing its stack of convolutional layers, pooling operations, and final fully connected layers. Adapted from Huyen's post.

strong representational capacity. However, VGG models are parameter-heavy (e.g., 14.7M parameters for VGG-16) and computationally expensive, making them less practical for deployment [11]. Additionally, they tend to overfit on smaller datasets such as CIFAR-10. Prior compression efforts focused on pruning [4] or low-rank approximations [8], which reduce parameters post-training but require iterative fine-tuning.

In contrast, bottleneck blocks enable *end-to-end* efficiency by learning compact representations during training. While pruning removes "unimportant" weights heuristically, bottlenecks systematically constrain channel dimensions where VGG stacks multiple convolutions (e.g., Conv2-x layers). This architectural distinction motivates our integration of bottlenecks as a native compression mechanism for VGG.

### 2.3. Extensions Combining VGG Structures with Bottlenecks and Residuals

Several recent works have sought to combine VGG-like depth with bottleneck and residual designs, improving both efficiency and accuracy:

- **ResNeXt** [13] extended ResNet by using grouped convolutions inside residual blocks, inspired by VGG's repeating structure.
- **PyramidNet** [3] introduced additive bottlenecks that gradually increase feature dimensions across layers, combining VGG-like uniformity with progressive width scaling.
- **RepVGG** [2] proposed a re-parameterization strategy where training-time multi-branch structures (including 1×1 convolutions) collapse into a plain VGG-like architecture at inference, achieving both residual benefits during training and inference efficiency.

These works demonstrate the potential of integrating bottleneck structures and residual connections into simpler, highly efficient convolutional networks.

### 2.4. Proposed method: VGG with Bottleneck blocks (VGG_BN)

Building on these insights, we propose **VGG_BN** to combine VGG's hierarchical feature extraction with ResNet-inspired bottleneck efficiency. The implementation of **VGGBottleneckBlock** is provided in the Appendix. This block encourages the model to encode the most important information in a lower-dimensional space, discarding less informative features in a three-stage squeeze–expand pattern:

1. **Reduce**: 1×1 convolution to reduce channels by a factor of $r$ (default $r$=4)
2. **Conv**: 3×3 convolution (padding=1)
3. **Expand**: 1×1 convolution to restore original channels

The final VGG_BN architecture consists of a sequence of these bottleneck blocks (applied where the base VGG configuration has consecutive convolutional layers), followed by an average pooling layer and a fully connected layer for classification. Standard convolutional layers with batch normalization and ReLU are used when a convolutional layer in the base VGG is followed by a max-pooling layer. This design choice leads to parameter efficiency, reduced FLOPs, and scalability, which are empirically validated on MNIST and CIFAR-

10 benchmarks. In VGG-16-BN, uneven groups of conv layers leave some single 3×3 convolutions unbottlenecked, yielding approximately 6.3M parameters. By contrast, VGG19_BN has only even-length convolution groups, so every pair is bottlenecked, reducing its parameter count further to 1.25M. This gives VGG19_BN fewer parameters.

## 3. Experiments

### 3.1. Datasets and Preprocessing

All data are shuffled with a batch size of 128. This batch size was chosen as a trade-off between training speed and gradient estimate accuracy.

#### 3.1.1. MNIST

The MNIST dataset consists of 70,000 grayscale images of handwritten digits (60,000 training, 10,000 test) at *28x28* pixels resolution [10]. They have been resized to 32×32 with bilinear interpolation, normalized with mean 0.1307 and std 0.3081, and replicate channels to form 3-channel inputs. This resizing ensures consistent input dimensions across datasets. There are 10 classes (digits 0–9), with a balanced distribution across classes. MNIST is a benchmark for evaluating basic classification performance on small-scale image data and verifying the initial functionality of our models.

#### 3.1.2. CIFAR10

CIFAR-10 contains 60,000 color images (50,000 training, 10,000 test) at 32×32 pixels resolution, organized into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck [9]. Each class has exactly 6,000 images, providing a balanced dataset for evaluating model robustness on more complex, natural-image categories. We normalized each channel using means (0.4914, 0.4822, 0.4465) and stds (0.2023, 0.1994, 0.2010). These normalization values are derived from the ImageNet dataset and applied to center the data and improve training stability. A simple data augmentation was performed - random horizontal flipping (p=0.5).

### 3.2. Baseline Models

To provide a comparison against well-established architectures, we evaluated four baseline models (VGG-16, VGG-19 [12], ResNet-18, and ResNet-50 [6]) alongside our proposed models VGG with BottleNeck (VGG16-BN and VGG19-BN). **VGG** model family employs repeated 3×3 convolutional blocks with increasing depth (16/19 layers). While effective, their dense fully connected layers lead to high parameter counts (14.7M/20.0M) and FLOPs (629M/799M) (Table 1, 2), which limits their applicability in resource-constrained settings. The **ResNet** uses residual connections to mitigate vanishing gradients, enabling the training of deeper networks. ResNet-18 (11.2M params, 74M FLOPs) offers efficiency, while ResNet-50 (23.5M params, 169M FLOPs) (Table 1, 2) uses bottleneck blocks to reduce computational cost while maintaining high accuracy. These baseline models were chosen to represent a range of complexity and performance characteristics commonly found in image classification tasks.

| Metric | VGG-16 | VGG-19 | ResNet-18 | ResNet-50 | VGG16-BN | VGG19-BN |
|---|---|---|---|---|---|---|
| Params (M) | 14.7 | 20.0 | 11.2 | 23.5 | 6.1 | 1.4 |
| FLOPs (M) | 629 | 799 | 74 | 169 | 218 | 69 |
| Accuracy (%) | 99.4 | 99.5 | 99.3 | 99.5 | 99.6 | 99.5 |

Table 1. Models efficiency and performance comparison on MNIST. FLOPs calculated for 32×32 input.

### 3.3. Implementation Details

All models were implemented using PyTorch and trained on NVIDIA GeForce RTX 4070 (Driver Version 570.133.07; CUDA 12.8) with 12 GB of VRAM. We used the SGD with momentum (0.9) optimizer with a learning rate of $[1 \times 10^{-3}]$ and a weight decay of $[5 \times 10^{-4}]$. Learning rates were fixed to isolate architecture differences, though adaptive optimizers (e.g., Adam) may further improve VGG-BN. Models were trained for a maximum of 20 epochs. For the loss function the CrossEntropyLoss was used.

