Deeper Networks for Image Classification

230277302 Tianxiu Ma

1 Introduction

The purpose of image classification is to automatically categorize images into predefined classes. With the advancement of computer vision, the introduction of deep learning technologies has brought significant progress to image classification tasks. Before the rise of deep learning, image classification often relied on manually designed feature extraction methods such as SIFT (Scale-Invariant Feature Transform) and HOG (Histogram of Oriented Gradients). Later, with the introduction of deep learning, Convolutional Neural Networks (CNNs) became extremely popular due to their excellent ability to automatically extract and learn image features. Currently, the most commonly used models for image classification include ResNet, VGGNet, GoogleNet, and Transformer models.

2 Related Work

When choosing a classifier, several factors need to be considered, including the characteristics of the data, such as the dimensionality of the data, sample size, and noise level. Additionally, the requirements of the classification task should be considered, such as the number of categories, the difficulty of classification, etc. Finally, computational resources should also be taken into account, including the computing resources required for training and inference of the classifier.

2.1 Resnet

ResNet, by introducing residual connections, has solved the problem of vanishing gradients during the training of very deep networks, allowing for the construction of deeper architectures. This structure has significantly enhanced ResNet's performance in handling extremely complex visual recognition tasks, including image classification, object detection, semantic segmentation, and image generation. Nevertheless, deep ResNet models may experience overfitting when there is insufficient data, and training such models requires substantial computational resources. The concept of ResNet has also been extended to non-visual tasks such as natural language processing and audio processing, where residual connections help the network learn deep representations.

2.2 VGGNet

VGGNet is renowned for its simple and uniform structure, particularly its use of 3x3 convolutional kernels and 2x2 pooling layers across all convolutional layers. Despite its simplicity, VGGNet excels at extracting complex image features, making it highly effective on small-scale datasets. Due to its large number of parameters, VGGNet has substantial storage requirements. Additionally, pretrained VGG models are commonly used for transfer learning, where researchers often start with models pretrained on large datasets, such as ImageNet, and then fine-tune them for specific smaller datasets. In response to its high computational costs, various network compression and acceleration techniques have been developed, such as weight pruning, quantization, and knowledge distillation, to enable its operation on resource-constrained devices.

3 Model Description

VGGNet is widely used due to its simplicity and effectiveness, particularly on small-scale datasets, while ResNet excels at handling very deep network architectures. For this classification task, I plan to use both VGGNet and ResNet models.

3.1 ResNet Model Architecture

1. Residual Units: Each residual block contains two or three convolutional layers, typically interspersed with batch normalization and ReLU activation functions. Beyond these processing steps, each unit directly "jumps"

the original input information to the end of the unit. The advantage of this is that the network does not need to learn the complete output but rather the difference between the input and the output. These skip connections allow information to be directly transmitted to deeper layers, helping to solve the problem of vanishing gradients[1].

2. The ResNet network begins with a large processing step to preliminarily process the image, followed by a series of residual units, and ends with a global average pooling layer (a method of simplifying information) and a classification layer to determine the category of the image[1].

3. At the end of the ResNet network, global average pooling is used to reduce the complexity of the entire model and to minimize the risk of overfitting[1].

3.2 VGGNet Model Architecture

1. **Convolutional Layers**: The VGG network utilizes multiple convolutional layers, primarily employing relatively small 3×3 convolution kernels with a stride of 1. Padding between these layers is set to 1, ensuring that the spatial dimensions of the input image remain unchanged after each convolution.

2. **Pooling Layers**: Following several consecutive convolutional layers, there is usually a max pooling layer using 2×2 filters with a stride of 2. This helps to reduce the spatial dimensions of the data.

3. **Repetitive Pattern**: A distinctive feature of the VGG network is its repetitive structure, where a pattern of the same number of convolutional layers followed by a pooling layer is repeated multiple times. This design simplifies the learning process of the network and aids in capturing more complex features.

4. **Fully Connected Layers**: After several convolutional and pooling layers, the VGG network includes several fully connected layers with a large number of nodes (e.g., 4096).

