DATA130051.01 Computer Vision Final Project

吴嘉骜 21307130203

June 24, 2024

Abstract

This report presents the completion of three tasks for the *Computer Vision* Final Project. Task one involves comparing the performance of self-supervised learning and supervised learning on image classification tasks. We utilize SimCLR with ResNet-18 as the base network, trained on STL-10, and evaluate it by Linear Classification Protocol on CIFAR-100. Task two concentrates on comparing the performance of image classification models based on CNN and Transformer architectures. We use EfficientNet and ResNet-50 as CNN models, MobileViT and Swin-T as Transformer models, trained on CIFAR-100 with the same policy, and adopt data augmentation techniques like CutMix to improve performance. Task three embraces the object reconstruction and novel view synthesis based on NeRF. We train the NeRF model based on our own photographs with NeRF-PyTorch and obtain a short video of the synthesized novel view. The report details the training specifics, performance metrics, and visualization results for all tasks.

Experiment environment

GPU: V100-32GB * 4

CPU: 6 vCPU Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz

PyTorch 2.3.0 Python 3.12(ubuntu22.04) Cuda 12.1

More details can be found in the project repository.

Model links

Source code: Github link

Model parameters and videos: Baidu Netdisk link

1 Task1: Self-supervised and Supervised Learning on Image Classification

1.1 Introduction

Self-supervised learning has recently gained significant attention as an alternative to traditional supervised learning methods in computer vision. While supervised learning relies on labeled data for training, self-supervised learning leverages unlabeled data to learn meaningful representations. This task explores

and compares the performance of supervised learning and self-supervised learning approaches image classification.

In this task, we use the **SimCLR** framework for self-supervised learning. We train a ResNet-18 model on the **STL-10** dataset using SimCLR and evaluate its performance on the **CIFAR-100** dataset using the **Linear Classification Protocol (LCP)**. Then we compare the results with a pre-trained ResNet-18 model and a ResNet-18 model trained from scratch on CIFAR-100.

1.2 SimCLR

SimCLR (Simple Framework for Contrastive Learning of Visual Representations) is a state-of-the-art self-supervised learning technique introduced by Chen et al.[1].

A key component of SimCLR is the contrastive loss function, which encourages the model to assign similar representations to augmented views of the same image and dissimilar representations to views of different images. By training a model to maximize this agreement, SimCLR learns powerful representations that can be fine-tuned for downstream tasks like image classification. A sketch of the SimCLR framework is shown in Figure 1.

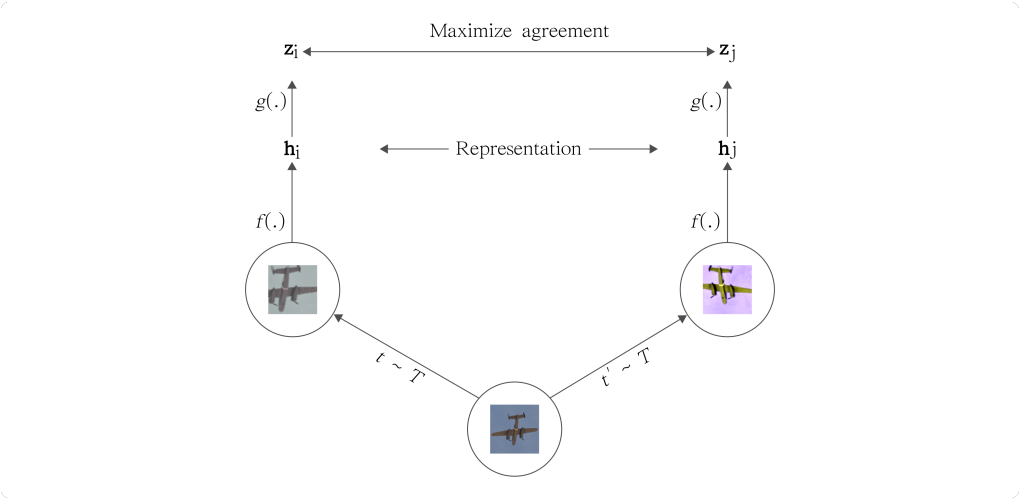


Figure 1: SimCLR Framework

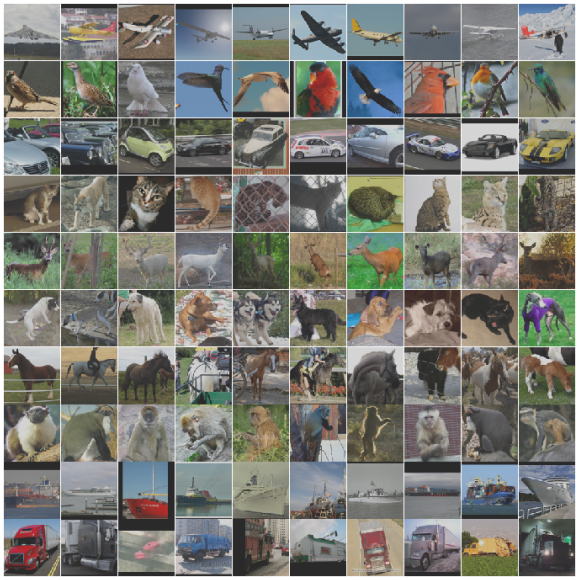
1.3 Datasets

We use two datasets: STL-10 and CIFAR-100 in this task, which are commonly used for evaluating self-supervised learning and supervised learning models, respectively. Some samples from the datasets are shown in Figure 2.

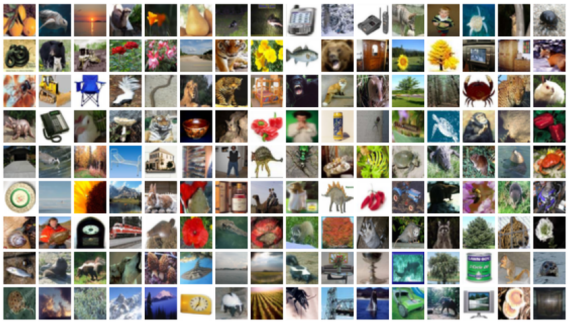
The STL-10[11] is an image dataset derived from ImageNet and popularly used to evaluate algorithms of unsupervised feature learning or self-taught learning. Besides 100,000 unlabeled images, it contains 13,000 labeled images from 10 object classes, among which 5,000 images are partitioned for training while the remaining 8,000 images for testing. All the images are color images with 96×96 pixels in size.

The CIFAR-100[12] (Canadian Institute for Advanced Research, 100 classes) consists of 60000 32x32 color images. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. There are 600 images

per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). There are 500 training images and 100 testing images per class.



(a) STL-10 Dataset



(b) CIFAR-100 Dataset

Figure 2: Dataset Samples

1.4 Model Settings

We have borrowed much of our code implementation from [13], but have made additional modifications to meet the requirements of the task.

The settings can be divided into two parts: the self-supervised learning part and linear classification part.

```
# Self-supervised Learning Settings
model_name = "ResNet18"
batch_size = 1024  # 256 per GPU
epochs = 125
learning_rate = 0.0003
weight_decay = 1e-4
optimizer = "Adam"
scheduler = CosineAnnealingLR(T_max=len(train_loader), eta_min=1e-6)
temperature = 0.07
out_dim = 128  # feature dimension
n_views = 2  # number of views for contrastive learning
```

We choose NCE (Noise Contrastive Estimation) loss as the contrastive loss function. This loss function is defined as follows:

$$\mathcal{L}_{NCE}(z_i, z_j) = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)}$$

where z_i and z_j are the representations of two augmented views of the same image, $\text{sim}(z_i, z_j) = z_i^T z_j / \|z_i\| \|z_j\|$ is the cosine similarity, and τ is the temperature parameter.

