

```

# CIFAR10 training
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
import torchvision
import torchvision.transforms as T
from torchvision import models
from tqdm import tqdm

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", DEVICE)

BATCH_SIZE = 256
LR = 1e-3
MAX_EPOCHS = 60
TARGET_TRAIN_ACC = 0.90

# ----- Fast Augmentation -----
train_tf = T.Compose([
    T.RandomCrop(32, padding=4),
    T.RandomHorizontalFlip(),
    T.ToTensor(),
    T.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5)),
])

test_tf = T.Compose([
    T.ToTensor(),
    T.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5)),
])

# ----- Load dataset -----
train_full = torchvision.datasets.CIFAR10("data", train=True,
download=True, transform=train_tf)
test_set = torchvision.datasets.CIFAR10("data", train=False,
download=True, transform=test_tf)

train_set, val_set = random_split(train_full, [45000, 5000])

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

# ----- Model: MobileNetV3 (FAST) -----
model = models.mobilenet_v3_small(weights=None)
model.classifier[3] = nn.Linear(model.classifier[3].in_features, 10)
model = model.to(DEVICE, memory_format=torch.channels_last)

```

```

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,
T_max=MAX_EPOCHS)

scaler = torch.cuda.amp.GradScaler()

# ----- Train -----
def evaluate(loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for X, y in loader:
            X = X.to(DEVICE, memory_format=torch.channels_last)
            y = y.to(DEVICE)
            out = model(X)
            pred = out.argmax(1)
            total += y.size(0)
            correct += (pred == y).sum().item()
    return correct/total

for epoch in range(1, MAX_EPOCHS+1):
    model.train()
    correct = 0
    total = 0

    for X, y in tqdm(train_loader, desc=f"Epoch {epoch}"):
        X = X.to(DEVICE, memory_format=torch.channels_last)
        y = y.to(DEVICE)

        optimizer.zero_grad()

        with torch.cuda.amp.autocast():
            out = model(X)
            loss = criterion(out, y)

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

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

    train_acc = correct / total
    val_acc = evaluate(val_loader)
    scheduler.step()

```

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

```
if train_acc >= TARGET_TRAIN_ACC:  
    print("Reached 90% training accuracy. Stopping.")  
    break
```

```
# ----- Test -----
```

```
test_acc = evaluate(test_loader)  
print(f"Final Test Accuracy: {test_acc*100:.2f}%")
```

Device: cuda

100%|██████████| 170M/170M [00:04<00:00, 37.3MB/s]

/tmp/ipython-input-2470047190.py:52: FutureWarning:
`torch.cuda.amp.GradScaler(args...)` is deprecated. Please use
`torch.amp.GradScaler('cuda', args...)` instead.

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

Epoch 1: 0%|██████████| 0/176 [00:00<?, ?it/s]/tmp/ipython-input-2470047190.py:80: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.

```
with torch.cuda.amp.autocast():
```

Epoch 1: 100%|██████████| 176/176 [00:48<00:00, 3.64it/s]

Epoch 1 | Train Acc: 27.78% | Val Acc: 10.38%

Epoch 2: 100%|██████████| 176/176 [00:20<00:00, 8.42it/s]

Epoch 2 | Train Acc: 38.87% | Val Acc: 10.38%

Epoch 3: 100%|██████████| 176/176 [00:19<00:00, 8.88it/s]

Epoch 3 | Train Acc: 43.94% | Val Acc: 21.46%

Epoch 4: 100%|██████████| 176/176 [00:19<00:00, 9.15it/s]

Epoch 4 | Train Acc: 46.69% | Val Acc: 46.58%

Epoch 5: 100%|██████████| 176/176 [00:20<00:00, 8.65it/s]

Epoch 5 | Train Acc: 49.00% | Val Acc: 50.70%

Epoch 6: 100%|██████████| 176/176 [00:19<00:00, 8.84it/s]

Epoch 6 | Train Acc: 51.48% | Val Acc: 52.28%

Epoch 7: 100%|██████████| 176/176 [00:18<00:00, 9.29it/s]

Epoch 7 | Train Acc: 52.63% | Val Acc: 53.28%

Epoch 8: 100%|██████████| 176/176 [00:20<00:00, 8.76it/s]