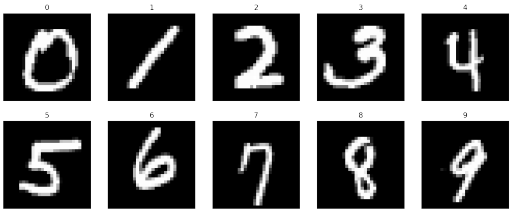
5. **Activation Function**: The VGG network employs the ReLU activation function after each convolutional layer to enhance the network's non-linear characteristics[4].

4 Experiments

4.1 Datasets

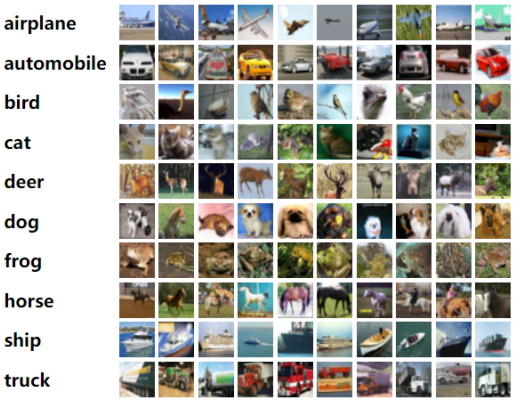
I plan to use the MNIST and CIFAR-10 datasets for classification tasks.

The MNIST dataset consists of 70,000 grayscale images, each depicting a handwritten digit from 0 to 9. Each image is standardized to 28x28 pixels with the digits centered within the image. Typically, the dataset is divided into two segments: 60,000 images for training and 10,000 for testing.



Although the MNIST dataset usually does not require complex models, there are situations where using advanced models might offer advantages. For example, even though pretrained VGGNet models are generally trained on color images, applying their weights at the beginning of training on MNIST could help accelerate model convergence or enhance performance in handling more complex tasks, such as recognizing digits in various styles. Additionally, using ResNet to process MNIST could also facilitate technical transfer, allowing insights and experience gained from MNIST to be applied to more complex related tasks. This approach not only helps improve efficiency for specific tasks but also expands our understanding and application of deep learning models in different scenarios[5].

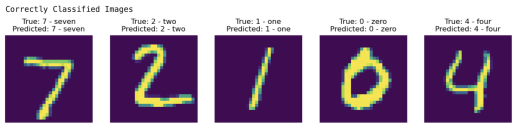
The CIFAR-10 dataset consists of 60,000 color images, each 32x32 pixels, evenly distributed across 10 categories with 6,000 images per category. The categories include: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is divided into a training set of 50,000 images and a test set of 10,000 images. Although the dataset includes images from a variety of natural categories, the variation within each category is relatively limited, allowing models to effectively classify by recognizing fewer patterns[6].



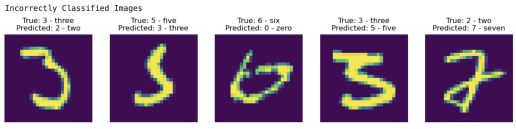
The CIFAR-10 dataset consists of color images, and the objects within these images vary significantly, necessitating more complex models to capture richer feature information. Both ResNet and VGGNet excel in handling this type of image data.

4.2 Testing Results

The following results show the digits that were correctly classified when using the ResNet model on the MNIST dataset.



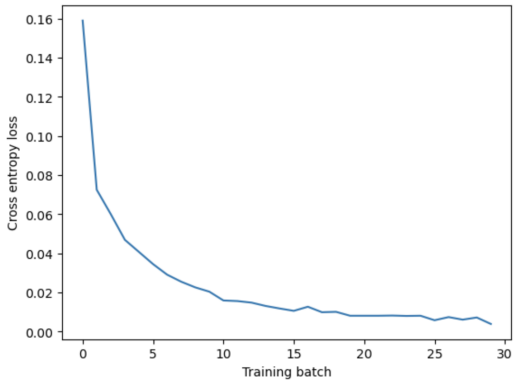
The test results indicate that when using the ResNet model on the MNIST dataset, the digits 2 and 7, as well as 5 and 3, are prone to confusion.