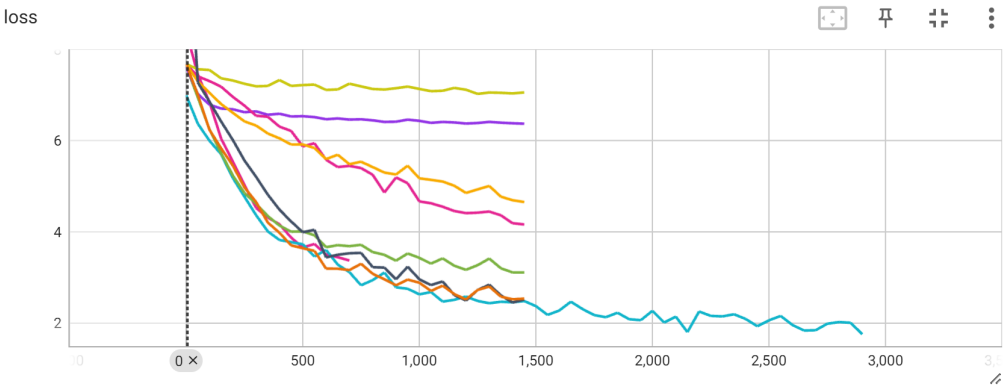
```
# Linear Classification Settings (For LCP; Pre-training and Scratch)
model_name = "ResNet18"
batch_size = 128
epochs = 40
learning_rate = 0.0005
weight_decay = 8e-4
optimizer = "Adam"
criterion = nn.CrossEntropyLoss()
scheduler = None
transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565,
            0.2761))
    ])
transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565,
            0.2761))
    ])
```

We utilize Logistic Regression as the protocol. This protocol involves training a linear classifier on top of the frozen features extracted from the SSL pre-trained model. The classifier is trained using the labeled data from CIFAR-100 and evaluated on the test set. The performance is measured in terms of top-1 and top-5 accuracy. Top-1 accuracy is the percentage of correct predictions among the total predictions, while top-5 accuracy considers the prediction to be correct if the true label is within the top 5 predicted labels.

1.5 Hyperparameter Tuning

We have conducted a grid search to tune the hyperparameters for both the self-supervised learning and linear classification parts. Here we only show the former part, and a TensorBoard visualization of the tuning results is shown in Figure 3. We have tried different batch sizes, learning rates, and temperature values for the contrastive loss function.

We can observe that the model is sensitive to the learning rate and temperature values. We then choose the best-performing hyperparameters based on the loss curves.



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
b1024-lr0.0003-t0.07-e15	3.3754	8.3862	8.3862	3.3754	↓5.0108	↓-60%	0	700
b256-lr0.0003-t0.07-e15	1.761	6.9912	6.9912	1.761	↓5.2302	↓-75%	0	2,900
b256-lr0.0003-t0.1-e15	3.1138	7.6241	7.6241	3.1138	↓4.5103	↓-59%	0	1,450
b256-lr0.0003-t0.5-e15	6.3692	7.6098	7.6098	6.3692	↓1.2406	↓-16%	0	1,450
b256-lr0.003-t0.07-e15	4.1639	7.6742	7.6742	4.1639	↓3.5104	↓-46%	0	1,450
b256-lr0.03-t0.07-e15	7.0214	7.6742	7.6742	7.0538	↓0.6204	↓-8%	0	1,450
b512-lr0.00003-t0.07-e15	4.6549	7.6742	7.6742	4.6549	↓3.0193	↓-39%	0	1,450
b512-lr0.0003-t0.01-e15	2.4541	11.4537	11.4537	2.5236	↓8.9301	↓-78%	0	1,450
b512-lr0.0003-t0.07-e15	2.5219	7.6727	7.6727	2.541	↓5.1316	↓-67%	0	1,450

Figure 3: SimCLR Hyperparameter Tuning

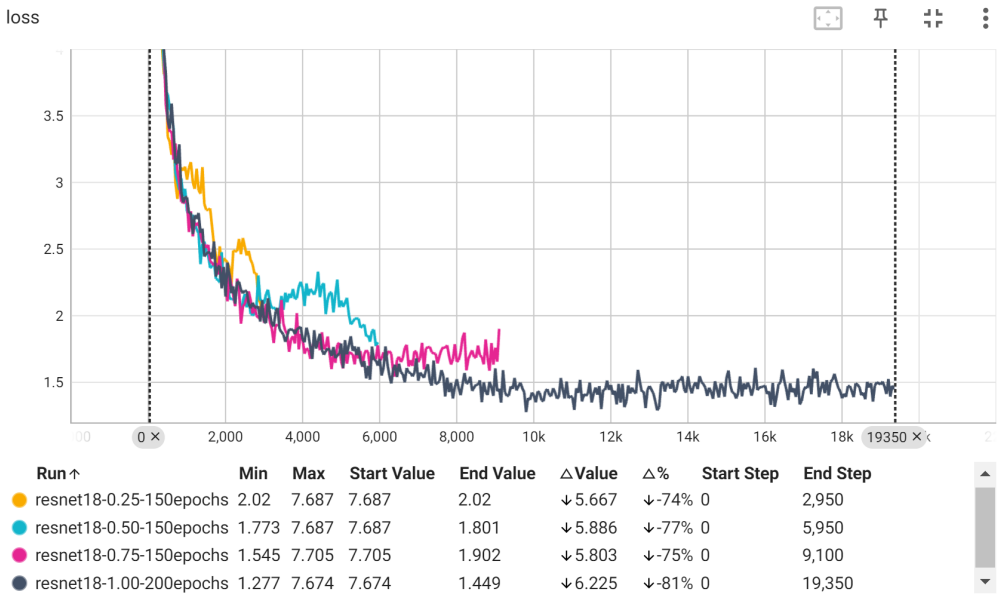
1.6 Experiment Results

We first show the training loss curves for the SimCLR model on STL-10 in Figure 4. Note that we train the model with full dataset for 200 epochs, and 150 epochs respectively for the subset ratio of 0.75, 0.5, and 0.25. We can observe that the model converges well during training, with the loss decreasing steadily over epochs.

Additionally, we evaluate the performance of the SimCLR models on CIFAR-100 using the LCP for 40 epochs. We also compare the results with a pre-trained ResNet-18 model on ImageNet and a ResNet-18 model trained from scratch on CIFAR-100 with the same training(finetuning) policy. The top-1 and top-5 accuracies on test set are shown in Table 1. The accuracy curves are shown in Figure 5 and 6.

