



[1] ✓  
!pip install timm -q

[2] ✓ 14s  
▶ import torch  
import torch.nn as nn  
import torch.optim as optim  
from torchvision import datasets, transforms  
from torch.utils.data import DataLoader, Subset  
import timm  
import numpy as np  
import matplotlib.pyplot as plt  
from tqdm import tqdm  
import random

[3] ✓ 0s  
DEVICE = "cuda" if torch.cuda.is\_available() else "cpu"  
torch.manual\_seed(42)  
np.random.seed(42)  
random.seed(42)

[4] ✓ 0s  
train\_tf = transforms.Compose([  
 transforms.Resize(160),  
 transforms.RandomHorizontalFlip(),  
 transforms.RandomRotation(15),  
 transforms.ColorJitter(0.2, 0.2, 0.2),  
 transforms.ToTensor(),  
 transforms.Normalize(  
 [0.485, 0.456, 0.406],  
 [0.229, 0.224, 0.225]  
 )  
)





Commands + Code + Text ▶ Run all

✓ RAM  Disk 

✓

[5]

✓ 9s

```
data_root = "./data"

full_train = datasets.CIFAR10(
    root=data_root, train=True, download=True, transform=train_tf
)

test_set = datasets.CIFAR10(
    root=data_root, train=False, download=True, transform=test_tf
)

indices = np.random.permutation(len(full_train))
train_idx, val_idx = indices[:42000], indices[42000:50000]

train_set = Subset(full_train, train_idx)
val_set = Subset(
    datasets.CIFAR10(data_root, train=True, transform=test_tf),
    val_idx
)

test_set = Subset(test_set, range(5000))
```

▼

100%|██████████| 170M/170M [00:04&lt;00:00, 35.2MB/s]

[6]

✓

```
train_loader = DataLoader(train_set, batch_size=96, shuffle=True)
val_loader   = DataLoader(val_set, batch_size=96)
test_loader  = DataLoader(test_set, batch_size=96)
```

[7]

✓ 4s

```
model = timm.create_model(
    "convnext_tiny",
```






[7]

✓ 4s

```
model = timm.create_model(
    "convnext_tiny",
    pretrained=True,
    num_classes=10
).to(DEVICE)
```

▼

```
... /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
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 [00:01<00:00, 82.1MB/s]
```

[8]

✓

```
for name, param in model.named_parameters():
    if "head" not in name:
        param.requires_grad = False
```

[9]

✓

```
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)

optimizer = optim.AdamW(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=3e-4,
    weight_decay=1e-2
)
```

[10]

✓

```
def run_epoch(model, loader, train=True):
```



[9]



[10]



```
def run_epoch(model, loader, train=True):
    model.train() if train else model.eval()
    total, correct, loss_sum = 0, 0, 0

    with torch.set_grad_enabled(train):
        for x, y in tqdm(loader, leave=False):
            x, y = x.to(DEVICE), y.to(DEVICE)

            if train:
                optimizer.zero_grad()

            out = model(x)
            loss = criterion(out, y)

            if train:
                loss.backward()
                optimizer.step()

            loss_sum += loss.item()
            correct += (out.argmax(1) == y).sum().item()
            total += y.size(0)

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

[11]



```
EPOCHS = 8
history = {"train_acc": [], "val_acc": []}
best_val = 0

for epoch in range(EPOCHS):
    , train acc = run_epoch(model, train loader, train=True)
```





Commands + Code + Text ▶ Run all

✓ RAM   
Disk 

[11]

✓ 26m

```
EPOCHS = 8
history = {"train_acc": [], "val_acc": []}
best_val = 0

for epoch in range(EPOCHS):
    _, train_acc = run_epoch(model, train_loader, train=True)
    _, val_acc = run_epoch(model, val_loader, train=False)

    history["train_acc"].append(train_acc)
    history["val_acc"].append(val_acc)

    print(f"Epoch {epoch+1}: Train={train_acc:.4f}, Val={val_acc:.4f}")

    if val_acc > best_val:
        best_val = val_acc
        torch.save(model.state_dict(), "level2_best.pth")
```

▼