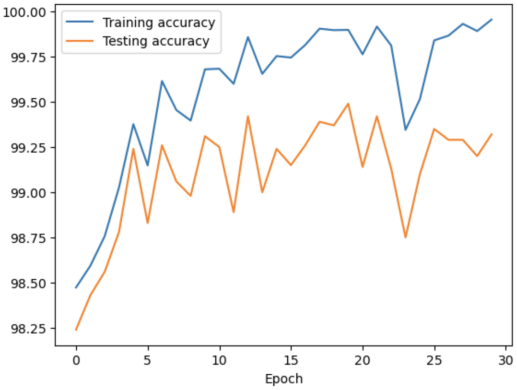
The image below demonstrates that in the CIFAR-10 dataset, images of different categories with similar features are prone to confusion, such as varying angles of the images which can lead to misclassification.

Below are the best results obtained by training ResNet on the MNIST dataset, along with the loss graph, and graphs for train accuracy and test accuracy.

The loss graph:

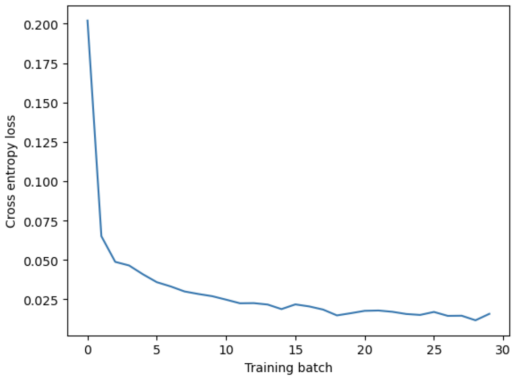


The graphs for train accuracy and test accuracy:

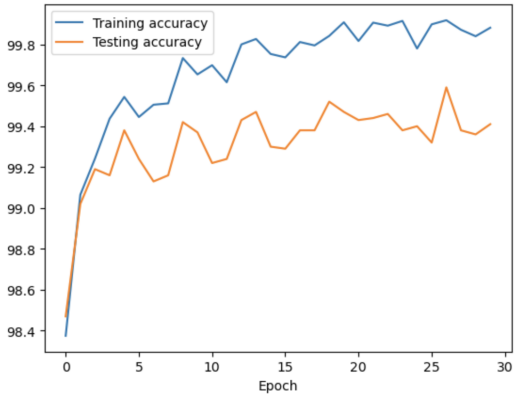


Below are the best results obtained by training VGGNet on the MNIST dataset, along with the loss graph, and graphs for train accuracy and test accuracy.

The loss graph:



The graphs for train accuracy and test accuracy:



4.3 Further Evaluation For ResNet

Initially, I trained the MNIST dataset using ResNet-18 and achieved a high test accuracy with simple parameter settings, as shown in Figure 1. Given that further optimization may not significantly improve the results, I plan to train using the CIFAR-10 dataset next.

At the beginning, I trained the CIFAR-10 dataset using the ResNet-18 model, setting the number of epochs to 20 and the learning rate to 0.001. I optimized the model using the SGD optimizer. With this setup, I achieved a classification test accuracy of 77.72%.

1. **change SGD to Adam**: I optimized the network parameters using the Stochastic Gradient Descent (SGD) algorithm. Subsequently, I switched to the Adam optimizer because Adam typically converges faster and is more robust to the choice of hyperparameters, especially the learning rate. As evident from Table 2, the test accuracy with SGD is lower than with Adam. This may be because the performance of SGD heavily relies on proper initial settings and precise adjustment of parameter scheduling. If these steps are not adequately managed, SGD's effectiveness might not match that of Adam.

2. **Experiment with learning rate and epochs**: I noticed that the model exhibited unstable accuracy, with abrupt fluctuations across consecutive epochs. To address this issue, I plan to experiment with adjusting the batch size and learning rate. Additionally, as shown in Table 2, the accuracy improves with an increase in epochs. This indicates that the model generally better generalizes to unseen data as the number of training epochs increases because it has more opportunities to learn complex patterns and features from the data, rather than merely memorizing the training set.

   Regarding the adjustment of the learning rate, the original Adam optimizer is set at a learning rate of 0.001. When I increased the learning rate to 0.01, while keeping other conditions the same over the same 20 epochs, the accuracy dropped from 81.39% to 79.62%. This decrease in accuracy could be due to the inherent noise in gradient estimates during mini-batch training, with a higher learning rate possibly amplifying the impact of this noise, leading to fluctuations in training outcomes and a reduction in accuracy.