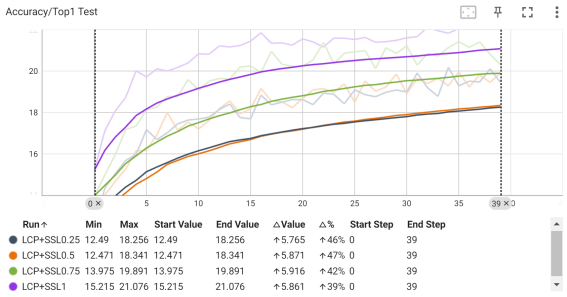
Table 1: Training 40 epochs on CIFAR-100 with LCP

Models	**Top-1 Accuracy**	**Top-5 Accuracy**
Full Data SSL	22.32	47.36
0.75 Data SSL	20.31	44.63
0.5 Data SSL	19.91	44.10
0.25 Data SSL	19.45	43.52
Pre-trained on ImageNet	57.83	84.63
Trained from Scratch	53.99	81.28



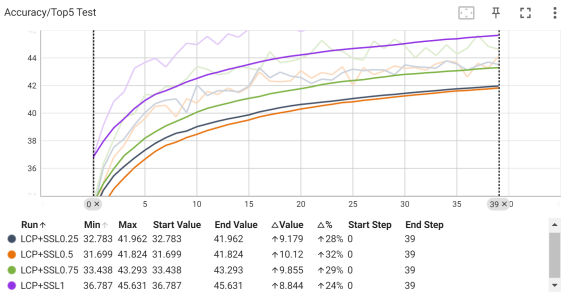
Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
resnet18-0.25-150epochs	2.02	7.687	7.687	2.02	↓5.667	↓-74%	0	2,950
resnet18-0.50-150epochs	1.773	7.687	7.687	1.801	↓5.886	↓-77%	0	5,950
resnet18-0.75-150epochs	1.545	7.705	7.705	1.902	↓5.803	↓-75%	0	9,100
resnet18-1.00-200epochs	1.277	7.674	7.674	1.449	↓6.225	↓-81%	0	19,350

Figure 4: SimCLR Training Loss on STL-10



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
LCP+SSL0.25	12.49	18.256	12.49	18.256	↑5.765	↑46%	0	39
LCP+SSL0.5	12.471	18.341	12.471	18.341	↑5.871	↑47%	0	39
LCP+SSL0.75	13.975	19.891	13.975	19.891	↑5.916	↑42%	0	39
LCP+SSL1	15.215	21.076	15.215	21.076	↑5.861	↑39%	0	39

(a) Top-1 Accuracy



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
LCP+SSL0.25	32.783	41.962	32.783	41.962	↑9.179	↑28%	0	39
LCP+SSL0.5	31.699	41.824	31.699	41.824	↑10.12	↑32%	0	39
LCP+SSL0.75	33.438	43.293	33.438	43.293	↑9.855	↑29%	0	39
LCP+SSL1	36.787	45.631	36.787	45.631	↑8.844	↑24%	0	39

(b) Top-5 Accuracy

Figure 5: SimCLR Performance on CIFAR-100 (Values smoothed)

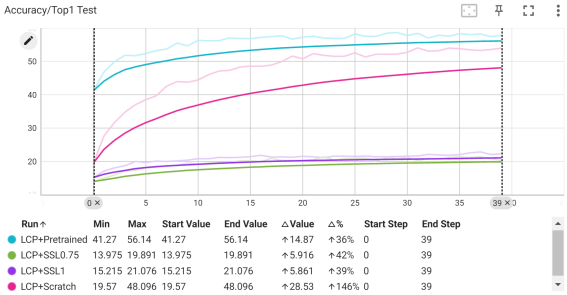
1.7 Discussion

First, we can observe that as the dataset size decreases, the model's performance also decreases (for both SSL loss and classification accuracies). This is expected, as the model has less data to learn from, resulting in lower accuracy.

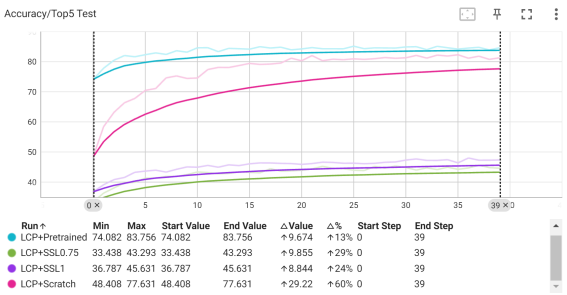
Second, the SimCLR model's performance is significantly lower than the pre-trained model on ImageNet and the model trained from scratch. This is not expected, as self-supervised learning methods like SimCLR are designed to learn powerful representations that can be fine-tuned for downstream tasks. The performance gap could be due to several factors.

1. Representation Quality: The quality of the learned representations could be a factor. SSL methods like SimCLR rely heavily on the quality of the augmentations and the contrastive learning framework to learn useful features. If the augmentations do not sufficiently capture the essential variations in the data, the representations might not be as discriminative as those learned through supervised learning.

2. Training Epochs: The duration of training could also be a factor. The results are based on training



(a) Top-1 Accuracy



(b) Top-5 Accuracy

Figure 6: More Model Performances on CIFAR-100 (Values smoothed)

for 150/200 epochs, which might not be sufficient for the SSL models to fully converge and learn effective representations. In contrast, supervised learning models might achieve better performance with fewer epochs due to the direct availability of labeled data.

3. Dataset Scale and Diversity: The pre-trained ResNet-18 model on ImageNet benefits from a significantly larger and more diverse dataset. This pre-training on a vast amount of varied images allows the model to learn very robust and generalizable features, which transfer well to the CIFAR-100 dataset. SSL models, on the other hand, were trained on a smaller and potentially less diverse dataset, limiting their ability to generalize.

To address these issues and improve the performance of SSL models, we can: extend the training duration, enhance data augmentation, use larger and more diverse datasets, and explore more SSL methods like BYOL[2] or MoCo[4]. We may try colour distortion and cropping as augmentations as the original SimCLR paper[1] suggests.

2 Task2: CNN vs. Transformer for Image Classification

2.1 Introduction

Convolutional Neural Networks (CNNs) have been the dominant architecture for image classification tasks over long periods. However, recent research has shown that Transformers, originally designed for natural language processing tasks, can also achieve competitive performance on image classification tasks. This task aims to compare the performance of CNNs and Transformers on image classification tasks using the CIFAR-100 dataset.

In this task, we have tried two pairs: **EfficientNet-B0** and **MobileViT-S** as the first pair, and **ResNet-50** and **Swin-T** as the second pair. The former small models are our first trial, where the performance is not as good as expected, so we try the latter middle models again, and observe the model parameters' impact on the performance. We train the models on CIFAR-100 with the same (pairwise) training policy and data augmentation techniques like CutMix to improve performance. And their parameter sizes are show in Table 3. We train the models on CIFAR-100 with the same training policy (for each pair) and data augmentation techniques like **CutMix** to improve performance.