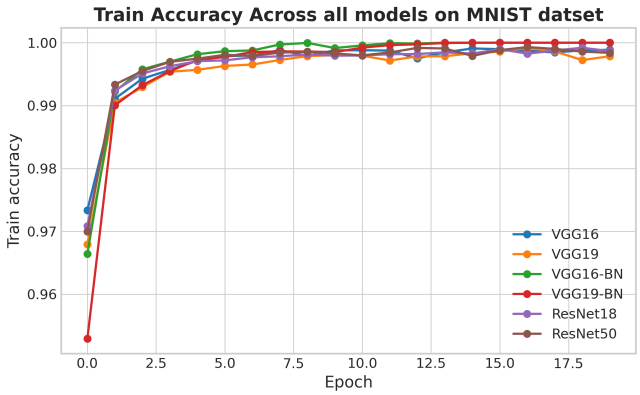
```
Trainable parameters: 6,103,578
Total parameters: 6,103,578
Percentage trainable: 100.00%
Epoch 1/20 - Train loss: 0.1155, Train acc: 0.9664, Test loss: 0.0416, Test acc: 0.9872
Epoch time: 0:00:16.584361 (Train: 0:00:15.081082, Eval: 0:00:01.503277
Epoch 2/20 - Train loss: 0.0256, Train acc: 0.9922, Test loss: 0.0319, Test acc: 0.9901
Epoch time: 0:00:16.557096 (Train: 0:00:15.130755, Eval: 0:00:01.426339
Epoch 3/20 - Train loss: 0.0149, Train acc: 0.9958, Test loss: 0.0279, Test acc: 0.9915
Epoch time: 0:00:16.715151 (Train: 0:00:15.225267, Eval: 0:00:01.489883
```

Figure 3. Spinet of training Log over Epochs for VGG-BN model on MNIST dataset

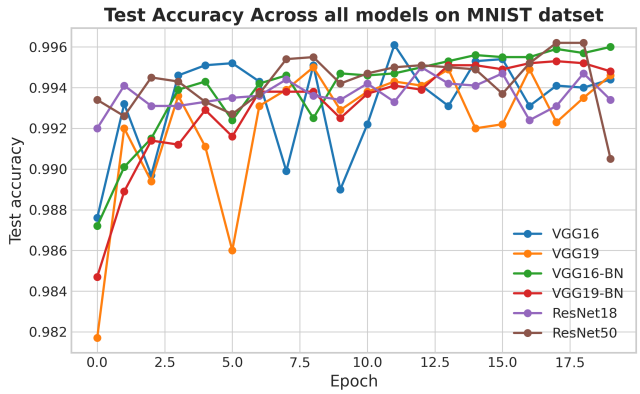
## 4. Results

### 4.1. MNIST Performance

All models achieved near-perfect performance on MNIST, with test accuracies exceeding 99% (Table 1),

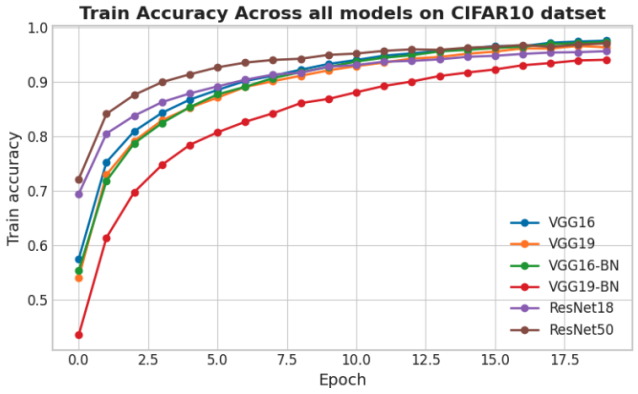



(a) Training accuracy.

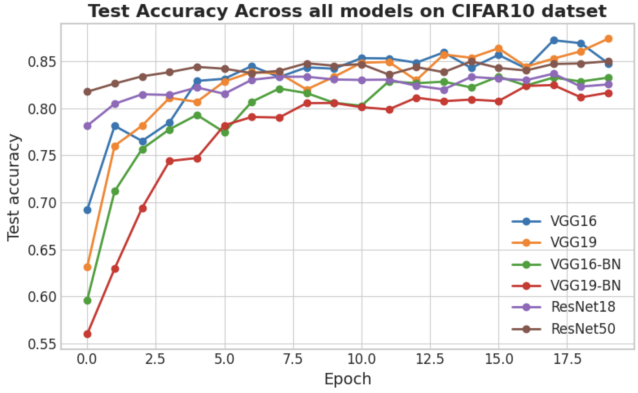



(b) Test accuracy.

Figure 4. MNIST accuracies over 20 epochs for six models: (a) training, (b) test.




(a) Training accuracy.




(b) Test accuracy.

Figure 5. CIFAR10 accuracies over 20 epochs for six models: (a) training, (b) test.

demonstrating the simplicity of MNIST for modern CNNs. Figure 4 shows both training and test accuracy curves. The proposed model, VGG-BN, reached 99% test accuracy at the same speed as standard VGG model (Figure 4b). However, VGG19-BN achieved 99.5% accuracy using 1.4M parameters, a 93% fewer parameters than VGG-19 (20M) (Table 1). The ResNet-50 showed minimal test fluctuation (±0.1%), due to its residual connections, while VGG-19 experienced more fluctuation (±0.5%)

The training loss curves in Figure 6a reveal that VGG-BN's model convergence follow the same patter as benchmark models.

### 4.2. CIFAR-10 Performance

CIFAR-10 presents a more challenging classification task, as reflected in the performance gap between training and test accuracy (Figure 5). All models reached between 95% and 100% training accuracy, but test accuracy plateaued between 81% and 90% (Figure 5b; Table 2). VGG-19 achieved the highest test accuracy (87.3%) but required most parameters (20M) and FLOPs (799M), which makes it computationally expensive. On the other hand, ResNet-18 balanced accuracy and efficiency, achieving 85.5% test accuracy with only 74M FLOPs—8.5× fewer than VGG-16 (629M FLOPs), which reached a similar accuracy of 84.9%. Our proposed models, VGG16-BN/VGG19-BN, delivered competitive performance (83.5 % and 81.8% accuracy, respectively) using significantly fewer parameters (6.1M/1.4M) and required less computational power (218m/69M FLOPs), offering a strong trade-off between efficiency and accuracy.

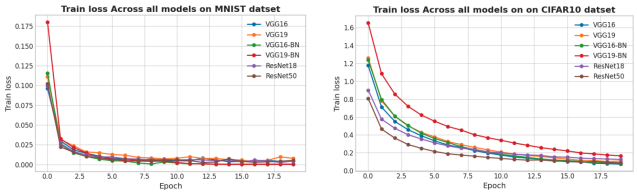
The figure 6b shows VGG-BN has the same loss descent pattern as the rest of the models, eventually converging.

### 4.3. Computational Efficiency

Comparisons of parameter counts and FLOPs (Tables 1 and 2) highlight several key trade-offs and the effi-

| Metric | VGG-16 | VGG-19 | ResNet-18 | ResNet-50 | VGG16-BN | VGG19-BN |
|---|---|---|---|---|---|---|
| Params (M) | 14.7 | 20.0 | 11.2 | 23.5 | 6.1 | 1.4 |
| FLOPs (M) | 629 | 799 | 74 | 169 | 218 | 69 |
| Accuracy (%) | 84.9 | 87.3 | 85.5 | 84.3 | 83.5 | 81.8 |

Table 2. Models efficiency and performance comparison on CIFAR-10. FLOPs calculated for 32×32 input.




