

```
# installing timm library for better model loading
!pip install timm -q
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
import torchvision.transforms as transforms
from torchvision.datasets import CIFAR10
import timm
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
import os
from google.colab import drive

# mount drive to save models
drive.mount('/content/drive')

save_path = '/content/drive/MyDrive/CIFAR10_Challenge'
os.makedirs(save_path, exist_ok=True)

# check if gpu is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Mounted at /content/drive
Using device: cuda
```

```
# set seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)

# Data augmentation for training
# using 128x128 because its faster than 224 but still gives good results
transform_train = transforms.Compose([
    transforms.Resize(128),
    transforms.RandomCrop(128, padding=16), # random crops help with generalization
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(0.2, 0.2, 0.2), # slight color variations
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # imagenet stats
])

# simpler transforms for validation/test
transform_test = transforms.Compose([
    transforms.Resize(128),
```

```
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

print("Downloading CIFAR-10 dataset...")
full_train = CIFAR10(root='./data', train=True, download=True, transform=transform_train)
test_set = CIFAR10(root='./data', train=False, download=True, transform=transform_test)
```

Downloading CIFAR-10 dataset...  
100%|██████████| 170M/170M [00:05<00:00, 30.2MB/s]

```
# split into 80-10-10 as required
train_set, val_set = random_split(full_train, [45000, 5000],
                                    generator=torch.Generator().manual_seed(42))
test_set = torch.utils.data.Subset(test_set, range(5000))

print(f"Train samples: {len(train_set)} | Val samples: {len(val_set)} | Test samples: {len(test_set)}")
```