Epoch 8 | Train Acc: 54.29% | Val Acc: 53.94%

Epoch 9: 100%|██████████| 176/176 [00:19<00:00, 8.82it/s]
Epoch 9 | Train Acc: 55.34% | Val Acc: 57.22%
Epoch 10: 100%|██████████| 176/176 [00:19<00:00, 9.08it/s]
Epoch 10 | Train Acc: 56.87% | Val Acc: 58.24%
Epoch 11: 100%|██████████| 176/176 [00:19<00:00, 8.91it/s]
Epoch 11 | Train Acc: 57.74% | Val Acc: 57.58%
Epoch 12: 100%|██████████| 176/176 [00:20<00:00, 8.80it/s]
Epoch 12 | Train Acc: 58.62% | Val Acc: 58.32%
Epoch 13: 100%|██████████| 176/176 [00:19<00:00, 9.21it/s]
Epoch 13 | Train Acc: 59.35% | Val Acc: 59.22%
Epoch 14: 100%|██████████| 176/176 [00:19<00:00, 8.85it/s]
Epoch 14 | Train Acc: 60.14% | Val Acc: 60.60%
Epoch 15: 100%|██████████| 176/176 [00:20<00:00, 8.72it/s]
Epoch 15 | Train Acc: 60.70% | Val Acc: 60.02%
Epoch 16: 100%|██████████| 176/176 [00:19<00:00, 9.21it/s]
Epoch 16 | Train Acc: 61.76% | Val Acc: 60.94%
Epoch 17: 100%|██████████| 176/176 [00:19<00:00, 8.99it/s]
Epoch 17 | Train Acc: 62.25% | Val Acc: 62.44%
Epoch 18: 100%|██████████| 176/176 [00:19<00:00, 8.83it/s]
Epoch 18 | Train Acc: 63.01% | Val Acc: 61.56%
Epoch 19: 100%|██████████| 176/176 [00:19<00:00, 9.24it/s]
Epoch 19 | Train Acc: 63.50% | Val Acc: 62.86%
Epoch 20: 100%|██████████| 176/176 [00:19<00:00, 9.05it/s]
Epoch 20 | Train Acc: 64.28% | Val Acc: 63.34%
Epoch 21: 100%|██████████| 176/176 [00:19<00:00, 8.85it/s]
Epoch 21 | Train Acc: 64.75% | Val Acc: 64.04%
Epoch 22: 76%|██████████| 133/176 [00:14<00:04, 8.97it/s]

```

-----
-----
KeyboardInterrupt                                Traceback (most recent call
last)
/tmp/ipython-input-2470047190.py in <cell line: 0>()
    72     total = 0
    73
--> 74     for X, y in tqdm(train_loader, desc=f"Epoch {epoch}"):
    75         X = X.to(DEVICE, memory_format=torch.channels_last)
    76         y = y.to(DEVICE)

/usr/local/lib/python3.12/dist-packages/tqdm/std.py in __iter__(self)
   1179
   1180     try:
-> 1181         for obj in iterable:
   1182             yield obj
   1183             # Update and possibly print the progressbar.

/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py
in __next__(self)
    732         #
TODO(https://github.com/pytorch/pytorch/issues/76750)
    733         self._reset() # type: ignore[call-arg]
--> 734         data = self._next_data()
    735         self._num_yielded += 1
    736         if (

/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py
in _next_data(self)
   1490
   1491         assert not self._shutdown and
self._tasks_outstanding > 0
-> 1492         idx, data = self._get_data()
   1493         self._tasks_outstanding -= 1
   1494         if self._dataset_kind == _DatasetKind.Iterable:

/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py
in _get_data(self)
   1442         elif self._pin_memory:
   1443             while self._pin_memory_thread.is_alive():
-> 1444                 success, data = self._try_get_data()
   1445                 if success:
   1446                     return data

/usr/local/lib/python3.12/dist-packages/torch/utils/data/dataloader.py
in _try_get_data(self, timeout)
   1283         # (bool: whether successfully get data, any: data if
successful else None)
   1284         try:
-> 1285         data = self._data_queue.get(timeout=timeout)

```

```

1286         return (True, data)
1287     except Exception as e:

/usr/lib/python3.12/queue.py in get(self, block, timeout)
178         if remaining <= 0.0:
179             raise Empty
--> 180         self.not_empty.wait(remaining)
181         item = self._get()
182         self.not_full.notify()

/usr/lib/python3.12/threading.py in wait(self, timeout)
357         else:
358             if timeout > 0:
--> 359                 gotit = waiter.acquire(True, timeout)
360             else:
361                 gotit = waiter.acquire(False)

```

KeyboardInterrupt:

```

# ----- Test -----
test_acc = evaluate(test_loader)
print(f"Final Test Accuracy: {test_acc*100:.2f}%")

Final Test Accuracy: 65.07%

```