(a) MNIST training loss (b) CIFAR-10 training loss

Figure 6. Training loss across all models for (a) MNIST and (b) CIFAR-10.

ciency of the proposed VGG-BN models. On MNIST, all models performed similarly despite large differences in parameter counts. On CIFAR-10, VGG19-BN achieved 81.8% accuracy with only 1.4M parameters—a 93% reduction compared to VGG-19. ResNet-18 also demonstrated strong efficiency, requiring 8.5× fewer FLOPs than VGG-16 to achieve similar accuracy. Our VGG16-BN matched ResNet-18's performance ( 85%) while using just 6.1M parameters—about 1.8× fewer than ResNet-18. **Summary:** The VGG-BN architectures substantially reduce the model size and computational cost, with only minor accuracy trade-offs on more complex datasets. This makes them attractive for deployment in resource-constrained environments.

## 5. Error Analysis

To better understand models limitations, we generated confusion matrices and per-class classification reports for each architecture on both MNIST and CIFAR-10 (see Appendix for details). In this paragraph, we summarize the most salient findings.

We focus on recall as the primary metric for error analysis: (1) our balanced datasets make precision/recall equally weighted in F1 less informative, (2) recall directly identifies classes where models fail to detect valid instances, and (3) for many vision applications, missing true positives (low recall) is more critical than false positives.

### 5.1. MNIST Failure Modes

Table 3 reports, for each architecture, the MNIST digit with the lowest recall and its value. This compact summary highlights the most challenging digit for each model. Even though overall test accuracy exceeds 98%, every architecture struggles with a small subset of confusing digits—most often "7" and "4".

| **Model** | **Worst Recall Digit** | **Recall (%)** |
|---|---|---|
| VGG-16 (plain) | 7 | 98.64 |
| VGG-16-BN | 2 | 99.23 |
| VGG-19 (plain) | 4 | 98.68 |
| VGG-19-BN | 4 | 99.19 |
| ResNet-18 | 7 | 99.12 |
| ResNet-50 | 7 | 98.83 |

Table 3. Per-model worst-case MNIST recall

Despite high overall accuracy, all networks share common failure modes. For example, VGG-16, ResNet-18, and ResNet-50 most often misclassify 7's, while both VGG-19 variants struggle with 4's, especially when the crossbar is faint or missing. Interestingly, VGG-16-BN's single lowest-recall digit is "2" (99.23%), where heavily blurred or smudged samples can resemble a "1".

Figure 8 (and additional examples in the Appendix) shows the first six misclassified images for VGG-16 vs. VGG-16-BN. These visualizations reveal that simple handwriting variations—partial occlusion, uneven strokes, and atypical slants—drive most errors. Bottlenecking (as in VGG-16-BN) subtly shifts these failure modes (e.g. "2" → "1" mistakes) but does not eliminate them.

By pinpointing which digits each model finds hardest, we gain actionable insights for targeted data augmentation (e.g. stroke-thickening, elastic distortions) and specialized regularization to boost generalization further.

### 5.2. CIFAR10 Failure Modes

Table 4 shows, for each model, the class with the lowest recall on CIFAR-10. This highlights which categories each network finds most challenging. CIFAR-10's low resolution and overlapping features between animal classes often lead to confusion, especially between cats and dogs.

| Model | Worst Recall Class | Recall (Min%) |
|---|---|---|
| ResNet-18 | cat | 63.5 |
| ResNet-50 | dog | 70.7 |
| VGG-16 (plain) | cat | 66.8 |
| VGG-16-BN | cat | 67.0 |
| VGG-19 (plain) | cat | 79.0 |
| VGG-19-BN | cat | 64.1 |

Table 4. Per-model worst-case recall on CIFAR-10. "Cat" dominates most failure modes, with one model worst on "dog."

Qualitative analysis (Figure 9) confirms these trends: models often confuse "cat" and "dog" images, likely due to similar fur textures, poses, or backgrounds. Other errors arise from background dominance (e.g., airplane tails misclassified as birds in blue sky) or low-resolution details. Notably, while VGG-16-BN slightly improves its worst-case recall on "cat" (67.0% vs. 66.8% in the plain VGG-16), VGG-19-BN's worst-case drop on "cat" to 64.1% (from 79.0% in VGG-19) indicates that aggressive bottlenecking can increase sensitivity to ambiguous features. These insights suggest targeted data augmentations—especially for "cat"—could further strengthen performance.

Even with strong overall accuracy, all models exhibit systematic weaknesses in ambiguous or visually similar classes. Error analysis by recalling and visualizing misclassified examples provides actionable insight for future model or data augmentation improvements.

Figure 7 visualizes the trade-off between computational cost (FLOPs) and test accuracy across all six models. We observe a clear Pareto front:

- VGG-19 (plain) achieves the highest accuracy (87.3%) but at the highest cost (799M FLOPs).
- ResNet-18 balances moderate compute (74M FLOPs) with strong accuracy (85.5%), placing it near the Pareto optimal curve.
- Bottleneck variants push further left: VGG16-BN (218M FLOPs, 83.5% accuracy) and VGG19-BN (69M FLOPs, 81.8% accuracy) demonstrate how aggressive parameter reduction trades off only gently in accuracy.

This analysis highlights ResNet-18 and VGG16-BN as attractive choices for resource-constrained deployment, offering strong accuracy at a moderate computational cost.

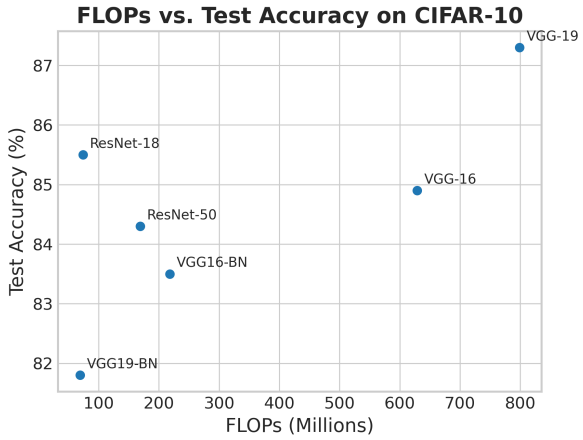



Figure 7. Trade-off between computational cost (FLOPs) and test accuracy on CIFAR-10 for six architectures.

## 6. Conclusion

In this paper, we introduced a VGG_BN, a bottleneck-augmented variant of the classic VGG architecture, designed to address the high parameter count and computational cost of standard VGG networks. Through experiments on MNIST and CIFAR-10—benchmarked against VGG-16/19 and ResNet-18/50—we showed that VGG_BN could: (1) dramatically reduce model size and compute (up to 93% fewer parameters and 3–4$\times$ fewer FLOPs); (2) maintain near-perfect accuracy on MNIST aand incurs only a modest 3–6% accuracy drop on the more challenging CIFAR-10 dataset; (3) reveal class-specific weaknesses, notably in "cat" vs. "dog" recall, pointing to targeted data-augmentation or class-balanced training as an actionable improvement. Although some degree of overfitting remains across all models, our analysis indicates that simple architectural changes—such as bottlenecking—can substantially improve the efficiency of VGG-style networks without sacrificing their hierarchical feature extraction strengths.

Future work will explore the integration of additional regularization techniques (e.g., dropout, mixup), domain-specific augmentations, transfer learning on larger datasets, and real-world deployment benchmarks to further validate the practicality of VG_BN for resource-constrained environments.