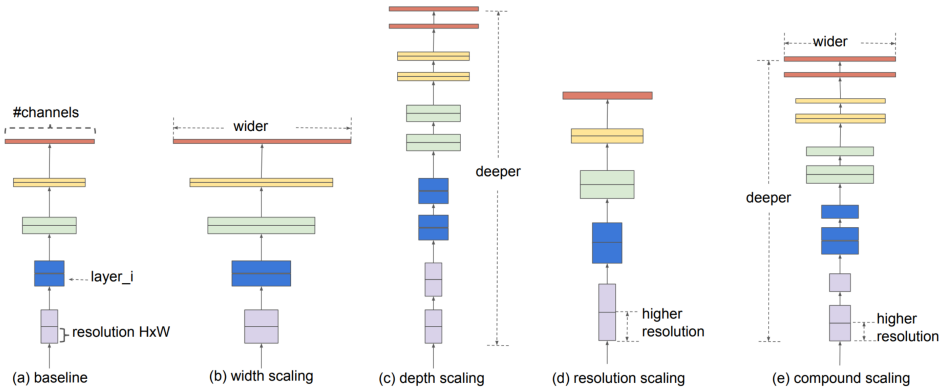


Figure 7: EfficientNet Architecture

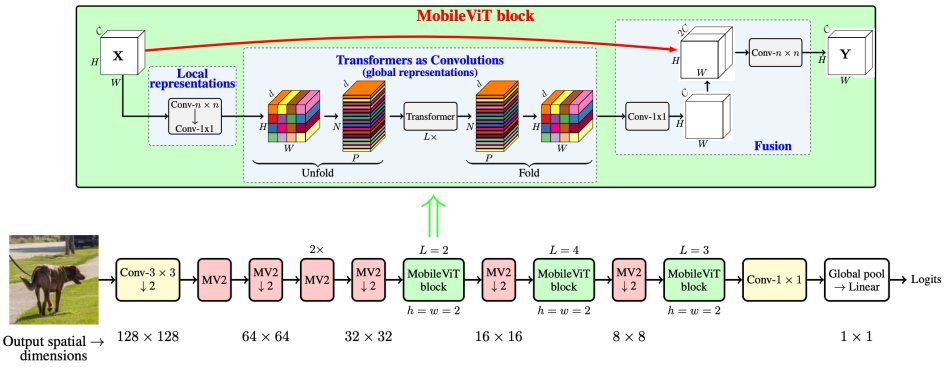


Figure 8: MobileViT Architecture

2.1.1 EfficientNet and MobileViT

EfficientNet[8] is a CNN architecture that scales all dimensions of depth, width, and resolution using a compound coefficient. Unlike traditional methods that scale these factors arbitrarily, EfficientNet uses fixed scaling coefficients to uniformly scale the network. This compound scaling method ensures that as the input image size increases, the network depth, width, and resolution scale appropriately, allowing the network to capture more detailed patterns. The EfficientNet architecture is shown in Figure 7. We use the samllest EfficientNet-B0 in this task.

MobileViT[10] is a light-weight, general-purpose, and mobile-friendly Vision Transformer. It combines the efficiency of MobileNetV3 and the power of Vision Transformer to achieve competitive performance on image classification tasks. The core idea of MobileViT is to learn global representations with transformers as convolutions. The MobileViT architecture is shown in Figure 8. We use the smallest MobileViT-S in this task.

2.1.2 ResNet-50 and Swin Transformer

ResNet is a widely used CNN architecture for image classification. The residual block proposed by He et al.[3] efficiently solves the degradation problem frequently occurred in deep networks, which is an identity shortcut connection that skips one or more layers shown in Figure 9. Thanks to this kind of mapping, the network can always backpropagate first-order information from one layer to another no matter how

deep it is, so as to address the problem of vanishing/exploding gradient. We use the ResNet-50 model in this task.

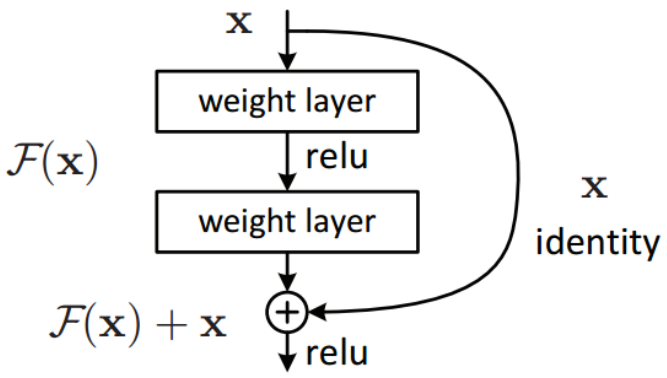


Figure 9: ResNet Block

Swin Transformer[5] is a hierarchical Vision Transformer that uses shifted windows to capture local and global information effectively. The Swin Transformer divides the input image into non-overlapping patches and processes them through a series of transformer layers. The shifted windows mechanism allows the model to capture both local and global information efficiently. The Swin Transformer architecture is shown in Figure 10. We use the smallest Swin Transformer model (Swin-T) in this task.

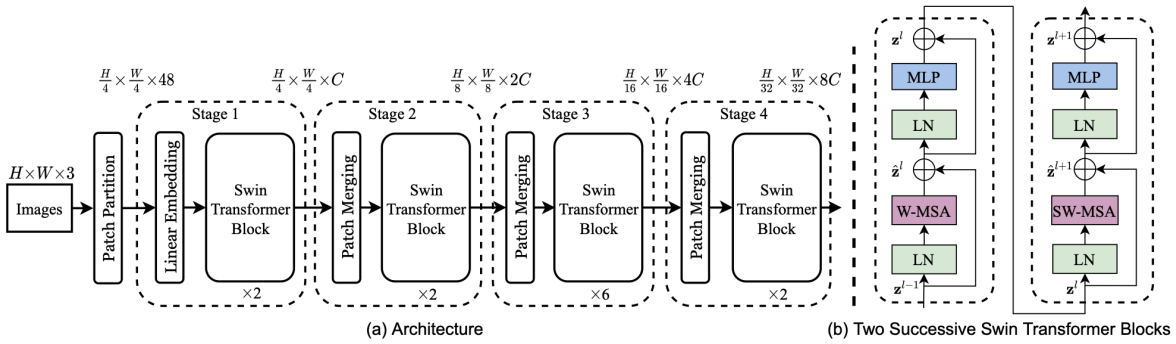


Figure 10: (a) The architecture of Swin-T; (b) two successive Swin Transformer Blocks

2.1.3 CutMix

CutMix[9] is a data augmentation technique that combines two images by cutting and pasting regions of one image onto another. This technique encourages the model to learn from the mixed images, improving generalization and robustness. A CutMix example is shown in Figure 11.

2.2 Datasets

We still use the CIFAR-100 dataset in this task, and the training and testing set split follows the same settings as in Task 1.

Figure 11: CutMix Data Augmentation[15]

2.3 Model Settings

Since all the above models are first pre-trained on ImageNet, which is 224x224, far larger than CIFAR-100, we need to modify some settings to adapt to the smaller image size and the number of classes. For EfficientNet-B0, MobileViT-S and ResNet-50, we modify the first convolution layer to kernel size 3 and stride 1, instead of 7 and 2 in the original model. We also modify the number of classes to 100, instead of 1000 in the original model. For Swin-T, we modify the patch size to 2 and window size to 4, instead of 4 and 7 in the original model. We also modify the number of classes to 100, instead of 1000 in the original model. The settings for EfficientNet-B0 and MobileViT-S are shown below.

```
# EfficientNet-B0 and MobileViT-S Settings
model_name = "EfficientNet-B0"
batch_size = 512
epochs = 300  # iterations = 300 * 50000 / 512 = 29375
learning_rate = 0.002
weight_decay = 5e-4
optimizer = "Adam"
criterion = nn.CrossEntropyLoss()
scheduler = ReduceLROnPlateau(factor=0.5,patience=5,cooldown=5,min_lr=1e-6)
warmup_epochs = 10
```