3. **Change the model structure**: After upgrading from ResNet-18 to ResNet-34, I observed a noticeable increase in training time per epoch. Despite using the same parameter settings, the performance of ResNet-34 was found to be lower compared to ResNet-18. This phenomenon could be attributed to the increased depth of the network, which adds to the model's complexity. For a less complex dataset, such as CIFAR-10, a deeper network might learn noise in the data rather than the underlying signal. Therefore, if the dataset itself does not require such high model complexity, adding more layers may not necessarily lead to an improvement in performance.

4. **Add data augmentation**: When training the CIFAR-10 dataset using the ResNet18 model without data augmentation techniques, signs of overfitting were observed as indicated in Figure 2. To address this issue, we implemented data augmentation techniques, including random rotations, flips, and adjustments to brightness and contrast, to increase the diversity of the training data. Although these strategies effectively enhanced the model's generalization capabilities—reducing the discrepancy between train and test accuracy on both the MNIST and CIFAR-10 datasets as shown in Tables 3 and 2—they sometimes resulted in features being too abstract, complicating intuitive task interpretation[3].

   Specifically, when adjustments to brightness and contrast were applied to the CIFAR-10 data, they could distort crucial image features. Such excessive modifications may hinder the model's ability to recognize and learn effective features, thereby reducing classification accuracy.

5. **Add Squeeze-and-Excitation Block**: From the confusion matrix in Figures 3 and 4, it is evident that the categories "bird" and "airplane" are easily confused in the CIFAR-10 dataset. To address this issue, several strategies can be implemented to enhance the model's discriminative ability:

- Data Augmentation: By applying transformations such as rotation, scaling, translation, and horizontal flipping, the model's ability to interpret images from different angles and scales can be improved. This method helps the model learn and adapt to the subtle differences in similarly shaped objects.
- Attention Mechanism: Integrating SE (Squeeze-and-Excitation) blocks into the ResNet model enhances the model's focus on and learning of key features. This attention mechanism allows the network to concentrate more on essential information within the image, thereby increasing the accuracy of distinguishing similarly shaped categories.

Experimental results show that after adding SE blocks, the test accuracy reached 81.54%, an improvement from 81.39% without SE blocks (as shown in Table 2). Additionally, as seen from Figures 3 and 4, the number of times "bird" was mistakenly classified as "airplane" decreased from 113 to 66, and the classification accuracy of other categories also improved.

6. **Adjusting the parameters of the first convolutional layer in the ResNet18 model**: from the original `kernel_size=7`, `stride=2`, `padding=3` to `kernel_size=3`, `stride=1`, `padding=1`—significantly improved the test accuracy on the CIFAR-10 dataset (from 81.39% to 87.92%). This is because the image size in the CIFAR-10 dataset is 32x32, much smaller than the 224x224 images typically used with the ResNet architecture designed for ImageNet. Using a smaller `kernel_size` and `stride` better adapts to small-sized images. A smaller `kernel_size` retains more image details. A smaller `stride` (`stride=1`) means that the convolutional layer will have a higher spatial resolution when processing images, reducing potential information loss in the initial convolutional layer. When using a larger `stride`, such as the original `stride=2`, especially with small-sized images, important detail information may be lost early on.

7. **Different weight initialization methods serve specific purposes**: Xavier initialization is primarily designed for S-shaped activation functions such as Sigmoid and Tanh. It optimizes the training process by maintaining consistent variance in inputs and outputs, which helps prevent the issue of vanishing gradients. On the other hand, He initialization is specifically designed for ReLU and its variants, adjusting the variance of weights to accommodate the non-saturation characteristic of ReLU, which zeroes out negative inputs[2].

   However, in practical applications, particularly on the CIFAR-10 dataset, Xavier initialization has shown better performance than He initialization. This may be due to its ability to maintain a uniform distribution of activations and gradients, providing stability even when used with ReLU, in relatively shallow networks. This suggests that in cases where the network depth is not particularly significant, Xavier initialization may outperform He initialization, which is specifically designed for deeper networks and ReLU, due to its stability.

