```
!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
from collections import Counter

# Mount drive
drive.mount('/content/drive')
save_dir = '/content/drive/MyDrive/CIFAR10_Challenge'
os.makedirs(save_dir, exist_ok=True)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Set seeds
torch.manual_seed(42)
np.random.seed(42)

print("\n" + "="*70)
print("LEVEL 4: ENSEMBLE LEARNING")
print("="*70)
print("\nPart 1: Train ConvNeXt-Tiny (new model)")
print("Part 2: Ensemble with ResNet50 + EfficientNet-B2")
print("="*70)
```

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

======================================================================
LEVEL 4: ENSEMBLE LEARNING
======================================================================

Part 1: Train ConvNeXt-Tiny (new model)
Part 2: Ensemble with ResNet50 + EfficientNet-B2
======================================================================
```

```
print("\n" + "="*70)
print("PART 1: Training ConvNeXt-Tiny")
```

```
print("="*70)
print("\nConvNeXt is a modern CNN that applies transformer design principles")
print("to traditional convolutions. It's one of the best pure CNN architectures.")

# Data preparation
train_transforms = transforms.Compose([
    transforms.Resize(128),
    transforms.RandomCrop(128, padding=16),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    transforms.RandomErasing(p=0.3)
])

test_transforms = transforms.Compose([
    transforms.Resize(128),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

print("\nLoading CIFAR-10...")
train_dataset = CIFAR10(root='./data', train=True, download=True, transform=train_transforms)
test_dataset = CIFAR10(root='./data', train=False, download=True, transform=test_transforms)
```

```
======================================================================
PART 1: Training ConvNeXt-Tiny
======================================================================

ConvNeXt is a modern CNN that applies transformer design principles
to traditional convolutions. It's one of the best pure CNN architectures.

Loading CIFAR-10...
100%|██████████| 170M/170M [00:23<00:00, 7.14MB/s]
```

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

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

batch_size = 128

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

```
                          num_workers=2, pin_memory=True)

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

# Load ConvNeXt-Tiny model
print("\nInitializing ConvNeXt-Tiny...")
model = timm.create_model('convnext_tiny', pretrained=True, num_classes=10)
model = model.to(device)

print(f"Parameters: {sum(p.numel() for p in model.parameters()):,}")

# Training setup
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
scaler = torch.amp.GradScaler('cuda')

# Two-phase training
print("\nTraining strategy:")
print("  Phase 1: Train head only (5 epochs)")
print("  Phase 2: Fine-tune all layers (20 epochs)")

# Phase 1: Freeze backbone
for name, param in model.named_parameters():
    if 'head' in name:
        param.requires_grad = True
    else:
        param.requires_grad = False

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

num_epochs_phase1 = 5
num_epochs_phase2 = 20

scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs_phase1)

history = {
    'train_loss': [],
    'train_acc': [],
    'val_loss': [],
    'val_acc': []
}
```

```
Train: 45000 | Val: 5000 | Test: 5000

Initializing ConvNeXt-Tiny...
/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 resta
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%                                                  114M/114M  [00:02<00:00,  93.2MB/s]
Parameters: 27,827,818

Training strategy:
  Phase 1: Train head only (5 epochs)
  Phase 2: Fine-tune all layers (20 epochs)
```

```
def train_epoch(model, loader, criterion, optimizer, scaler):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    pbar = tqdm(loader, desc='Training')
    for inputs, targets in pbar:
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()

        with torch.amp.autocast('cuda'):
            outputs = model(inputs)
            loss = criterion(outputs, targets)

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

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

        pbar.set_postfix({'loss': f'{loss.item():.3f}', 'acc': f'{100.*correct/total:.2f}%'})

    return running_loss / len(loader), correct / total

@torch.no_grad()
def evaluate(model, loader, criterion):
    model.eval()
    running_loss = 0.0
```