The settings for ResNet-50 and Swin-T are shown below.

```
# ResNet-50 and Swin-T Settings
batch_size = 128
epochs = 300  # iterations = 300 * 50000 / 128 = 117187
learning_rate = 0.001
weight_decay = 1e-5
optimizer = "AdamW"
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
scheduler = CosineAnnealingLR(T_max=len(train_loader), eta_min=1e-6)
warmup_epochs = 10
clip_grad_norm_(model.parameters(), 0.3)
```

The basic image augmentation and augmentation with CutMix are shown below.

```
# Basic Image Augmentation
train_transform_basic = v2.Compose([
    v2.PILToTensor(),
    v2.RandomResizedCrop(size=(32, 32), antialias=True),
    v2.RandomHorizontalFlip(p=0.5),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565,
        0.2761]),
])

test_transform = v2.Compose([
    v2.PILToTensor(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565,
        0.2761]),
])
trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True,
    num_workers=4)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=False,
    num_workers=4)

# Augmentation with CutMix
def collate_fn_cutmix(batch):
    return cutmix(*default_collate(batch))
trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True,
    num_workers=4, collate_fn=collate_fn_cutmix)
testloader = DataLoader(testset, batch_size=batch_size, shuffle=False,
    num_workers=4, collate_fn=collate_fn_cutmix)
```

We simply use classification accuracy as the evaluation metric for all models.

2.4 Hyperparameter Tuning

Here we only show the tuning results for ResNet-50 and Swin-T. A detailed tuning results can be seen in Table 2.

We observe that the model is sensitive to the learning rate and batch size values. For both models, a batch size of 128 performs well. But the learning rate of 0.005 is better for ResNet-50, while 0.001 is better for Swin-T. In training, we choose the best setting for Swin-T, since we care more about the performance of the Transformer models.

Table 2: Hyperparameter Grid Search Results (Top-1 Accuracy, each for 20 epochs)

Model Configuration	Batch Size	Learning Rate	Weight Decay	Val Loss	Val Accuracy (%)
ResNet50_AdamW	128	0.005	0.0001	3.8279	16.56
ResNet50_AdamW	**128**	**0.005**	**0.001**	**3.7020**	**18.84**
ResNet50_AdamW	128	0.001	0.001	3.7301	17.72
ResNet50_AdamW	128	0.001	0.0001	4.0010	12.35
ResNet50_AdamW	128	0.001	1e-05	4.0144	11.63
ResNet50_AdamW	128	0.001	1e-05	4.0602	11.34
ResNet50_AdamW	256	0.005	1e-05	3.7996	15.68
ResNet50_AdamW	256	0.005	0.0001	3.8697	15.57
ResNet50_AdamW	256	0.005	0.001	3.9381	14.05
ResNet50_AdamW	256	0.001	1e-05	4.1399	8.91
ResNet50_AdamW	256	0.001	0.0001	4.1134	9.62
ResNet50_AdamW	256	0.001	0.001	4.1282	9.63
ResNet50_AdamW	512	0.005	0.0001	3.9988	11.76
ResNet50_AdamW	1024	0.005	0.0001	4.0276	11.35
Swin-T_AdamW	128	0.005	1e-05	3.7502	16.87
Swin-T_AdamW	128	0.005	0.0001	3.9698	11.80
Swin-T_AdamW	128	0.005	0.001	4.2237	7.21
Swin-T_AdamW	128	0.001	0.001	4.0481	10.72
Swin-T_AdamW	128	0.001	0.0001	3.6752	18.56
Swin-T_AdamW	**128**	**0.001**	**1e-05**	**3.5197**	**22.92**
Swin-T_AdamW	256	0.005	1e-05	3.7768	16.00
Swin-T_AdamW	256	0.005	0.0001	3.9655	12.37
Swin-T_AdamW	256	0.005	0.001	4.1455	8.60
Swin-T_AdamW	256	0.001	1e-05	3.5735	22.19
Swin-T_AdamW	256	0.001	0.0001	3.6271	20.27
Swin-T_AdamW	256	0.001	0.001	3.9782	11.94
Swin-T_AdamW	512	0.001	1e-05	3.5375	21.39
Swin-T_AdamW	1024	0.001	1e-05	3.6858	19.40

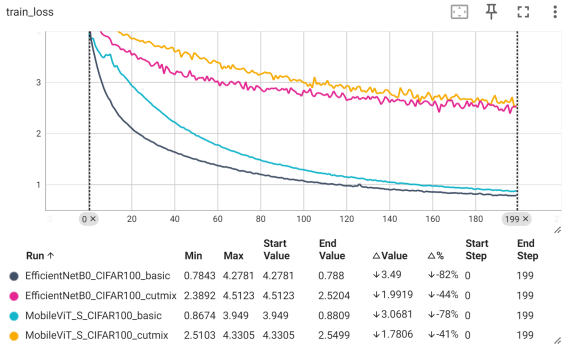
2.5 Experiment Results

We present the training results for all models in Table 3.

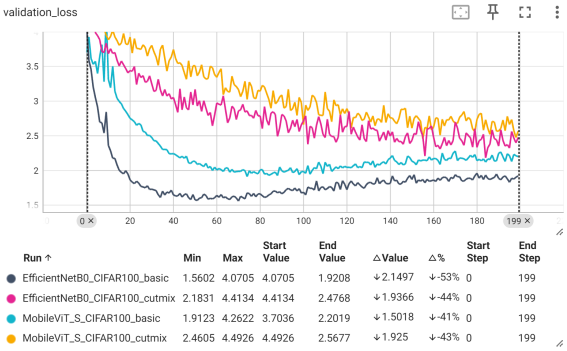
Table 3: CNN and Transformer model performances on CIFAR-100

Architecture	Augmentation	#Params	Epochs	Train Loss	Val Loss	Best Val Accuracy (%)
EfficientNet-B0	Cutmix	5.3M	200	2.5204	2.4768	50.67
MobileViT-S	Cutmix	5.6M	200	2.5499	2.5677	46.93
EfficientNet-B0	Basic	5.3M	200	0.788	1.9208	63.23
MobileViT-S	Basic	5.6M	200	0.8809	2.2019	50.67
ResNet-50	Cutmix	25.56M	300	2.1283	2.6981	59.32 (57.40/200 epochs)
Swin-T	Cutmix	28.5M	300	2.5881	3.0105	47.39 (46.12/200 epochs)
ResNet-50	Basic	25.56M	300	1.0047	2.6981	69.82 (69.00/200 epochs)
Swin-T	Basic	28.5M	300	1.229	2.5184	53.82 (51.53/200 epochs)

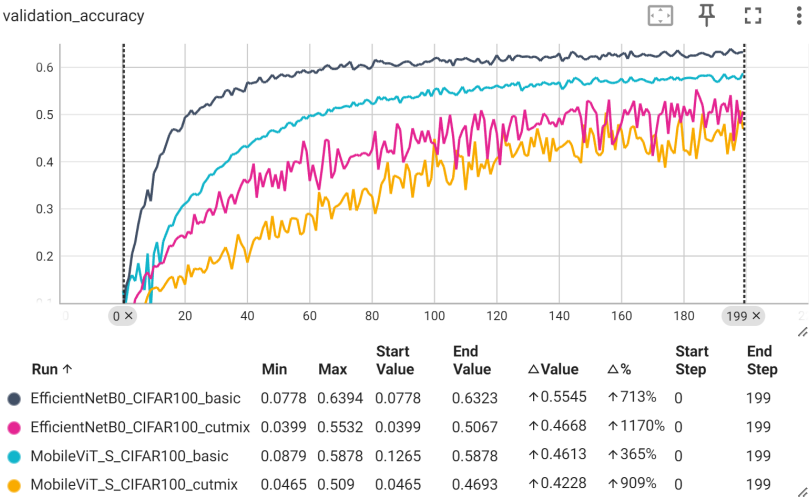
In our experiments, we first train the EfficientNet-B0 and MobileViT-S on CIFAR-100 for 200 epochs, and the training curves are shown in Figure 12.