## References


[1] DigitalOcean Community. Deep learning architectures explained: Resnet, inceptionv3, squeezenet, 2025. Accessed: 2025-04-21. 1

[2] Xiangxiang Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Repvgg: Making vgg-style convnets great again. *arXiv preprint arXiv:2101.03697*, 2021. 2

[3] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5927–5935, 2017. 1, 2

[4] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 3

[7] K. Huyen. An overview of vgg16 and nin models. https://lekhuyen.medium.com/an-overview-of-vgg16-and-nin-models-96e4bf398484, 2023. Accessed: 2025-04-21. 2

[8] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014. 2

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. Technical Report. 3

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

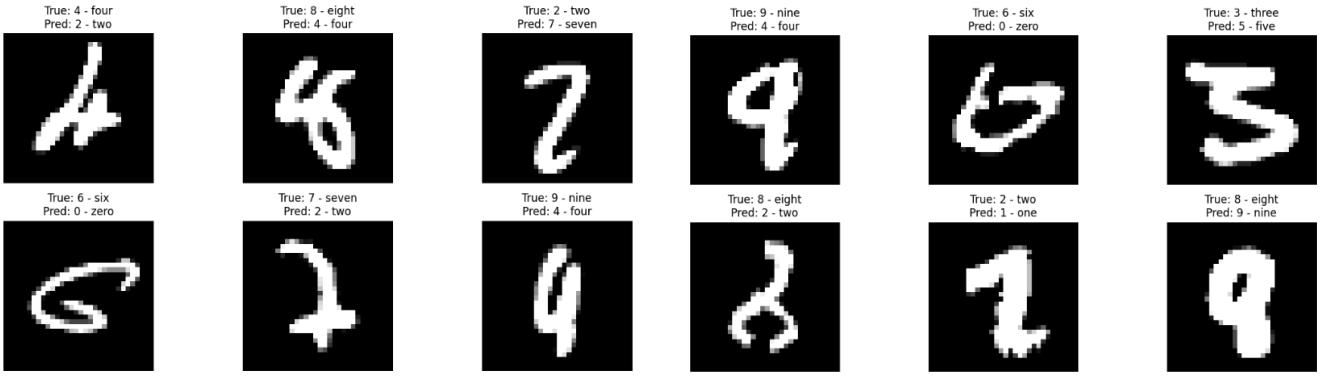
[11] Zhuang Li et al. Small-scale vgg-like convolutional neural networks for image classification. *arXiv preprint arXiv:1907.01395*, 2019. 2

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2015. 1, 3

[13] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500, 2017. 2

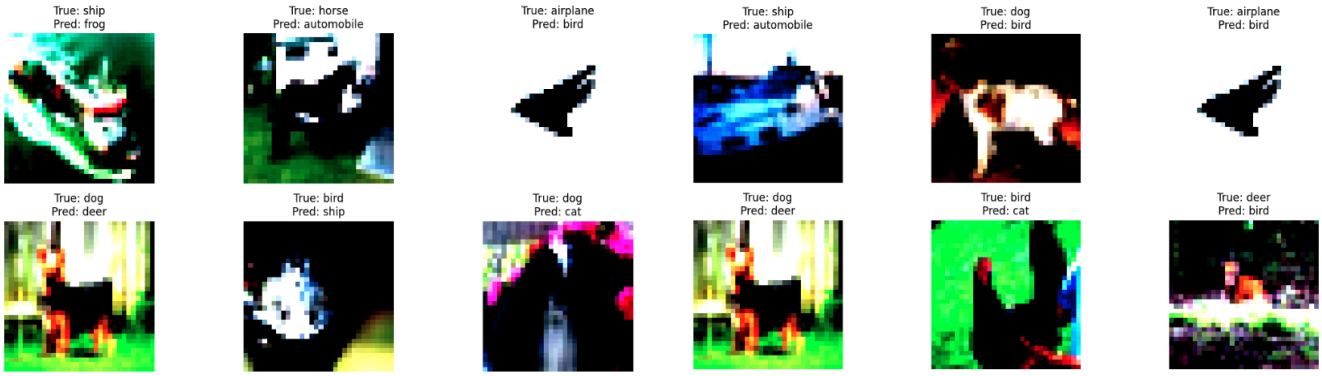
# Appendix

## Additional figures for error analysis




(a) MNIST missclassified images from VGG-16 model (b) MNIST missclassified images from our VGG_BN-16 model

Figure 8. MNIST missclassifications (a) VGG-16 model, (b) VGG_BN-16 model.




(a) CIFAR10 missclassified images from VGG-16 model (b) CIFAR10 missclassified images from our VGG_BN-16 model

Figure 9. CIFAR10 missclassifications (a) VGG-16 model, (b) VGG_BN-16 model.

## Code snippets for bottleneck implementation

```
class VGGBottleneckBlock(nn.Module):
    def __init__(self, in_channels, out_channels, bottleneck_ratio=4):
        super().__init__()
        mid_channels = out_channels // bottleneck_ratio
        self.reduce = nn.Conv2d(in_channels, mid_channels, kernel_size=1)
        self.bn1 = nn.BatchNorm2d(mid_channels)

        self.conv = nn.Conv2d(mid_channels, mid_channels, kernel_size=3, padding
            ↪ =1)
        self.bn2 = nn.BatchNorm2d(mid_channels)

        self.expand = nn.Conv2d(mid_channels, out_channels, kernel_size=1)
        self.bn3 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
```

```
        self.shortcut = nn.Sequential()
        if in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        identity = self.shortcut(x)
        out = self.relu(self.bn1(self.reduce(x)))
        out = self.relu(self.bn2(self.conv(out)))
        out = self.bn3(self.expand(out))
        out += identity
        out = self.relu(out)
        return out
```

**Full training logs with timestamps**

To illustrate the training progress, the Figure 10 shows the training log for VGG_BN-16 model on MNIST dataset.