```
    correct = 0
    total = 0

    for inputs, targets in tqdm(loader, desc='Validation'):
        inputs, targets = inputs.to(device), targets.to(device)

        with torch.amp.autocast('cuda'):
            outputs = model(inputs)
            loss = criterion(outputs, targets)

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

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

```
# Phase 1 Training
print("\n" + "="*70)
print("Phase 1: Training head")
print("="*70)

best_val_acc = 0.0

for epoch in range(num_epochs_phase1):
    print(f"\nEpoch {epoch+1}/{num_epochs_phase1}")

    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer, scaler)
    val_loss, val_acc = evaluate(model, val_loader, criterion)
    scheduler.step()

    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)

    print(f"Train: {train_acc*100:.2f}% | Val: {val_acc*100:.2f}%")

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save({
            'model_state_dict': model.state_dict(),
            'val_acc': val_acc
        }, f'{save_dir}/level4_convnext_best.pth')
        print(f"✓ Saved! Best: {val_acc*100:.2f}%")
```

```
======================================================================
Phase 1: Training head
======================================================================
```

```
Epoch 1/5
Training: 100%|██████████| 352/352 [01:40<00:00,  3.52it/s, loss=0.771, acc=86.16%]
Validation: 100%|██████████| 40/40 [00:15<00:00,  2.63it/s]
Train: 86.16% | Val: 88.82%
✓ Saved! Best: 88.82%

Epoch 2/5
Training: 100%|██████████| 352/352 [01:25<00:00,  4.10it/s, loss=0.837, acc=88.98%]
Validation: 100%|██████████| 40/40 [00:10<00:00,  3.99it/s]
Train: 88.98% | Val: 88.44%

Epoch 3/5
Training: 100%|██████████| 352/352 [01:24<00:00,  4.17it/s, loss=0.783, acc=89.71%]
Validation: 100%|██████████| 40/40 [00:08<00:00,  4.50it/s]
Train: 89.71% | Val: 89.56%
✓ Saved! Best: 89.56%

Epoch 4/5
Training: 100%|██████████| 352/352 [01:26<00:00,  4.06it/s, loss=0.750, acc=90.44%]
Validation: 100%|██████████| 40/40 [00:10<00:00,  3.99it/s]
Train: 90.44% | Val: 89.70%
✓ Saved! Best: 89.70%

Epoch 5/5
Training: 100%|██████████| 352/352 [01:25<00:00,  4.14it/s, loss=0.738, acc=90.68%]
Validation: 100%|██████████| 40/40 [00:08<00:00,  4.49it/s]Train: 90.68% | Val: 89.52%
```

```python
# Phase 2: Fine-tune
print("\n" + "="*70)
print("Phase 2: Fine-tuning entire model")
print("="*70)

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

optimizer = optim.AdamW(model.parameters(), lr=5e-5, weight_decay=0.01)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs_phase2)

patience = 7
patience_counter = 0

for epoch in range(num_epochs_phase2):
    print(f"\nEpoch {num_epochs_phase1 + epoch + 1}/{num_epochs_phase1 + num_epochs_phase2}")

    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer, scaler)
    val_loss, val_acc = evaluate(model, val_loader, criterion)
    scheduler.step()

    history['train_loss'].append(train_loss)
    history['train_acc'].append(train_acc)
    history['val_loss'].append(val_loss)
    history['val_acc'].append(val_acc)
```

```
        print(f"Train: {train_acc*100:.2f}% | Val: {val_acc*100:.2f}%")

        if val_acc > best_val_acc:
            best_val_acc = val_acc
            patience_counter = 0
            torch.save({
                'model_state_dict': model.state_dict(),
                'val_acc': val_acc
            }, f'{save_dir}/level4_convnext_best.pth')