Train samples: 45000 | Val samples: 5000 | Test samples: 5000

```
# batch size of 128 works well with gpu memory
bs = 128

train_loader = DataLoader(train_set, batch_size=bs, shuffle=True,
                         num_workers=2, pin_memory=True, persistent_workers=True)
val_loader = DataLoader(val_set, batch_size=bs, shuffle=False,
                         num_workers=2, pin_memory=True, persistent_workers=True)
test_loader = DataLoader(test_set, batch_size=bs, shuffle=False,
                         num_workers=2, pin_memory=True)

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Load pretrained ResNet50 - this is the key to transfer learning
print("\nLoading ResNet50 with ImageNet weights...")
model = timm.create_model('resnet50', pretrained=True, num_classes=10)
model = model.to(device)

total_params = sum(p.numel() for p in model.parameters())
print(f"Total parameters: {total_params:,}")
```

```
Loading ResNet50 with ImageNet weights...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
model.safetensors: 100%                                         102M/102M [00:01<00:00, 94.4MB/s]
Total parameters: 23,528,522
```

```
scaler = torch.cuda.amp.GradScaler()

# loss function
criterion = nn.CrossEntropyLoss()

# First we'll train just the last layer (faster convergence)
for name, param in model.named_parameters():
    if 'fc' not in name: # freeze everything except the classifier
        param.requires_grad = False

optimizer = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()),
                       lr=1e-3, weight_decay=0.01)

epochs_stage1 = 5 # first stage: train classifier only
epochs_stage2 = 15 # second stage: fine-tune everything
total_eps = epochs_stage1 + epochs_stage2

scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs_stage1)

# tracking metrics
metrics = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': [], 'lr': []}

/tmp/ipython-input-2019566532.py:1: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)` instead.
scaler = torch.cuda.amp.GradScaler()
```

```
def train_one_epoch(model, loader, criterion, optimizer, scaler):
    """training loop for one epoch"""
    model.train()
    running_loss = 0
    correct = 0
    total = 0

    pbar = tqdm(loader, desc='Training')
    for imgs, labels in pbar:
        imgs, labels = imgs.to(device), labels.to(device)
```

```
optimizer.zero_grad()

# mixed precision forward pass
with torch.cuda.amp.autocast():
    outputs = model(imgs)
    loss = criterion(outputs, labels)

    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()

    running_loss += loss.item()
    _, preds = outputs.max(1)
    total += labels.size(0)
    correct += preds.eq(labels).sum().item()

    pbar.set_postfix({'loss': f'{loss.item():.3f}', 'acc': f'{100.*correct/total:.1f}%'})

return running_loss / len(loader), correct / total
```

```
@torch.no_grad()
def evaluate(model, loader, criterion):
    """evaluation function"""
    model.eval()
    running_loss = 0
    correct = 0
    total = 0

    for imgs, labels in tqdm(loader, desc='Validating'):
        imgs, labels = imgs.to(device), labels.to(device)

        with torch.cuda.amp.autocast():
            outputs = model(imgs)
            loss = criterion(outputs, labels)

            running_loss += loss.item()
            _, preds = outputs.max(1)
            total += labels.size(0)
            correct += preds.eq(labels).sum().item()

    return running_loss / len(loader), correct / total
```

```
print("\n" + "="*70)
print("STAGE 1: Training classifier head (5 epochs)")
print("=*70)

best_acc = 0
```

```
for ep in range(epochs_stage1):
    print(f"\nEpoch {ep+1}/{epochs_stage1}")

    tr_loss, tr_acc = train_one_epoch(model, train_loader, criterion, optimizer, scaler)
    v_loss, v_acc = evaluate(model, val_loader, criterion)
    scheduler.step()

    metrics['train_loss'].append(tr_loss)
    metrics['train_acc'].append(tr_acc)
    metrics['val_loss'].append(v_loss)
    metrics['val_acc'].append(v_acc)
    metrics['lr'].append(optimizer.param_groups[0]['lr'])

    print(f"Train acc: {tr_acc*100:.2f}% | Val acc: {v_acc*100:.2f}%")

    # save if this is the best model so far
    if v_acc > best_acc:
        best_acc = v_acc
        torch.save({
            'epoch': ep,
            'model': model.state_dict(),
            'acc': v_acc
        }, f'{save_path}/level1_best.pth')
        print(f"Saved checkpoint! Best acc: {v_acc*100:.2f}%")