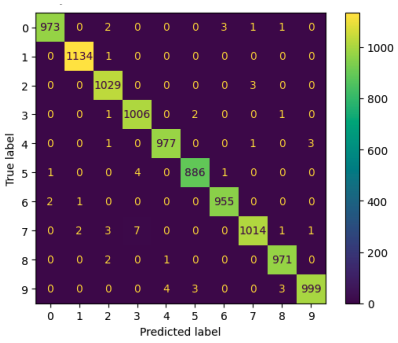
```
Trainable parameters: 6,103,578
Total parameters: 6,103,578
Percentage trainable: 100.00%
Epoch 1/20 - Train loss: 0.1155, Train acc: 0.9664, Test loss: 0.0416, Test acc: 0.9872
Epoch time: 0:00:16.584361 (Train: 0:00:15.081082, Eval: 0:00:01.503277
Epoch 2/20 - Train loss: 0.0256, Train acc: 0.9922, Test loss: 0.0319, Test acc: 0.9901
Epoch time: 0:00:16.557096 (Train: 0:00:15.130755, Eval: 0:00:01.426339
Epoch 3/20 - Train loss: 0.0149, Train acc: 0.9958, Test loss: 0.0279, Test acc: 0.9915
Epoch time: 0:00:16.715151 (Train: 0:00:15.225267, Eval: 0:00:01.489883
Epoch 4/20 - Train loss: 0.0102, Train acc: 0.9970, Test loss: 0.0200, Test acc: 0.9939
Epoch time: 0:00:16.488254 (Train: 0:00:14.986232, Eval: 0:00:01.502021
Epoch 5/20 - Train loss: 0.0068, Train acc: 0.9982, Test loss: 0.0199, Test acc: 0.9943
Epoch time: 0:00:16.437718 (Train: 0:00:14.935194, Eval: 0:00:01.502521
Epoch 6/20 - Train loss: 0.0048, Train acc: 0.9987, Test loss: 0.0263, Test acc: 0.9924
Epoch time: 0:00:16.416437 (Train: 0:00:14.928188, Eval: 0:00:01.488248
Epoch 7/20 - Train loss: 0.0045, Train acc: 0.9988, Test loss: 0.0199, Test acc: 0.9942
Epoch time: 0:00:16.469751 (Train: 0:00:14.971186, Eval: 0:00:01.498563
Epoch 8/20 - Train loss: 0.0021, Train acc: 0.9997, Test loss: 0.0167, Test acc: 0.9946
Epoch time: 0:00:16.718173 (Train: 0:00:15.111710, Eval: 0:00:01.606461
Epoch 9/20 - Train loss: 0.0008, Train acc: 1.0000, Test loss: 0.0291, Test acc: 0.9925
Epoch time: 0:00:16.584483 (Train: 0:00:15.126084, Eval: 0:00:01.458398
Epoch 10/20 - Train loss: 0.0033, Train acc: 0.9991, Test loss: 0.0176, Test acc: 0.9947
Epoch time: 0:00:16.350525 (Train: 0:00:14.872824, Eval: 0:00:01.477699
Epoch 11/20 - Train loss: 0.0020, Train acc: 0.9996, Test loss: 0.0189, Test acc: 0.9946
Epoch time: 0:00:16.197719 (Train: 0:00:14.725281, Eval: 0:00:01.472435
Epoch 12/20 - Train loss: 0.0010, Train acc: 0.9999, Test loss: 0.0182, Test acc: 0.9947
Epoch time: 0:00:16.354548 (Train: 0:00:14.857454, Eval: 0:00:01.497093
Epoch 13/20 - Train loss: 0.0011, Train acc: 0.9999, Test loss: 0.0171, Test acc: 0.9950
Epoch time: 0:00:16.064281 (Train: 0:00:14.643841, Eval: 0:00:01.420438
Epoch 14/20 - Train loss: 0.0005, Train acc: 1.0000, Test loss: 0.0164, Test acc: 0.9953
Epoch time: 0:00:16.928508 (Train: 0:00:15.428233, Eval: 0:00:01.500273
Epoch 15/20 - Train loss: 0.0005, Train acc: 1.0000, Test loss: 0.0161, Test acc: 0.9956
Epoch time: 0:00:16.670570 (Train: 0:00:15.122684, Eval: 0:00:01.547884
Epoch 16/20 - Train loss: 0.0005, Train acc: 1.0000, Test loss: 0.0160, Test acc: 0.9955
Epoch time: 0:00:16.629894 (Train: 0:00:15.042564, Eval: 0:00:01.587328
Epoch 17/20 - Train loss: 0.0005, Train acc: 1.0000, Test loss: 0.0159, Test acc: 0.9955
Epoch time: 0:00:16.729999 (Train: 0:00:15.372846, Eval: 0:00:01.357152
Epoch 18/20 - Train loss: 0.0006, Train acc: 1.0000, Test loss: 0.0156, Test acc: 0.9959
Epoch time: 0:00:16.916450 (Train: 0:00:15.456933, Eval: 0:00:01.459516
Epoch 19/20 - Train loss: 0.0006, Train acc: 1.0000, Test loss: 0.0161, Test acc: 0.9957
Epoch time: 0:00:16.686420 (Train: 0:00:15.139832, Eval: 0:00:01.546586
```

Figure 10. Training Log over Epochs for VGG-BN model on MNIST dataset

## Complete confusion matrices and classification reports for all models

### MNIST: Failure Examples, Confusion Matrices, and Classification Reports

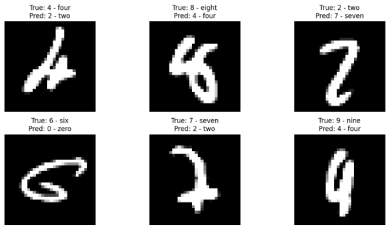
We conducted a detailed error analysis on the MNIST test set to understand where each architecture struggles. For each model, we (a) plot its confusion matrix to visualize common prediction errors, (b) show the per-class precision, recall, and F1 scores, and (c) display a handful of representative misclassified digits to highlight recurring failure modes such as faint strokes or unusual slants.




(a) VGG-16 confusion matrix

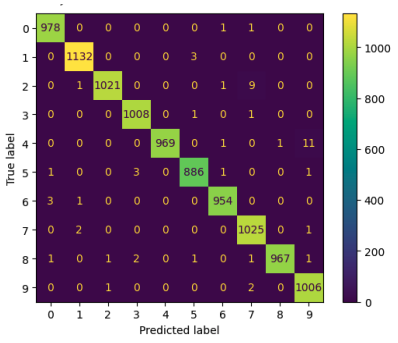
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 - zero | 1.00 | 0.99 | 0.99 | 980 |
| 1 - one | 1.00 | 1.00 | 1.00 | 1135 |
| 2 - two | 0.99 | 1.00 | 0.99 | 1032 |
| 3 - three | 0.99 | 1.00 | 0.99 | 1010 |
| 4 - four | 0.99 | 0.99 | 0.99 | 982 |
| 5 - five | 0.99 | 0.99 | 0.99 | 892 |
| 6 - six | 1.00 | 1.00 | 1.00 | 958 |
| 7 - seven | 1.00 | 0.99 | 0.99 | 1028 |
| 8 - eight | 0.99 | 1.00 | 1.00 | 974 |
| 9 - nine | 1.00 | 0.99 | 0.99 | 1009 |
| accuracy | | | 0.99 | 10000 |
| macro avg | 0.99 | 0.99 | 0.99 | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99 | 10000 |

(b) VGG-16 classification report



(c) VGG-16 misclassified examples

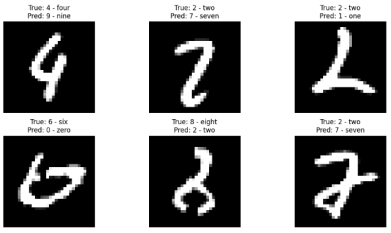
Figure 11. Error analysis for the plain VGG-16 model on MNIST: (a) confusion matrix, (b) detailed per-class metrics, (c) sample misclassifications illustrating typical error modes.