            print(f"✓ Saved! Best: {val_acc*100:.2f}%")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"\nEarly stopping!")
                break

    # Evaluate ConvNeXt-Tiny
    print("\nEvaluating ConvNeXt-Tiny on test set...")
    checkpoint = torch.load(f'{save_dir}/level4_convnext_best.pth')
    model.load_state_dict(checkpoint['model_state_dict'])

    test_loss, convnext_acc = evaluate(model, test_loader, criterion)

    print(f"\nConvNeXt-Tiny Test Accuracy: {convnext_acc*100:.2f}%")
```

```
Epoch 21/25
Training: 100%|██████████| 352/352 [01:37<00:00,  3.60it/s, loss=0.509, acc=99.62%]
Validation: 100%|██████████| 40/40 [00:08<00:00,  4.47it/s]
Train: 99.62% | Val: 97.92%
✓ Saved! Best: 97.92%

Epoch 22/25
Training: 100%|██████████| 352/352 [01:38<00:00,  3.58it/s, loss=0.502, acc=99.57%]
Validation: 100%|██████████| 40/40 [00:09<00:00,  4.05it/s]
Train: 99.57% | Val: 97.82%

Epoch 23/25
Training: 100%|██████████| 352/352 [01:36<00:00,  3.63it/s, loss=0.503, acc=99.67%]
Validation: 100%|██████████| 40/40 [00:09<00:00,  4.28it/s]
Train: 99.67% | Val: 98.08%
✓ Saved! Best: 98.08%

Epoch 24/25
Training: 100%|██████████| 352/352 [01:37<00:00,  3.61it/s, loss=0.501, acc=99.65%]
Validation: 100%|██████████| 40/40 [00:09<00:00,  4.16it/s]
Train: 99.65% | Val: 97.70%

Epoch 25/25
Training: 100%|██████████| 352/352 [01:36<00:00,  3.65it/s, loss=0.501, acc=99.66%]
Validation: 100%|██████████| 40/40 [00:09<00:00,  4.36it/s]
Train: 99.66% | Val: 97.98%

Evaluating ConvNeXt-Tiny on test set...
Validation: 100%|██████████| 40/40 [00:04<00:00,  9.16it/s]
ConvNeXt-Tiny Test Accuracy: 98.48%
```

```
# ============================================================================
# PART 2: ENSEMBLE - Combine All Three Models
# ============================================================================

print("\n" + "="*70)
print("PART 2: ENSEMBLE LEARNING")
print("="*70)
print("\nNow combining predictions from three different architectures:")
print("  1. ResNet50         (from Level 1)")
print("  2. EfficientNet-B2  (from Level 3)")
print("  3. ConvNeXt-Tiny    (just trained)")
print("\nUsing majority voting for final predictions")
print("="*70)

# Load all three models
print("\nLoading models...")

# Model 1: ResNet50
print("  Loading ResNet50...")
resnet = timm.create_model('resnet50', pretrained=False, num_classes=10)
resnet_checkpoint = torch.load(f'{save_dir}/level1_best.pth')
resnet.load_state_dict(resnet_checkpoint['model'])
```

```
resnet = resnet.to(device)
resnet.eval()

# Model 2: EfficientNet-B2
print("  Loading EfficientNet-B2...")
effnet = timm.create_model('efficientnet_b2', pretrained=False, num_classes=10)
effnet_checkpoint = torch.load(f'{save_dir}/level3_best_model.pth')
effnet.load_state_dict(effnet_checkpoint['model_state_dict'])
effnet = effnet.to(device)
effnet.eval()

# Model 3: ConvNeXt-Tiny (already loaded)
print("  ConvNeXt-Tiny already loaded")
model.eval()

print("\n✓ All models loaded successfully!")

# Get individual model predictions
def get_predictions(model, loader, model_name):
    """Get predictions from a single model"""
    all_preds = []
    all_targets = []
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, targets in tqdm(loader, desc=f'Predicting ({model_name})'):
            inputs = inputs.to(device)

            with torch.amp.autocast('cuda'):
                outputs = model(inputs)
                _, preds = outputs.max(1)

            all_preds.extend(preds.cpu().numpy())