```

```
=====
STAGE 1: Training classifier head (5 epochs)
=====
```

```
Epoch 1/5
Training:  0%|          | 0/352 [00:00<?, ?it/s]/tmp/ipython-input-2864813697.py:15: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch
      with torch.cuda.amp.autocast():
Training: 100%|██████| 352/352 [01:21<00:00,  4.31it/s, loss=0.980, acc=67.6%]
Validating:  0%|          | 0/40 [00:00<?, ?it/s]/tmp/ipython-input-2566612463.py:12: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torc
      with torch.cuda.amp.autocast():
Validating: 100%|██████| 40/40 [00:09<00:00,  4.13it/s]
Train acc: 67.60% | Val acc: 72.56%
Saved checkpoint! Best acc: 72.56%
```

```
Epoch 2/5
Training: 100%|██████| 352/352 [01:19<00:00,  4.41it/s, loss=0.783, acc=74.2%]
Validating: 100%|██████| 40/40 [00:09<00:00,  4.23it/s]
Train acc: 74.19% | Val acc: 74.86%
Saved checkpoint! Best acc: 74.86%
```

```
Epoch 3/5
Training: 100%|██████| 352/352 [01:19<00:00,  4.40it/s, loss=0.690, acc=75.5%]
Validating: 100%|██████| 40/40 [00:09<00:00,  4.24it/s]
Train acc: 75.52% | Val acc: 75.70%
Saved checkpoint! Best acc: 75.70%
```

```
Epoch 4/5
```

```
Training: 100%|██████████| 352/352 [01:21<00:00,  4.33it/s, loss=0.855, acc=76.4%]
Validating: 100%|██████████| 40/40 [00:08<00:00,  4.74it/s]
Train acc: 76.38% | Val acc: 77.14%
Saved checkpoint! Best acc: 77.14%

Epoch 5/5
Training: 100%|██████████| 352/352 [01:21<00:00,  4.33it/s, loss=0.653, acc=76.9%]
Validating: 100%|██████████| 40/40 [00:07<00:00,  5.14it/s]Train acc: 76.93% | Val acc: 76.08%
```

```
# Now unfreeze everything and fine-tune
print("\n" + "="*70)
print("STAGE 2: Fine-tuning full model (15 epochs)")
print("="*70)

for param in model.parameters():
    param.requires_grad = True

# lower learning rate for fine-tuning
optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=0.01)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs_stage2)

# early stopping setup
patience_max = 5
no_improve = 0

for ep in range(epochs_stage2):
    print(f"\nEpoch {epochs_stage1 + ep + 1}/{total_eps}")

    tr_loss, tr_acc = train_one_epoch(model, train_loader, criterion, optimizer, scaler)
    v_loss, v_acc = evaluate(model, val_loader, criterion)
    scheduler.step()

    metrics['train_loss'].append(tr_loss)
    metrics['train_acc'].append(tr_acc)
    metrics['val_loss'].append(v_loss)
    metrics['val_acc'].append(v_acc)
    metrics['lr'].append(optimizer.param_groups[0]['lr'])

    print(f"Train acc: {tr_acc*100:.2f}% | Val acc: {v_acc*100:.2f}%")

    if v_acc > best_acc:
        best_acc = v_acc
        no_improve = 0
        torch.save({
            'epoch': ep,
            'model': model.state_dict(),
            'acc': v_acc
        }, f'{save_path}/level1_best.pth')
    print(f"Saved checkpoint! Best acc: {v_acc*100:.2f}%")
```

```
else:  
    no_improve += 1  
    if no_improve >= patience_max:  
        print(f"\nStopping early - no improvement for {patience_max} epochs")  
        break  
  
# Load best model for testing  
print("\nLoading best checkpoint for final evaluation...")  
ckpt = torch.load(f'{save_path}/level1_best.pth')  
model.load_state_dict(ckpt['model'])  
  
test_loss, test_acc = evaluate(model, test_loader, criterion)  
  
print("\n" + "="*70)  
print("FINAL RESULTS")  
print("="*70)  
print(f"Test Accuracy: {test_acc*100:.2f}%")  
print(f"Best Val Acc: {best_acc*100:.2f}%")  
print(f"Target: 85%")  
print(f"Status: {'PASSED ✓' if test_acc >= 0.85 else 'FAILED'}")
```

```
=====  
STAGE 2: Fine-tuning full model (15 epochs)  
=====
```

```
Epoch 6/20  
Training: 0%| [ 0/352 [00:00<?, ?it/s]/tmp/ipython-input-2864813697.py:15: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch  
with torch.cuda.amp.autocast():  
Training: 100%|██████| 352/352 [01:31<00:00,  3.86it/s, loss=0.338, acc=86.7%]  
Validating: 0%| [ 0/40 [00:00<?, ?it/s]/tmp/ipython-input-2566612463.py:12: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `tor  
with torch.cuda.amp.autocast():  
Validating: 100%|██████| 40/40 [00:09<00:00,  4.26it/s]  
Train acc: 86.72% | Val acc: 90.60%  
Saved checkpoint! Best acc: 90.60%
```

```
Epoch 7/20  
Training: 100%|██████| 352/352 [01:32<00:00,  3.82it/s, loss=0.196, acc=92.4%]  
Validating: 100%|██████| 40/40 [00:09<00:00,  4.27it/s]  
Train acc: 92.39% | Val acc: 92.98%  
Saved checkpoint! Best acc: 92.98%
```

```
Epoch 8/20  
Training: 100%|██████| 352/352 [01:32<00:00,  3.81it/s, loss=0.141, acc=94.3%]  