4.4 Further Evaluation For VGGNet

Just like with ResNet, I have also tried adjusting the batch size, learning rate, and implementing data augmentation techniques in the VGG model, finding the results to be similar to those with ResNet. Beyond the optimization strategies mentioned for ResNet, the following methods can also be considered for the VGG network targeting the MNIST and CIFAR-10 datasets:

1. **Finding the suitable VGG architecture**: initially using the VGG16 model, the input channels were set to 64, and the output channels to 512. This increase in the number of filters per convolutional block meant that the model would have a higher number of parameters, requiring more computational resources and GPU memory. With this configuration, both the training and test accuracies of the VGG16 model were at 10%, indicating that the model could not train effectively, possibly due to insufficient hardware resources or incorrect CUDA environment configuration. To adapt to a more resource-constrained environment, the input channels were later adjusted to 16 and the output channels to 128. Although this

adjustment might compromise some model performance, it significantly reduces the demand on computational resources. The architecture of the modified model is shown in Figure 5.

According to data from the table below, VGG16 achieves higher test accuracy on the CIFAR-10 dataset compared to VGG11 and VGG19. This may be because the complexity of VGG16 is well-suited to the scale of the CIFAR-10 dataset. In contrast, VGG11 might be too simple to effectively learn all the features in the data, while VGG19, being overly complex, could lead to model overfitting. Therefore, VGG16 demonstrates superior performance in this context.

num	model	epoch No.	dataset	batchsize	use data augmentation or not	Optimization Algorithm	learning rate	highest train accuracy	highest test accuracy
1	VGG11	30	cifar10	32	no	Adam	0.001	86.58%	72.23%
2	VGG11	30	cifar10	32	yes	Adam	0.001	70.97%	73.11%
3	VGG16	30	cifar10	32	no	Adam	0.001	85.26%	74.69%
4	VGG16	30	cifar10	32	yes	Adam	0.001	72.41%	73.63%
8	VGG19	30	cifar10	32	yes	Adam	0.001	58.66%	60.83%
9	**VGG16**	30	**cifar10**	128	no	Adam	0.001	96.27%	**79.07%**

When I trained VGG16 and VGG19 on the MNIST dataset, the test accuracy was only 11%. This might be because VGG16 and VGG19 were originally designed to handle larger images. To adapt them to the 28x28 pixel single-channel images of MNIST, I simplified the VGG architecture and created a VGG7 model. This simplified version adjusts the network layer parameters to better accommodate smaller input sizes. With these adjustments, VGG7 showed significantly improved performance on the MNIST dataset, demonstrating good performance.

num	model	epoch No.	dataset	batchsize	use data augmentation or not	Optimization Algorithm	learning rate	highest train accuracy	highest test accuracy
1	VGG11	30	mnist	32	no	Adam	0.001	99.79%.	99.36%
2	VGG11	30	mnist	32	yes	Adam	0.001	98.97%	99.18%
3	VGG16	30	mnist	32	no	Adam	0.001	11.24%	11.35%
4	VGG16	30	mnist	32	yes	Adam	0.001	11.24%	11.35%
5	VGG19	30	mnist	32	yes	Adam	0.001	11.24%	11.35%
6	VGG7	30	mnist	32	no	Adam	0.001	99.79%	99.43%
7	VGG9	30	mnist	32	no	Adam	0.001	99.70%	99.40%
8	VGG9	30	mnist	32	yes	Adam	0.001	99.09%	99.08%
9	VGG7	30	mnist	128	no	Adam	0.001	99.95%	99.52%
10	**VGG7**	30	mnist	128	no	Adam	0.001	99.92%	**99.59%**

2. **Adjust the size of the convolutional kernel**: Under the condition that other factors remain unchanged, adjusting the first convolutional layer of the VGG network from a kernel size of 3x3 to 5x5, and increasing the padding from 1 to 2, resulted in an increase in model accuracy from 99.52% to 99.59%. This change is likely because the larger convolutional kernel is better suited for capturing the fine details of the smaller images in the MNIST dataset, thus providing improved performance when processing these small-sized images.