            all_targets.extend(targets.numpy())

            correct += preds.cpu().eq(targets).sum().item()
            total += targets.size(0)

    accuracy = correct / total
    return np.array(all_preds), np.array(all_targets), accuracy

# Get predictions from all models
print("\n" + "="*70)
print("Getting predictions from each model...")
print("="*70)

resnet_preds, targets, resnet_acc = get_predictions(resnet, test_loader, "ResNet50")
effnet_preds, _, effnet_acc = get_predictions(effnet, test_loader, "EfficientNet-B2")
convnext_preds, _, convnext_acc = get_predictions(model, test_loader, "ConvNeXt-Tiny")
```

```
print(f"\nIndividual Model Accuracies:")
print(f"  ResNet50:        {resnet_acc*100:.2f}%")
print(f"  EfficientNet-B2: {effnet_acc*100:.2f}%")
print(f"  ConvNeXt-Tiny:   {convnext_acc*100:.2f}%")

# Ensemble using majority voting
print("\n" + "="*70)
print("Creating ensemble predictions (majority voting)...")
print("="*70)

ensemble_preds = []

for i in range(len(targets)):
    # Get predictions from all three models for this sample
    votes = [resnet_preds[i], effnet_preds[i], convnext_preds[i]]

    # Majority voting
    vote_counts = Counter(votes)
    majority_vote = vote_counts.most_common(1)[0][0]
    ensemble_preds.append(majority_vote)

ensemble_preds = np.array(ensemble_preds)
ensemble_acc = (ensemble_preds == targets).sum() / len(targets)

print(f"\n✓ Ensemble Accuracy: {ensemble_acc*100:.2f}%")
print(f"  Improvement over best single model: +{(ensemble_acc - max(resnet_acc, effnet_acc, convnext_acc))*100:.2f}%")
```

```
======================================================================
PART 2: ENSEMBLE LEARNING
======================================================================

Now combining predictions from three different architectures:
  1. ResNet50         (from Level 1)
  2. EfficientNet-B2  (from Level 3)
  3. ConvNeXt-Tiny    (just trained)

Using majority voting for final predictions
======================================================================

Loading models...
  Loading ResNet50...
  Loading EfficientNet-B2...
  ConvNeXt-Tiny already loaded

✓ All models loaded successfully!

======================================================================
Getting predictions from each model...
======================================================================
Predicting (ResNet50): 100%|██████████| 40/40 [00:04<00:00,  8.56it/s]
Predicting (EfficientNet-B2): 100%|██████████| 40/40 [00:08<00:00,  4.46it/s]
Predicting (ConvNeXt-Tiny): 100%|██████████| 40/40 [00:04<00:00,  8.55it/s]
```

```
Individual Model Accuracies:
  ResNet50:        96.34%
  EfficientNet-B2: 94.82%
  ConvNeXt-Tiny:   98.48%

======================================================================
Creating ensemble predictions (majority voting)...
======================================================================

✓ Ensemble Accuracy: 97.58%
  Improvement over best single model: +-0.90%
```

```
print("\n" + "="*70)
print("LEVEL 4 FINAL RESULTS")
print("="*70)

print("\nINDIVIDUAL MODEL PERFORMANCE:")
print(f"  ResNet50 (Level 1):         {resnet_acc*100:.2f}%")
print(f"  EfficientNet-B2 (Level 3):  {effnet_acc*100:.2f}%")
print(f"  ConvNeXt-Tiny (Level 4):    {convnext_acc*100:.2f}%")

print(f"\nENSEMBLE PERFORMANCE:")
print(f"  Majority Voting:            {ensemble_acc*100:.2f}%")

print(f"\nPROGRESSION ACROSS LEVELS:")
print(f"  Level 1 (ResNet50):         96.34%")