(a) Training Loss for Small Models



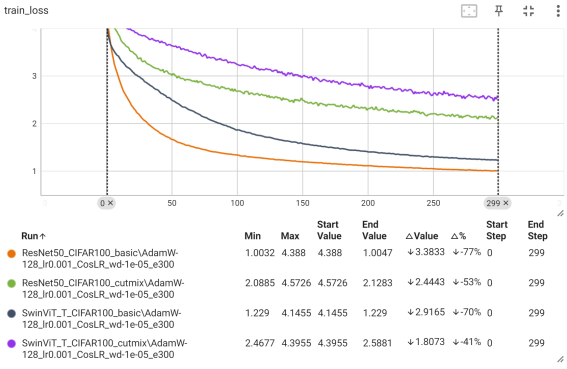
(b) Validation Loss for Small Models



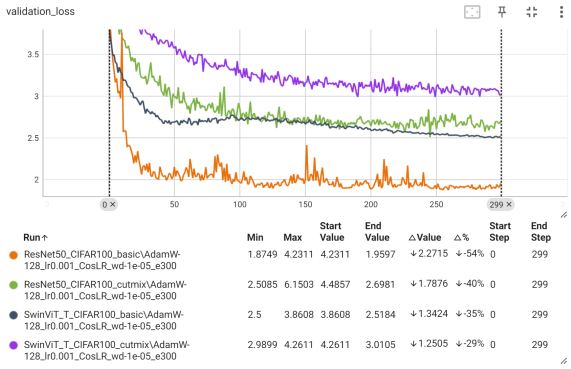
(c) Validation Accuracy for Small Models

Figure 12: Training Curves for Small Models

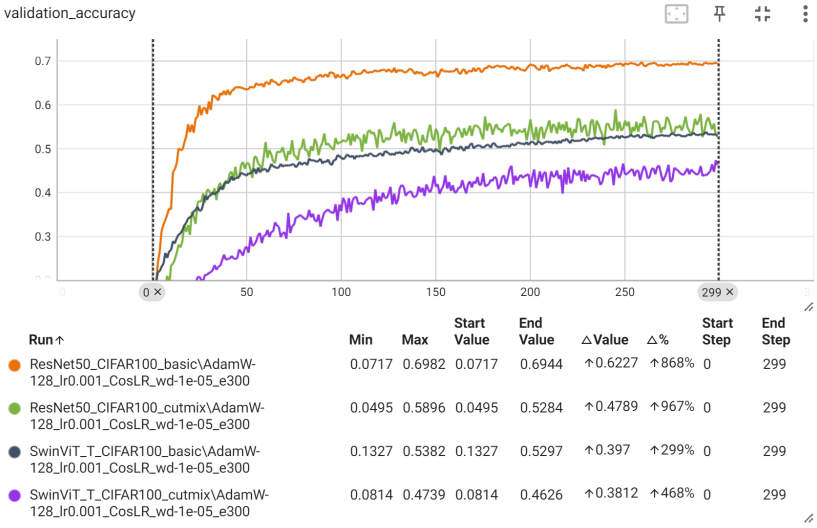
We then train the ResNet-50 and Swin-T on CIFAR-100 for 300 epochs, as shown in Figure 13.



(a) Training Loss for Middle Models



(b) Validation Loss for Middle Models



(c) Validation Accuracy for Middle Models

Figure 13: Training Curves for Middle Models

2.6 Discussion

From the results above, we can observe the following:

1. The small models (EfficientNet-B0 and MobileViT-S) achieve lower performance compared to the middle models (ResNet-50 and Swin-T). This is expected, but their difference is not significant, especially for Tansformer models. This may be due to the small size of the dataset and different expressive power of the models despite parameter sizes.

2. The models trained with CutMix augmentation perform worse than those trained with basic augmentation, but the overfitting problem is largely reduced. This is consistent with the idea that CutMix can improve the model's generalization and robustness by encouraging the model to learn from mixed images. But for low resolution images like CIFAR-100, the mixed images may perturb the model too much, leading to worse performance.

3. CNN-based models outperform Transformer-based models in both pairs. This is probably due to the small size of the dataset. Transformer models, which rely heavily on self-attention mechanisms, often require larger amounts of data to achieve comparable performance to CNNs. The limited amount of data in CIFAR-100 might not be sufficient for Transformer models to learn robust representations.

3 Task3: NeRF: 3D Reconstruction and View Synthesis

3.1 Introduction

NeRF (Neural Radiance Fields)[6] is a state-of-art method for synthesizing novel views of complex scenes by learning a mapping from input images to a continuous 5D representation of radiance fields. The model can generate high-quality photo-realistic images from novel viewpoints, enabling applications like view synthesis, 3D reconstruction, and image-based rendering.

The NeRF model consists of two components: a neural network that predicts the volume density and view-dependent emitted radiance at any 3D point, and a differentiable volume rendering algorithm that integrates the radiance field along the camera rays to generate the final image. The model is trained using a set of input images and corresponding camera poses to learn the radiance field representation. A sketch of the NeRF model is shown in Figure 14.

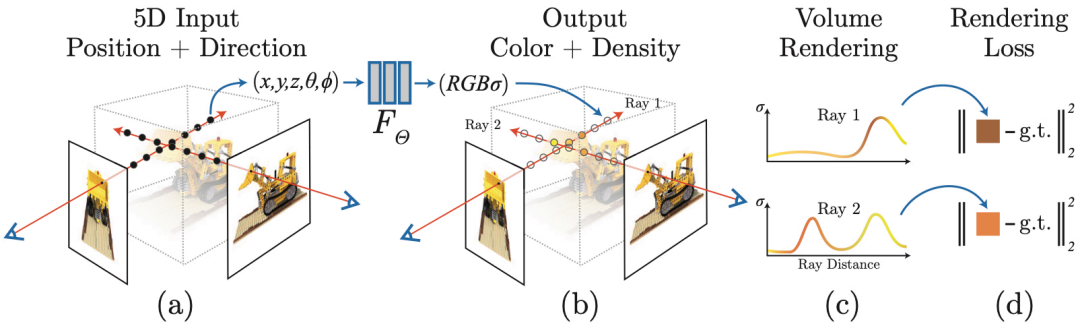


Figure 14: NeRF Model Pipeline

This task aims to implement NeRF and train it on our own dataset for object reconstruction and video rendering tasks. We use the PyTorch implementation of NeRF[14] as the base code and adapt it for training.

3.2 Datasets

We have taken two videos and extracted frames from them to create our dataset. Both videos are of static objects in a continuous moving view, the first being of a stationery bag and the second of steamed buns. Two sample frame images are shown in Figure 15.

We construct the dataset by following the steps below:

1. Extract frames from the videos at a fixed frame rate 5, and obtain 128 and 109 images for the two videos, respectively.