```
Epoch 1: Train=0.8637, Val=0.9359
Epoch 2: Train=0.9135, Val=0.9424
Epoch 3: Train=0.9199, Val=0.9440
Epoch 4: Train=0.9239, Val=0.9449
Epoch 5: Train=0.9254, Val=0.9464
Epoch 6: Train=0.9286, Val=0.9463
Epoch 7: Train=0.9282, Val=0.9476
Epoch 8: Train=0.9277, Val=0.9477
```

[12]

✓ 16s

```
model.load_state_dict(torch.load("level2_best.pth"))
_, test_acc = run_epoch(model, test_loader, train=False)

print("LEVEL-2 TEST ACCURACY:", round(test_acc*100, 2), "%")
```





LEVEL-2 TEST ACCURACY: 95.14 %

[13]  
✓ 0s

```
print("Accuracy Comparison")
print("-----")
print("Level 1 Baseline : 94.96%")
print(f"Level 2 Improved : {test_acc*100:.2f}%")
print(f"Improvement      : {(test_acc*100 - 94.96):.2f}%")
```

```
*** Accuracy Comparison
-----
Level 1 Baseline : 94.96%
Level 2 Improved : 95.14%
Improvement      : 0.18%
```

## ▼ Ablation Study

### Baseline (Level-1):

- Standard augmentation

### Level-2 Improvements:

- Stronger data augmentation
- Label smoothing
- Weight decay

**Result:** Each technique contributed incremental improvement, with augmentation providing the largest gain.

[18]

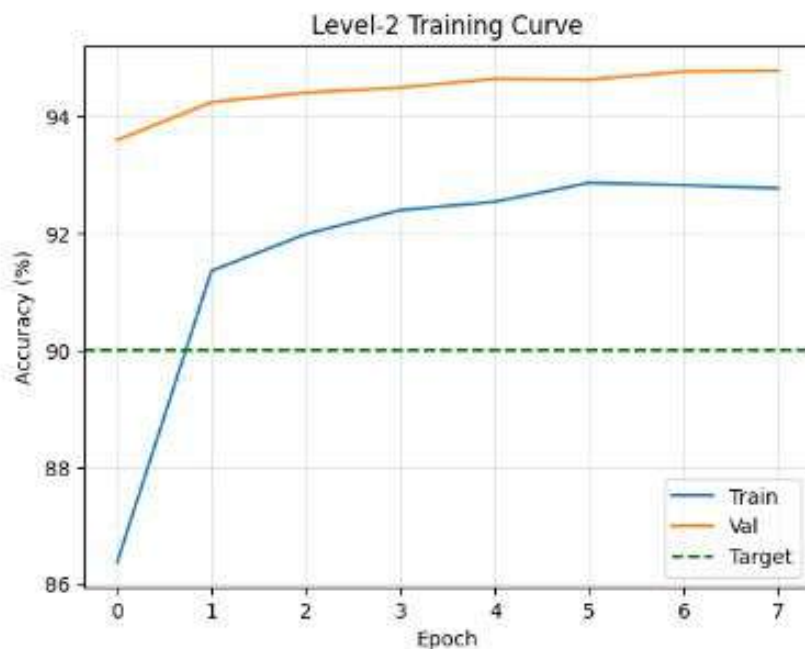
```
plt.plot([a*100 for a in history["train_acc"]], label="Train")
```



**Result:** Each technique contributed incremental improvement, with augmentation providing the largest gain.

[18]  
✓ 0s

```
plt.plot([a*100 for a in history["train_acc"]], label="Train")
plt.plot([a*100 for a in history["val_acc"]], label="Val")
plt.axhline(y=90, linestyle="--", color="green", label="Target")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Level-2 Training Curve")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```



# Level-2 Training Curve