print(f"  Level 2 (ResNet50 + Aug):   96.88%")
print(f"  Level 3 (EfficientNet-B2):  96.20%")
print(f"  Level 4 (Ensemble):         {ensemble_acc*100:.2f}%")

print(f"\nTARGET: 93-97%+")
print(f"STATUS: {'✓ PASSED' if ensemble_acc >= 0.93 else '✗ NEEDS WORK'}")
print("="*70)

# Per-class accuracy for ensemble
print("\n" + "="*70)
print("Ensemble Per-Class Accuracy")
print("="*70)

class_correct = [0] * 10
class_total = [0] * 10

for i in range(len(targets)):
    label = targets[i]
    class_correct[label] += (ensemble_preds[i] == targets[i])
    class_total[label] += 1

print("\nPer-class results:")
for i in range(10):
    acc = 100 * class_correct[i] / class_total[i]
```

```
    print(f"  {class_names[i]:12s}: {acc:5.2f}%")

# Visualizations
print("\nCreating visualizations...")

# 1. Model comparison bar chart
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

models = ['ResNet50\n(Level 1)', 'EfficientNet-B2\n(Level 3)',
          'ConvNeXt-Tiny\n(Level 4)', 'Ensemble\n(Voting)']
accuracies = [resnet_acc*100, effnet_acc*100, convnext_acc*100, ensemble_acc*100]
colors = ['skyblue', 'lightcoral', 'lightgreen', 'gold']

bars = axes[0].bar(models, accuracies, color=colors, edgecolor='black', linewidth=1.5)
axes[0].axhline(y=93, color='red', linestyle='--', label='Level 4 Target (93%)', linewidth=2)
axes[0].set_ylabel('Accuracy (%)', fontsize=12)
axes[0].set_title('Model Comparison - Level 4', fontsize=14, fontweight='bold')
axes[0].set_ylim(90, 100)
axes[0].legend()
axes[0].grid(True, alpha=0.3, axis='y')

# Add value labels on bars
for bar, acc in zip(bars, accuracies):
    height = bar.get_height()
    axes[0].text(bar.get_x() + bar.get_width()/2., height,
                f'{acc:.2f}%', ha='center', va='bottom', fontsize=11, fontweight='bold')

# 2. Per-class accuracy
class_accs = [100 * class_correct[i] / class_total[i] for i in range(10)]
colors_per_class = ['green' if acc >= 95 else 'orange' if acc >= 90 else 'red'
                    for acc in class_accs]

axes[1].bar(class_names, class_accs, color=colors_per_class, alpha=0.7, edgecolor='black')
axes[1].axhline(y=93, color='red', linestyle='--', label='Target (93%)', linewidth=2)
axes[1].set_ylabel('Accuracy (%)', fontsize=12)
axes[1].set_title('Ensemble Per-Class Accuracy', fontsize=14, fontweight='bold')
axes[1].set_xticklabels(class_names, rotation=45, ha='right')
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(f'{save_dir}/level4_ensemble_results.png', dpi=150, bbox_inches='tight')
plt.show()

print("✓ Visualizations saved")
```

```
======================================================================
LEVEL 4 FINAL RESULTS
======================================================================

INDIVIDUAL MODEL PERFORMANCE:
  ResNet50 (Level 1):         96.34%
  EfficientNet-B2 (Level 3):  94.82%
  ConvNeXt-Tiny (Level 4):    98.48%

ENSEMBLE PERFORMANCE:
  Majority Voting:            97.58%

PROGRESSION ACROSS LEVELS:
  Level 1 (ResNet50):         96.34%
```

```
# Save summary
summary = f"""
LEVEL 4 - COMPLETION SUMMARY
{'='*70}

Technique: Ensemble Learning (Majority Voting)
Models Used: 3 different architectures

INDIVIDUAL MODEL PERFORMANCE:
  1. ResNet50 (Level 1):        {resnet_acc*100:.2f}%
  2. EfficientNet-B2 (Level 3): {effnet_acc*100:.2f}%
  3. ConvNeXt-Tiny (Level 4):   {convnext_acc*100:.2f}%

ENSEMBLE PERFORMANCE:
  Majority Voting:              {ensemble_acc*100:.2f}%

IMPROVEMENT:
  vs Best Single Model:         +{(ensemble_acc - max(resnet_acc, effnet_acc, convnext_acc))*100:.2f}%

FULL PROGRESSION:
  Level 1 (ResNet50 baseline):     96.34%
  Level 2 (ResNet50 + augment):    96.88%
  Level 3 (EfficientNet-B2):       96.20%
  Level 4 (3-Model Ensemble):      {ensemble_acc*100:.2f}%

TARGET: 93-97%+
STATUS: {'PASSED ✓' if ensemble_acc >= 0.93 else 'NEEDS IMPROVEMENT'}

WHY ENSEMBLE LEARNING?
- Combines diverse architectures (ResNet, EfficientNet, ConvNeXt)
- Reduces individual model biases
- More robust predictions through voting
- Industry-standard technique for competitions

KEY INSIGHTS:
- Different architectures make different mistakes
```

```
- Ensemble corrects individual model errors
- Majority voting is simple but effective
- Model diversity is key to ensemble success

TECHNIQUES DEMONSTRATED ACROSS ALL LEVELS:
✓ Level 1: Transfer learning (ResNet50)
✓ Level 2: Advanced augmentation (Mixup, CutMix, TTA)
✓ Level 3: Custom architecture exploration (EfficientNet-B2)
✓ Level 4: Ensemble learning (3-model voting)

FILES SAVED:
- {save_dir}/level4_convnext_best.pth
- {save_dir}/level4_ensemble_results.png
- {save_dir}/level4_summary.txt

{'='*70}
"""

with open(f'{save_dir}/level4_summary.txt', 'w') as f:
    f.write(summary)

print(summary)

print("\n" + "="*70)
print("✓✓✓ LEVEL 4 COMPLETE! ✓✓✓")
print("="*70)
print("\nALL TECHNICAL LEVELS FINISHED!")
print("\nNext steps:")
print("  1. Take screenshots of all results")
print("  2. Prepare comprehensive documentation (10-page report)")
print("  3. Create final submission PDF with all Colab links")
print("  4. Submit before deadline!")
print("="*70)
```

```
LEVEL 4 - COMPLETION SUMMARY
======================================================================

Technique: Ensemble Learning (Majority Voting)
Models Used: 3 different architectures

INDIVIDUAL MODEL PERFORMANCE:
  1. ResNet50 (Level 1):        96.34%
  2. EfficientNet-B2 (Level 3): 94.82%
  3. ConvNeXt-Tiny (Level 4):   98.48%

ENSEMBLE PERFORMANCE:
  Majority Voting:              97.58%

IMPROVEMENT:
  vs Best Single Model:         +-0.90%
```

```
FULL PROGRESSION:
  Level 1 (ResNet50 baseline):     96.34%
  Level 2 (ResNet50 + augment):    96.88%
  Level 3 (EfficientNet-B2):       96.20%
  Level 4 (3-Model Ensemble):      97.58%

TARGET: 93-97%+
STATUS: PASSED ✓

WHY ENSEMBLE LEARNING?
- Combines diverse architectures (ResNet, EfficientNet, ConvNeXt)
- Reduces individual model biases
- More robust predictions through voting
- Industry-standard technique for competitions

KEY INSIGHTS:
- Different architectures make different mistakes
- Ensemble corrects individual model errors
- Majority voting is simple but effective
- Model diversity is key to ensemble success

TECHNIQUES DEMONSTRATED ACROSS ALL LEVELS:
✓ Level 1: Transfer learning (ResNet50)
✓ Level 2: Advanced augmentation (Mixup, CutMix, TTA)
✓ Level 3: Custom architecture exploration (EfficientNet-B2)
✓ Level 4: Ensemble learning (3-model voting)

FILES SAVED:
- /content/drive/MyDrive/CIFAR10_Challenge/level4_convnext_best.pth
- /content/drive/MyDrive/CIFAR10_Challenge/level4_ensemble_results.png
- /content/drive/MyDrive/CIFAR10_Challenge/level4_summary.txt

======================================================================


======================================================================
✓✓✓ LEVEL 4 COMPLETE! ✓✓✓
======================================================================
```