(a) VGG-19 confusion matrix

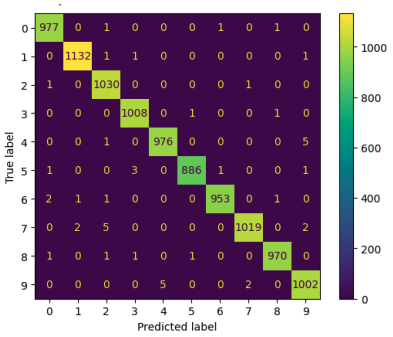
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 - zero | 0.99 | 1.00 | 1.00 | 980 |
| 1 - one | 1.00 | 1.00 | 1.00 | 1135 |
| 2 - two | 1.00 | 0.99 | 0.99 | 1032 |
| 3 - three | 1.00 | 1.00 | 1.00 | 1010 |
| 4 - four | 1.00 | 0.99 | 0.99 | 982 |
| 5 - five | 0.99 | 0.99 | 0.99 | 892 |
| 6 - six | 1.00 | 1.00 | 1.00 | 958 |
| 7 - seven | 0.99 | 1.00 | 0.99 | 1028 |
| 8 - eight | 1.00 | 0.99 | 1.00 | 974 |
| 9 - nine | 0.99 | 1.00 | 0.99 | 1009 |
| accuracy | | | 0.99 | 10000 |
| macro avg | 0.99 | 0.99 | 0.99 | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99 | 10000 |

(b) VGG-19 classification report



(c) VGG-19 misclassified examples

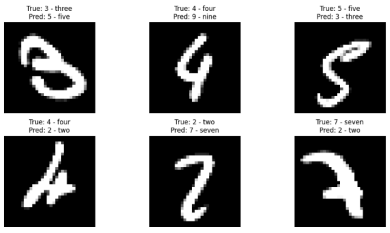
Figure 12. Error analysis for the plain VGG-19 model on MNIST: (a) confusion matrix, (b) detailed per-class metrics, (c) sample misclassifications illustrating typical error modes.




(a) ResNet-18 confusion matrix

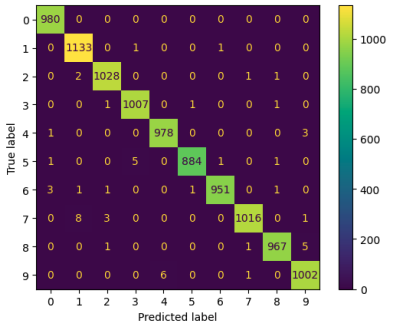
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 – zero | 0.99 | 1.00 | 1.00 | 980 |
| 1 – one | 1.00 | 1.00 | 1.00 | 1135 |
| 2 – two | 0.99 | 1.00 | 0.99 | 1032 |
| 3 – three | 1.00 | 1.00 | 1.00 | 1010 |
| 4 – four | 0.99 | 0.99 | 0.99 | 982 |
| 5 – five | 1.00 | 0.99 | 1.00 | 892 |
| 6 – six | 1.00 | 0.99 | 1.00 | 958 |
| 7 – seven | 1.00 | 0.99 | 0.99 | 1028 |
| 8 – eight | 1.00 | 1.00 | 1.00 | 974 |
| 9 – nine | 0.99 | 0.99 | 0.99 | 1009 |
| accuracy | | | 1.00 | 10000 |
| macro avg | 1.00 | 1.00 | 1.00 | 10000 |
| weighted avg | 1.00 | 1.00 | 1.00 | 10000 |

(b) ResNet-18 classification report



(c) ResNet-18 misclassified examples

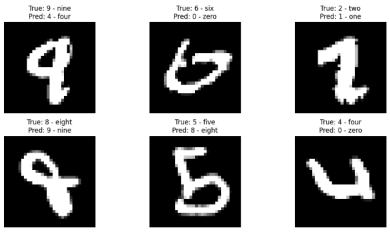
Figure 13. Error analysis for the ResNet-18 model on MNIST: (a) confusion matrix, (b) detailed per-class metrics, (c) sample misclassifications illustrating typical error modes.




(a) ResNet-50 confusion matrix

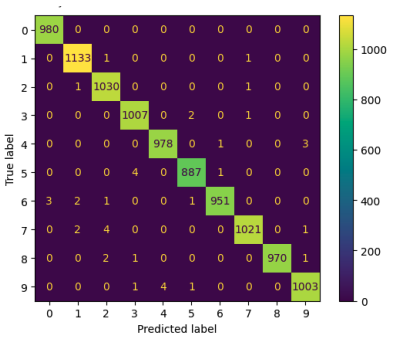
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 – zero | 0.99 | 1.00 | 1.00 | 980 |
| 1 – one | 0.99 | 1.00 | 0.99 | 1135 |
| 2 – two | 0.99 | 1.00 | 1.00 | 1032 |
| 3 – three | 0.99 | 1.00 | 1.00 | 1010 |
| 4 – four | 0.99 | 1.00 | 0.99 | 982 |
| 5 – five | 1.00 | 0.99 | 0.99 | 892 |
| 6 – six | 1.00 | 0.99 | 1.00 | 958 |
| 7 – seven | 1.00 | 0.99 | 0.99 | 1028 |
| 8 – eight | 1.00 | 0.99 | 0.99 | 974 |
| 9 – nine | 0.99 | 0.99 | 0.99 | 1009 |
| accuracy | | | 0.99 | 10000 |
| macro avg | 0.99 | 0.99 | 0.99 | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99 | 10000 |

(b) ResNet-50 classification report



(c) ResNet-50 misclassified examples

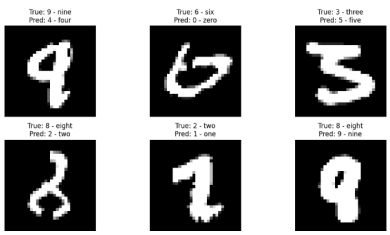
Figure 14. Error analysis for the ResNet-50 model on MNIST: (a) confusion matrix, (b) detailed per-class metrics, (c) sample misclassifications illustrating typical error modes.




(a) VGG-16-BN confusion matrix

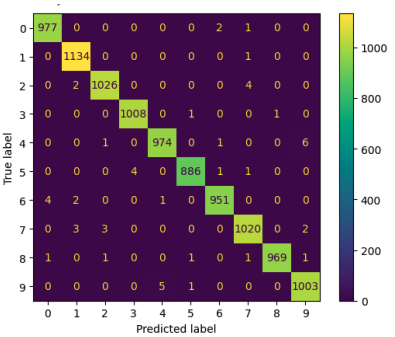
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 – zero | 1.00 | 1.00 | 1.00 | 980 |
| 1 – one | 1.00 | 1.00 | 1.00 | 1135 |
| 2 – two | 0.99 | 1.00 | 1.00 | 1032 |
| 3 – three | 0.99 | 1.00 | 1.00 | 1010 |
| 4 – four | 1.00 | 1.00 | 1.00 | 982 |
| 5 – five | 1.00 | 0.99 | 0.99 | 892 |
| 6 – six | 1.00 | 0.99 | 1.00 | 958 |
| 7 – seven | 1.00 | 0.99 | 1.00 | 1028 |
| 8 – eight | 1.00 | 1.00 | 1.00 | 974 |
| 9 – nine | 1.00 | 0.99 | 0.99 | 1009 |
| accuracy | | | 1.00 | 10000 |
| macro avg | 1.00 | 1.00 | 1.00 | 10000 |
| weighted avg | 1.00 | 1.00 | 1.00 | 10000 |

(b) VGG-16-BN classification report



(c) VGG-16-BN misclassified examples

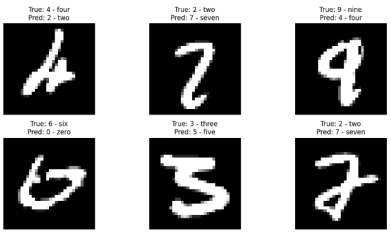
Figure 15. Error analysis for the VGG-16-BN model on MNIST: (a) confusion matrix, (b) detailed per-class metrics, (c) sample misclassifications illustrating typical error modes.