5 Conclusion

By continuously adjusting parameters, optimizer methods, and model architectures, I trained ResNet and VGGNet on both the CIFAR-10 and MNIST datasets, achieving higher accuracies. This indicates that the probability of correctly classifying images increases with these adjustments. From this experience, I learned that selecting the appropriate model architecture is crucial for different datasets and tasks. The impact of adjusting parameters such as learning rate, optimizer methods, and model architecture on model performance is significant. By carefully tuning these parameters, the optimal combination can be found to improve the model's accuracy. Understanding the characteristics of the dataset is vital for designing effective models. CIFAR-10 and MNIST have distinct features, requiring tailored adjustments to the model to adapt to these characteristics and achieve better classification performance. Training a model is not a one-time task but rather a continuous optimization process. By observing the model's performance during training and adjusting parameters and model structure accordingly, the model's performance can be continuously improved to excel in various tasks.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "*Deep Residual Learning for Image Recognition.*" 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "*Deep Residual Learning for Image Recognition: Supplementary Material.*" 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "*Bag of Tricks for Image Classification with Convolutional Neural Networks.*" 2019.

[4] K. Simonyan and A. Zisserman, "*Very deep convolutional networks for large-scale image recognition.*" 2014.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "*Gradient-based learning applied to document recognition.*" 1998.

[6] A. Krizhevsky,"*Learning Multiple Layers of Features from Tiny Images.*" 2009.

APPENDIX

A RESNET EXPERIMENT DETAILS

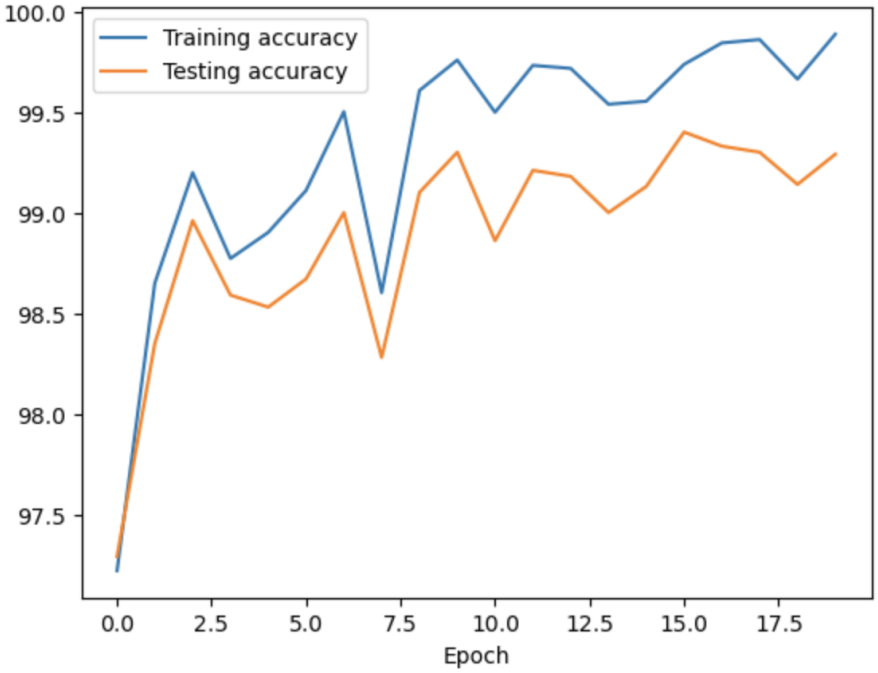


Figure 1: resnet18 on mnist dataset with 20 epochs

num	model	epoch No.	dataset	batchsize	data augmentation techniques	Optimization Algorithm	learning rate	highest train accuracy	highest test accuracy
1	resnet18	20	cifar10	32	Random horizontal flip & horizontal flip & Random rotation	Adam	0.001	81.19%	81.39%
2	resnet18	20	cifar10	32	Random horizontal flip & Random rotation & Adjust brightness and contrast	Adam	0.001	89.95%	80.71%