Validating: 100%|██████| 40/40 [00:09<00:00,  4.29it/s]  
Train acc: 94.32% | Val acc: 94.28%  
Saved checkpoint! Best acc: 94.28%
```

```
Epoch 9/20  
Training: 100%|██████| 352/352 [01:32<00:00,  3.82it/s, loss=0.177, acc=95.5%]  
Validating: 100%|██████| 40/40 [00:09<00:00,  4.26it/s]  
Train acc: 95.48% | Val acc: 95.20%  
Saved checkpoint! Best acc: 95.20%
```

```
Epoch 10/20
Training: 100%|██████████| 352/352 [01:32<00:00,  3.82it/s, loss=0.137, acc=96.1%]
Validating: 100%|██████████| 40/40 [00:09<00:00,  4.33it/s]
Train acc: 96.11% | Val acc: 95.40%
Saved checkpoint! Best acc: 95.40%
```

```
Epoch 11/20
Training: 100%|██████████| 352/352 [01:32<00:00,  3.81it/s, loss=0.150, acc=96.7%]
Validating: 100%|██████████| 40/40 [00:09<00:00,  4.26it/s]
Train acc: 96.73% | Val acc: 95.52%
Saved checkpoint! Best acc: 95.52%
```

```
Epoch 12/20
Training: 100%|██████████| 352/352 [01:31<00:00,  3.84it/s, loss=0.049, acc=97.2%]
Validating: 100%|██████████| 40/40 [00:09<00:00,  4.28it/s]
Train acc: 97.18% | Val acc: 95.62%
Saved checkpoint! Best acc: 95.62%
```

```
Epoch 13/20
Training: 100%|██████████| 352/352 [01:31<00:00,  3.83it/s, loss=0.119, acc=97.4%]
Validating: 100%|██████████| 40/40 [00:09<00:00,  4.34it/s]
Train acc: 97.39% | Val acc: 96.08%
Saved checkpoint! Best acc: 96.08%
```

```
# Plot training curves
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.plot(metrics['train_loss'], label='Train', linewidth=2)
ax1.plot(metrics['val_loss'], label='Val', linewidth=2)
ax1.axvline(x=epochs_stage1, color='red', linestyle='--', alpha=0.5, label='Unfreeze')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.set_title('Training Loss')
ax1.legend()
ax1.grid(True, alpha=0.3)

ax2.plot([a*100 for a in metrics['train_acc']], label='Train', linewidth=2)
ax2.plot([a*100 for a in metrics['val_acc']], label='Val', linewidth=2)
ax2.axvline(x=epochs_stage1, color='red', linestyle='--', alpha=0.5, label='Unfreeze')
ax2.axhline(y=85, color='green', linestyle='--', alpha=0.5, label='Target 85%')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy (%)')
ax2.set_title('Training Accuracy')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(f'{save_path}/level1_curves.png', dpi=150)
plt.show()
```

```
# Check per-class performance
print("\nPer-class accuracy breakdown:")
class_correct = [0] * 10
class_total = [0] * 10

model.eval()
with torch.no_grad():
    for imgs, labels in test_loader:
        imgs, labels = imgs.to(device), labels.to(device)
        outputs = model(imgs)
        _, preds = outputs.max(1)

        for i in range(len(labels)):
            lbl = labels[i].item()
            class_correct[lbl] += (preds[i] == labels[i]).item()
            class_total[lbl] += 1

    for i in range(10):
        acc = 100 * class_correct[i] / class_total[i]
        print(f'{class_names[i]:12s}: {acc:5.2f}%')

# visualize per-class accuracy
plt.figure(figsize=(10, 5))
class_accs = [100 * class_correct[i] / class_total[i] for i in range(10)]
bars = plt.bar(class_names, class_accs, color='steelblue')

# color code based on performance
for i, bar in enumerate(bars):
    if class_accs[i] >= 85:
        bar.set_color('green')
    elif class_accs[i] >= 75:
        bar.set_color('orange')
    else:
        bar.set_color('red')

plt.axhline(y=85, color='black', linestyle='--', alpha=0.5)
plt.xlabel('Class')
plt.ylabel('Accuracy (%)')
plt.title('Per-Class Test Accuracy')
plt.xticks(rotation=45, ha='right')
plt.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.savefig(f'{save_path}/level1_per_class.png', dpi=150)
plt.show()
```



**Training Loss**

```
# Save summary
summary_text = f"""
LEVEL 1 COMPLETION SUMMARY
{'='*70}

Model: ResNet50 (Transfer Learning)
Strategy: Two-stage training (freeze then fine-tune)
Dataset: CIFAR-10 (80-10-10 split)

RESULTS:
Test Accuracy: {test_acc*100:.2f}%
Best Val Accuracy: {best_acc*100:.2f}%
Total Epochs: {len(metrics['train_acc'])}

TARGET: 85%
STATUS: {'PASSED ✓' if test_acc >= 0.85 else 'NEEDS IMPROVEMENT'}
```

**SAVED FILES:**

- {save\_path}/level1\_best.pth
- {save\_path}/level1\_curves.png
- {save\_path}/level1\_per\_class.png

Next step: Level 2 (target 90%+)
{'='\*70}
"""

```
with open(f'{save_path}/level1_summary.txt', 'w') as f:
    f.write(summary_text)

print(summary_text)
print("\nLevel 1 complete!")
```

```
ship      : 97.82%
LEVEL 1 COMPLETION SUMMARY
```

**Per-Class Test Accuracy**