(a) VGG-19-BN confusion matrix

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 – zero | 0.99 | 1.00 | 1.00 | 980 |
| 1 – one | 0.99 | 1.00 | 1.00 | 1135 |
| 2 – two | 1.00 | 0.99 | 0.99 | 1032 |
| 3 – three | 1.00 | 1.00 | 1.00 | 1010 |
| 4 – four | 0.99 | 0.99 | 0.99 | 982 |
| 5 – five | 1.00 | 0.99 | 0.99 | 892 |
| 6 – six | 1.00 | 0.99 | 0.99 | 958 |
| 7 – seven | 0.99 | 0.99 | 0.99 | 1028 |
| 8 – eight | 1.00 | 0.99 | 1.00 | 974 |
| 9 – nine | 0.99 | 0.99 | 0.99 | 1009 |
| accuracy | | | 0.99 | 10000 |
| macro avg | 0.99 | 0.99 | 0.99 | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99 | 10000 |

(b) VGG-19-BN classification report

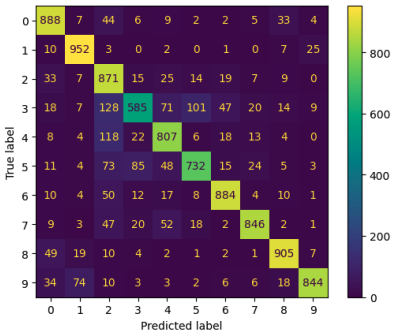


(c) VGG-19-BN misclassified examples

Figure 16. Error analysis for the VGG-19-BN model on MNIST: (a) confusion matrix, (b) detailed per-class metrics, (c) sample misclassifications illustrating typical error modes.

**CIFAR-10: Failure Examples, Confusion Matrices, and Classification Reports**

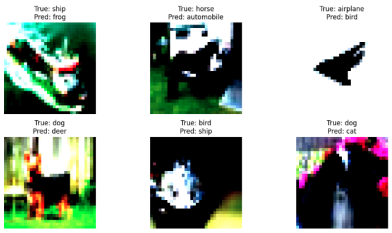
We performed an analogous error analysis on the CIFAR-10 test set to pinpoint each model's weaknesses. For each architecture, we show: (a) the confusion matrix to expose frequent misclassifications, (b) the per-class precision, recall, and F1-score breakdown, and (c) a few representative misclassified images that illustrate typical failure modes such as background dominance or class ambiguity.




(a) VGG-16 confusion matrix

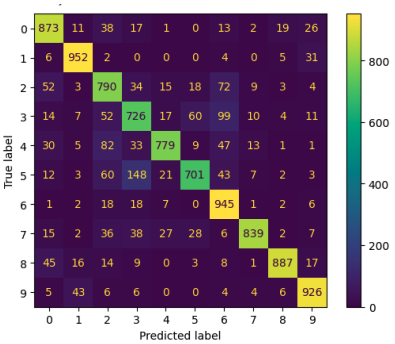
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.83 | 0.89 | 0.86 | 1000 |
| automobile | 0.88 | 0.95 | 0.91 | 1000 |
| bird | 0.64 | 0.87 | 0.74 | 1000 |
| cat | 0.78 | 0.58 | 0.67 | 1000 |
| deer | 0.78 | 0.81 | 0.79 | 1000 |
| dog | 0.83 | 0.73 | 0.78 | 1000 |
| frog | 0.89 | 0.88 | 0.89 | 1000 |
| horse | 0.91 | 0.85 | 0.88 | 1000 |
| ship | 0.90 | 0.91 | 0.90 | 1000 |
| truck | 0.94 | 0.84 | 0.89 | 1000 |
| accuracy | | | 0.83 | 10000 |
| macro avg | 0.84 | 0.83 | 0.83 | 10000 |
| weighted avg | 0.84 | 0.83 | 0.83 | 10000 |

(b) VGG-16 classification report




(c) VGG-16 misclassified examples

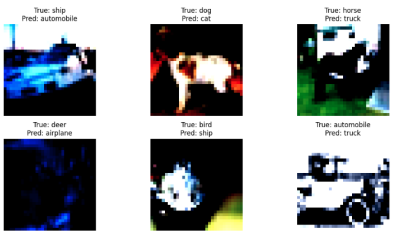
Figure 17. Error analysis for the plain VGG-16 model on CIFAR-10: (a) confusion matrix, (b) detailed per-class metrics, (c) representative misclassifications.




(a) VGG-19 confusion matrix

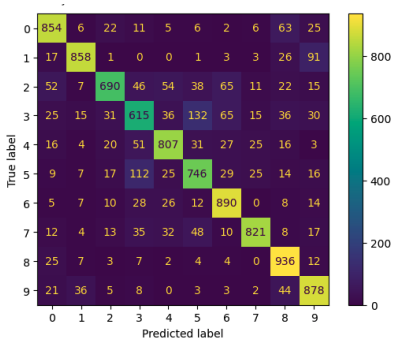
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.83 | 0.87 | 0.85 | 1000 |
| automobile | 0.91 | 0.95 | 0.93 | 1000 |
| bird | 0.72 | 0.79 | 0.75 | 1000 |
| cat | 0.71 | 0.73 | 0.72 | 1000 |
| deer | 0.90 | 0.78 | 0.83 | 1000 |
| dog | 0.86 | 0.70 | 0.77 | 1000 |
| frog | 0.76 | 0.94 | 0.84 | 1000 |
| horse | 0.95 | 0.84 | 0.89 | 1000 |
| ship | 0.95 | 0.89 | 0.92 | 1000 |
| truck | 0.90 | 0.93 | 0.91 | 1000 |
| accuracy | | | 0.84 | 10000 |
| macro avg | 0.85 | 0.84 | 0.84 | 10000 |
| weighted avg | 0.85 | 0.84 | 0.84 | 10000 |

(b) VGG-19 classification report




(c) VGG-19 misclassified examples

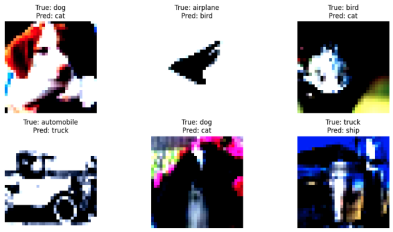
Figure 18. Error analysis for the plain VGG-19 model on CIFAR-10: (a) confusion matrix, (b) detailed per-class metrics, (c) representative misclassifications.