2. Pose estimation: Use COLMAP to estimate the camera poses for each image. COLMAP is a general-purpose Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline with a graphical and command-line interface. Do feature extraction, matching, and sparse reconstruction to estimate the camera poses. A reconstruction example is shown in Figure 16.

3. Use LLFF[16] to convert the COLMAP reconstruction to the NeRF dataset format. LLFF is a

(a) Sample Image 1

(b) Sample Image 2

Figure 15: Sample Images from the Dataset

method for training neural radiance fields on large-scale scenes. It converts the COLMAP reconstruction to the NeRF dataset format, including camera poses, intrinsics, and extrinsics.

4. Load the original images and LLFF format data into the NeRF model for training.

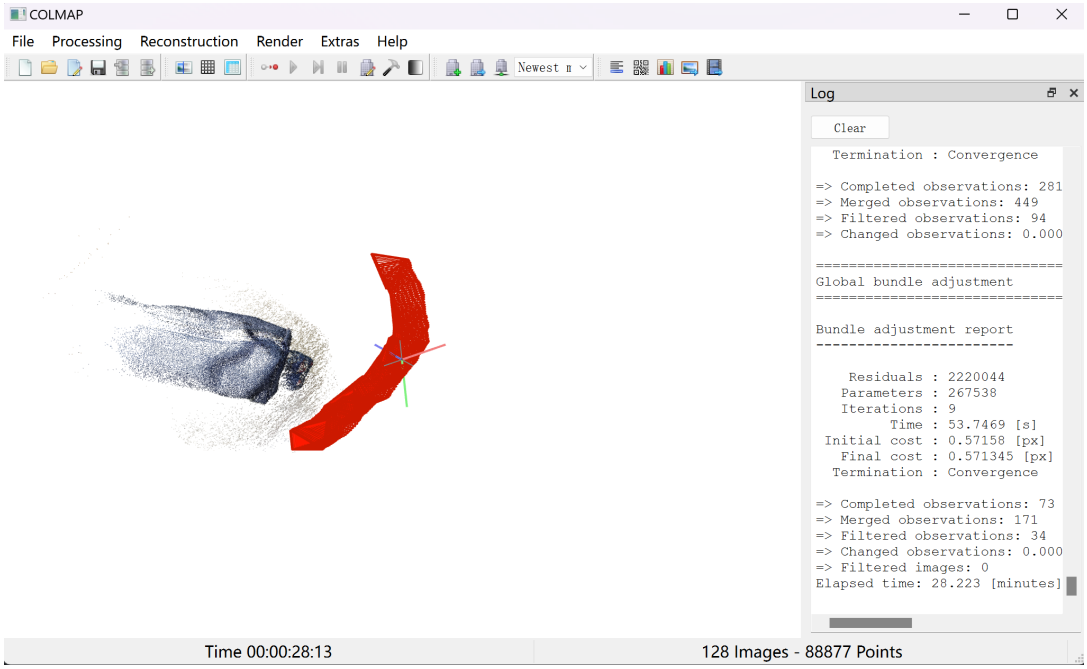


Figure 16: COLMAP Reconstruction Example

3.3 Model Settings

We split the dataset into training and test sets with a skip factor of 8, i.e. every 8th image is used for validation, and the spilt ratio is thus 7:1.

Some of the key model settings are as follows:

```
# NeRF Model Settings
netdepth = 8  # number of layers in the NeRF model
netwidth = 256  # channels per layer
netdepth_fine = 8  # number of layers in the fine network
netwidth_fine = 256  # channels per layer in the fine network
N_rand = 1024  # number of random rays per gradient step
no_batching = True  # only take random rays from 1 image at a time
factor = 8  # downsampling factor
llffhold = 8  # take every 1/N images as LLFF test set
N_samples = 64  # number of coarse samples per ray
N_importance = 64  # number of additional fine samples per ray
lrate = 5e-4  # learning rate
lrate_decay = 250  # exponential learning rate decay
optimizer = 'Adam'
epochs = 100k for "bag" and 50k for "steamedbuns"  # also iterations
```

We use MSE loss for training NeRF, namely the mean squared error between the predicted and ground truth colors of the images.

The evaluation metric is the PSNR (Peak Signal-to-Noise Ratio) between the predicted and ground truth images. PSNR is a commonly used metric to evaluate the quality of image reconstruction tasks. It measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. A higher PSNR value indicates better image quality.

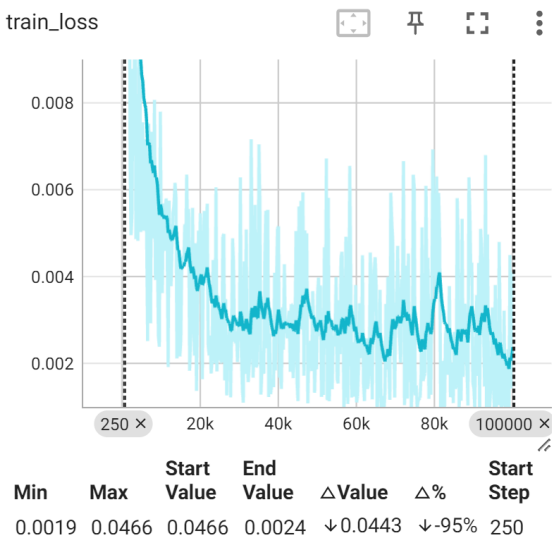
3.4 Experiment Results

We present the training results for NeRF in Table 4.

Table 4: NeRF Performance on the Dataset

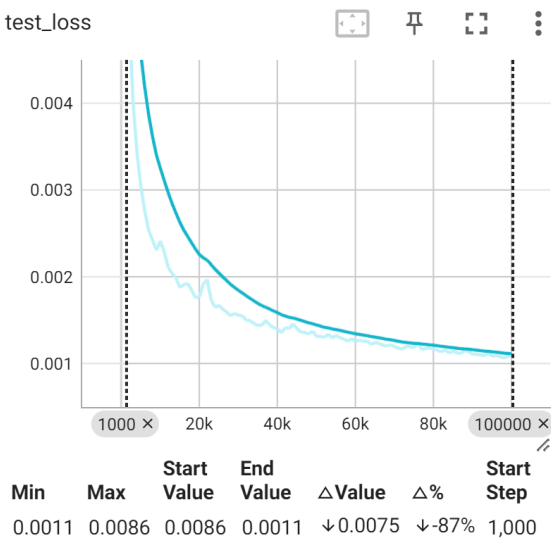
Video	Epochs	Train Loss	Test Loss	Test PSNR
Bag	100k	0.0041	0.0011	30.443
Steamed Buns	50k	0.0037	0.0009	30.4343

The training curves for NeRF are shown in Figure 17.