Table 1: resnet18 data augmentation changes

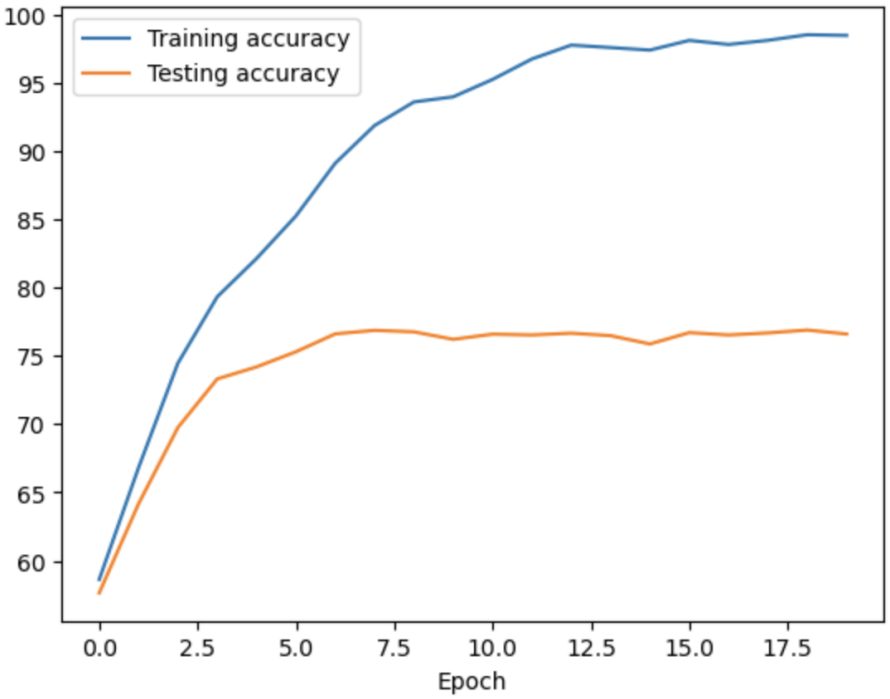


Figure 2: resnet18 on mnist dataset with no data augmentation

Figure 3: confusion matrix of resnet18 on mnist dataset with SE

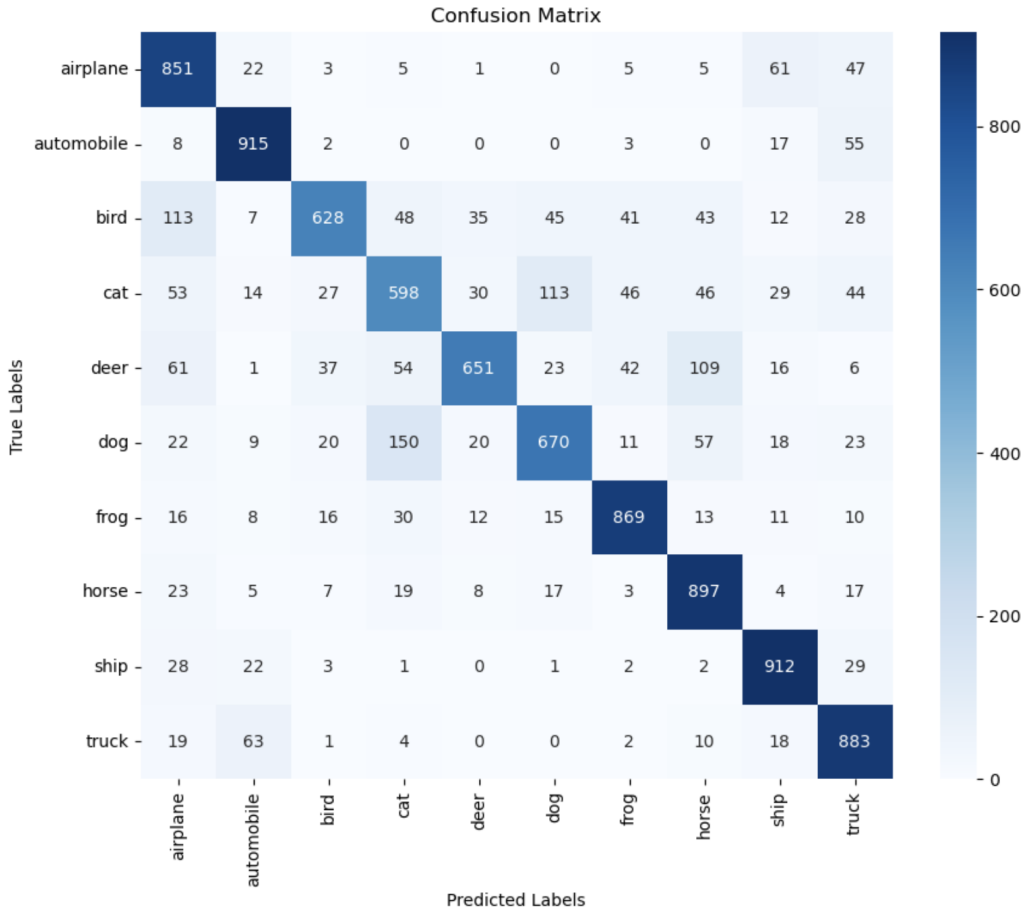


Figure 4: confusion matrix of resnet18 on mnist dataset

num	model	epoch No.	dataset	batchsize	use data augmentation or not	Optimization Algorithm	learning rate	highest train accuracy	highest test accuracy
1	resnet18	20	cifar10	32	no	Adam	0.001	98.54%	76.90%
2	resnet18	20	cifar10	32	yes	SGD	0.001	87.86%	77.72%
3	resnet18	20	cifar10	32	yes	Adam	0.001	81.19%	81.39%
4	resnet18	30	cifar10	32	yes	Adam	0.001	85.22%	83.37%
5	resnet18	50	cifar10	32	yes	Adam	0.001	**88.50%**	**84.30%**
6	resnet34	20	cifar10	32	yes	Adam	0.001	81.41%	80.79%
7	resnet34	30	cifar10	32	yes	Adam	0.001	84.37%	82.81%
8	resnet18	20	cifar10	128	yes	Adam	0.001	81.49%	80.97%
9	resnet18	20	cifar10	32	yes	Adam	0.01	83.40%	79.62%

Table 2: Under the condition that other parameters of the model remain unchanged, the accuracy changes as described above with incremental increases in the number of epochs. Additionally, under the same conditions, adjusting the learning rate of the ResNet-18 model from 0.001 to 0.01 resulted in a decrease in test accuracy. When switching the optimizer from SGD to Adam, the test accuracy improved.

num	model	epoch No.	dataset	batchsize	use data augmentation or not	Optimization Algorithm	learning rate	highest train accuracy	highest test accuracy
1	resnet18	20	mnist	32	no	Adam	0.001	99.89%	99.40%
2	resnet18	30	mnist	32	no	Adam	0.001	99.95%	99.49%
3	resnet18	50	mnist	32	no	Adam	0.001	99.98%	99.44%
4	resnet18	20	mnist	32	yes	Adam	0.001	99.40%	99.21%
5	resnet18	30	mnist	32	yes	Adam	0.001	99.53%	99.31%

Table 3: When applying the ResNet18 model on the MNIST dataset, adjust the model training by changing the number of training epochs and deciding whether to use data augmentation techniques.

B VGGNET EXPERIMENT DETAILS

```
VGGNet(
  (vgg_modules): ModuleList(
    (0): VGGBlock(
      (block): ModuleList(
        (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): ReLU()
        (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (1): VGGBlock(
      (block): ModuleList(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): ReLU()
        (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (2): VGGBlock(
      (block): ModuleList(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): ReLU()
        (4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (5): ReLU()
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (3): VGGBlock(
      (block): ModuleList(
        (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): ReLU()
        (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (5): ReLU()
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (4): VGGBlock(
      (block): ModuleList(
        (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU()
        (2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): ReLU()
        (4): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (5): ReLU()
        (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      )
    )
    (5): Flatten(start_dim=1, end_dim=-1)
    (6): Linear(in_features=512, out_features=4096, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.5, inplace=False)
    (9): Linear(in_features=4096, out_features=4096, bias=True)
    (10): ReLU()
    (11): Dropout(p=0.5, inplace=False)
    (12): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

Figure 5: VGGNet16 architecture