(a) ResNet-18 confusion matrix

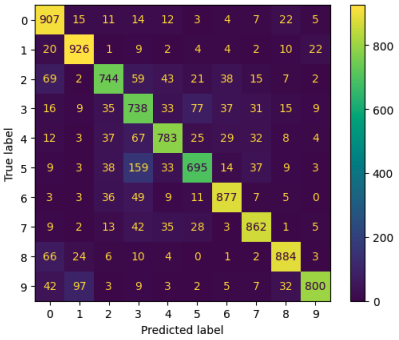
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.82 | 0.85 | 0.84 | 1000 |
| automobile | 0.90 | 0.86 | 0.88 | 1000 |
| bird | 0.85 | 0.69 | 0.76 | 1000 |
| cat | 0.67 | 0.61 | 0.64 | 1000 |
| deer | 0.82 | 0.81 | 0.81 | 1000 |
| dog | 0.73 | 0.75 | 0.74 | 1000 |
| frog | 0.81 | 0.89 | 0.85 | 1000 |
| horse | 0.90 | 0.82 | 0.86 | 1000 |
| ship | 0.80 | 0.94 | 0.86 | 1000 |
| truck | 0.80 | 0.88 | 0.84 | 1000 |
| accuracy | | | 0.81 | 10000 |
| macro avg | 0.81 | 0.81 | 0.81 | 10000 |
| weighted avg | 0.81 | 0.81 | 0.81 | 10000 |

(b) ResNet-18 classification report



(c) ResNet-18 misclassified examples

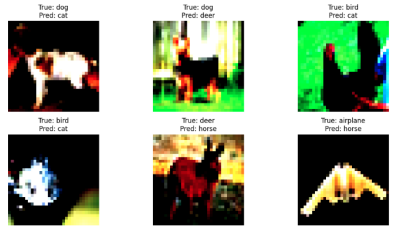
Figure 19. Error analysis for the ResNet-18 model on CIFAR-10: (a) confusion matrix, (b) detailed per-class metrics, (c) representative misclassifications.




(a) ResNet-50 confusion matrix

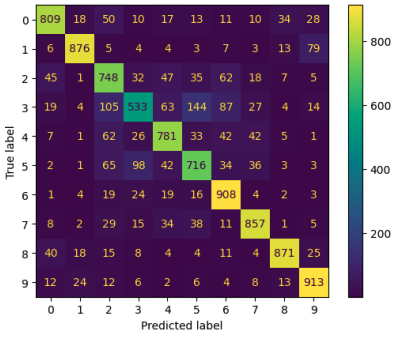
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.79 | 0.91 | 0.84 | 1000 |
| automobile | 0.85 | 0.93 | 0.89 | 1000 |
| bird | 0.81 | 0.74 | 0.77 | 1000 |
| cat | 0.64 | 0.74 | 0.68 | 1000 |
| deer | 0.82 | 0.78 | 0.80 | 1000 |
| dog | 0.80 | 0.69 | 0.74 | 1000 |
| frog | 0.87 | 0.88 | 0.87 | 1000 |
| horse | 0.86 | 0.86 | 0.86 | 1000 |
| ship | 0.89 | 0.88 | 0.89 | 1000 |
| truck | 0.94 | 0.80 | 0.86 | 1000 |
| accuracy | | | 0.82 | 10000 |
| macro avg | 0.83 | 0.82 | 0.82 | 10000 |
| weighted avg | 0.83 | 0.82 | 0.82 | 10000 |

(b) ResNet-50 classification report



(c) ResNet-50 misclassified examples

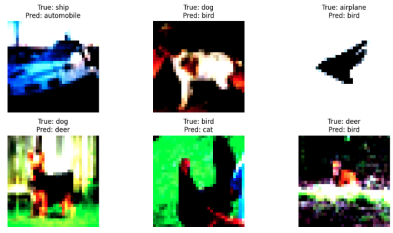
Figure 20. Error analysis for the ResNet-50 model on CIFAR-10: (a) confusion matrix, (b) detailed per-class metrics, (c) representative misclassifications.




(a) VGG-16-BN confusion matrix

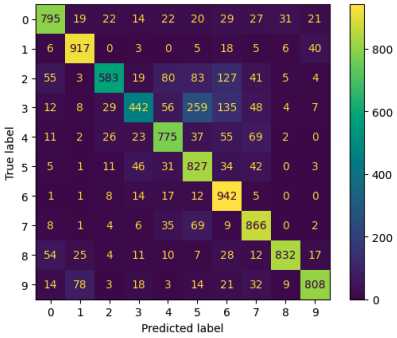
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.85 | 0.81 | 0.83 | 1000 |
| automobile | 0.92 | 0.88 | 0.90 | 1000 |
| bird | 0.67 | 0.75 | 0.71 | 1000 |
| cat | 0.71 | 0.53 | 0.61 | 1000 |
| deer | 0.77 | 0.78 | 0.78 | 1000 |
| dog | 0.71 | 0.72 | 0.71 | 1000 |
| frog | 0.77 | 0.91 | 0.83 | 1000 |
| horse | 0.85 | 0.86 | 0.85 | 1000 |
| ship | 0.91 | 0.87 | 0.89 | 1000 |
| truck | 0.85 | 0.91 | 0.88 | 1000 |
| accuracy | | | 0.80 | 10000 |
| macro avg | 0.80 | 0.80 | 0.80 | 10000 |
| weighted avg | 0.80 | 0.80 | 0.80 | 10000 |

(b) VGG-16-BN classification report



(c) VGG-16-BN misclassified examples

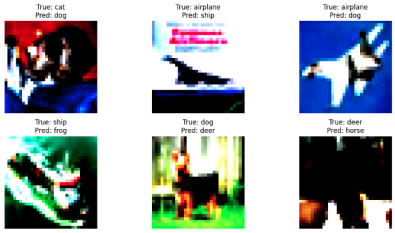
Figure 21. Error analysis for the VGG-16-BN model on CIFAR-10: (a) confusion matrix, (b) detailed per-class metrics, (c) representative misclassifications.




(a) VGG-19-BN confusion matrix

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| airplane | 0.83 | 0.80 | 0.81 | 1000 |
| automobile | 0.87 | 0.92 | 0.89 | 1000 |
| bird | 0.84 | 0.58 | 0.69 | 1000 |
| cat | 0.74 | 0.44 | 0.55 | 1000 |
| deer | 0.75 | 0.78 | 0.76 | 1000 |
| dog | 0.62 | 0.83 | 0.71 | 1000 |
| frog | 0.67 | 0.94 | 0.79 | 1000 |
| horse | 0.76 | 0.87 | 0.81 | 1000 |
| ship | 0.94 | 0.83 | 0.88 | 1000 |
| truck | 0.90 | 0.81 | 0.85 | 1000 |
| accuracy | | | 0.78 | 10000 |
| macro avg | 0.79 | 0.78 | 0.77 | 10000 |
| weighted avg | 0.79 | 0.78 | 0.77 | 10000 |

(b) VGG-19-BN classification report




(c) VGG-19-BN misclassified examples

Figure 22. Error analysis for the VGG-19-BN model on CIFAR-10: (a) confusion matrix, (b) detailed per-class metrics, (c) representative misclassifications.