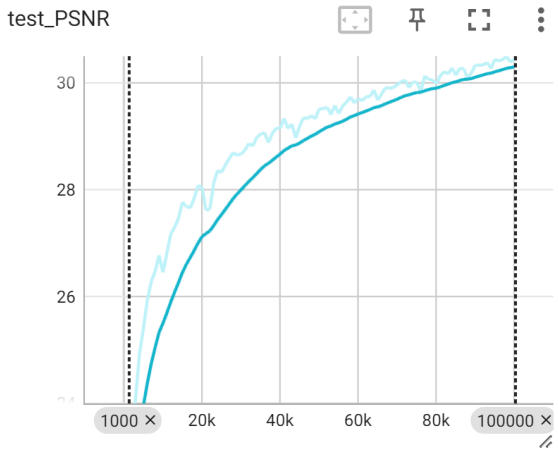
Min	Max	Start Value	End Value	△Value	△%	Start Step
0.0019	0.0466	0.0466	0.0024	↓0.0443	↓-95%	250

(a) Training Loss for Bag Video



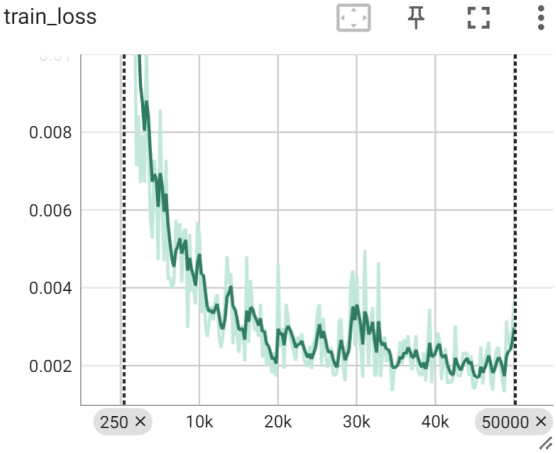
Min	Max	Start Value	End Value	△Value	△%	Start Step
0.0011	0.0086	0.0086	0.0011	↓0.0075	↓-87%	1,000

(b) Test Loss for Bag Video



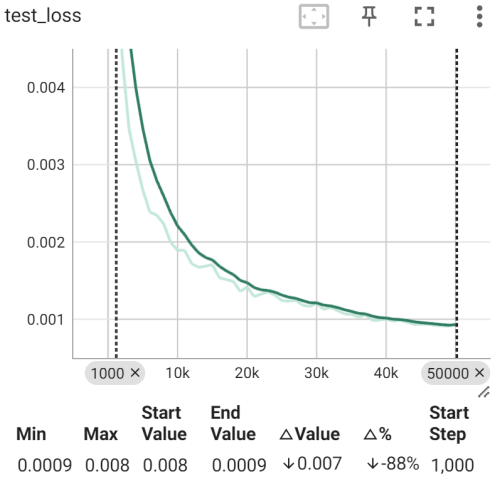
Min	Max	Start Value	End Value	△Value	△%	Sta Ste
21.0267	30.3007	21.0267	30.3007	↑9.2741	↑44%	1,0

(c) Test PSNR for Bag Video



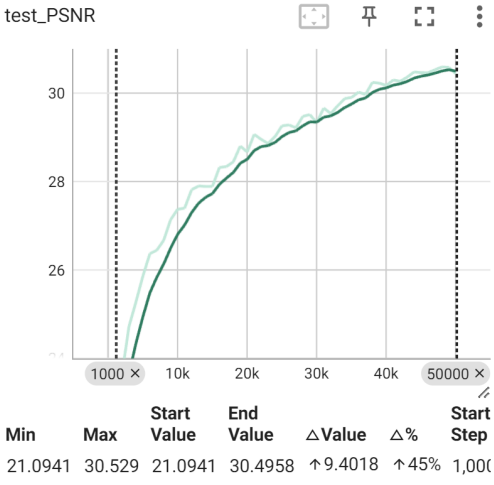
Min	Max	Start Value	End Value	△Value	△%	Start Step
0.0017	0.0413	0.0413	0.003	↓0.0384	↓-93%	250

(d) Training Loss for Steamed Buns Video



Min	Max	Start Value	End Value	△Value	△%	Start Step
0.0009	0.008	0.008	0.0009	↓0.007	↓-88%	1,000

(e) Test Loss for Steamed Buns Video



Min	Max	Start Value	End Value	△Value	△%	Start Step
21.0941	30.529	21.0941	30.4958	↑9.4018	↑45%	1,00(

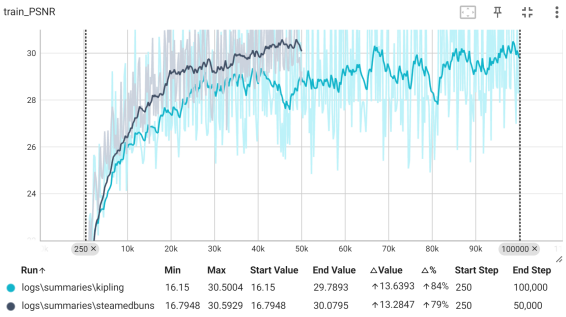
(f) Test PSNR for Steamed Buns Video

Figure 17: Training Curves for NeRF

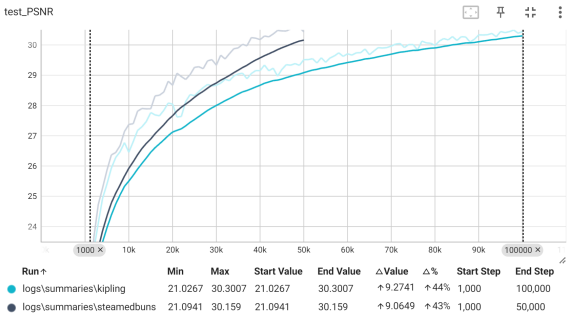
3.5 Discussion

From the results above, we can observe that the NeRF training loss decreases significantly at some initial epochs and fluctuates around a certain value after a certain number of epochs. However, the decrease of test loss and increase of test PSNR is more steady over time. This indicates that the model is learning to reconstruct the images better over time, and perhaps we have not overfitted the model to the training data.

We further overlap the two curves for each video to compare the training and test PSNRs, as shown in Figure 18.



(a) Comparison for train PSNRs



(b) Comparison for test PSNRs

Figure 18: Comparison of PSNRs for NeRF

From the results above, we can observe that the NeRF model performs well on both videos, but the "Steamed Buns" video has better performance than the "Bag" video in terms of quicker increase in PSNR. It only needs half the epochs to achieve a similar PSNR value compared to the "Bag" video. And from the plot we can forecast that the "Steamed Buns" video will achieve a higher PSNR value than the "Bag" video at 100k epochs.

This is probably due to the complexity of the scenes in the videos, where the "Steamed Buns" video has more detailed and textured objects compared to the "Bag" video. The NeRF model can better capture the radiance fields of complex scenes with more detailed objects, leading to better reconstruction and view synthesis results.

References

[1] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," in *Proc. of International Conference on Machine Learning (ICML)*, 2020.

[2] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning," *arXiv preprint arXiv:2006.07733*, 2020.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," in *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[5] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows (arXiv:2103.14030). arXiv. https://doi.org/10.48550/arXiv.2103.14030

[6] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis (arXiv:2003.08934). arXiv. https://doi.org/10.48550/arXiv.2003.08934

[7] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[8] Tan, M., & Le, Q. V. (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (arXiv:1905.11946). arXiv. http://arxiv.org/abs/1905.11946

[9] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., Yoo, Y. J., & Yoo, S. (2019). CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features (arXiv:1905.04899). arXiv. http://arxiv.org/abs/1905.04899

[10] Mehta, S., & Rastegari, M. (2022). MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer (arXiv:2110.02178). arXiv. http://arxiv.org/abs/2110.02178

[11] STL-10 dataset

[12] CIFAR-100 dataset

[13] SimCLR implementation

[14] NeRF PyTorch implementation

[15] CutMix visualization

[16] LLFF implementation