CNN-pytorch-tensorflow-transfer-learning-google-colab

April 29, 2025

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torch.multiprocessing

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Transform
transform = transforms.Compose(
    [transforms.Resize(32), transforms.ToTensor(), transforms.Normalize((0.5,),␣
 ↪(0.5,))]
)

# Dataset
trainset = torchvision.datasets.CIFAR10(
    root="./data", train=True, download=True, transform=transform
)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=64, shuffle=True, num_workers=2
)

testset = torchvision.datasets.CIFAR10(
    root="./data", train=False, download=True, transform=transform
)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=64, shuffle=False, num_workers=2
)


# Model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.net = nn.Sequential(
```

```
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((4, 4)),
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(),
            nn.Linear(256, 10),
        )

    def forward(self, x):
        return self.net(x)


# Loss and Optimizer
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training
EPOCHS = 30

if __name__ == '__main__':
    torch.multiprocessing.freeze_support()

    for epoch in range(EPOCHS):
        running_loss = 0.0
        for inputs, labels in trainloader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        print(f"Epoch {epoch+1}/{EPOCHS} - Loss: {running_loss/len(trainloader):
 ↪.4f}")

    print("Finished Training")
```

```python
    # Testing
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f"Accuracy on the 10000 test images: {100 * correct / total:.2f}%")
```

```
Epoch 1/30 - Loss: 1.3918
Epoch 2/30 - Loss: 0.9711
Epoch 3/30 - Loss: 0.7830
Epoch 4/30 - Loss: 0.6590
Epoch 5/30 - Loss: 0.5599
Epoch 6/30 - Loss: 0.4695
Epoch 7/30 - Loss: 0.3897
Epoch 8/30 - Loss: 0.3185
Epoch 9/30 - Loss: 0.2469
Epoch 10/30 - Loss: 0.1960
Epoch 11/30 - Loss: 0.1528
Epoch 12/30 - Loss: 0.1283
Epoch 13/30 - Loss: 0.1026
Epoch 14/30 - Loss: 0.0857
Epoch 15/30 - Loss: 0.0891
Epoch 16/30 - Loss: 0.0666
Epoch 17/30 - Loss: 0.0784
Epoch 18/30 - Loss: 0.0634
Epoch 19/30 - Loss: 0.0626
Epoch 20/30 - Loss: 0.0615
Epoch 21/30 - Loss: 0.0590
Epoch 22/30 - Loss: 0.0616
Epoch 23/30 - Loss: 0.0491
Epoch 24/30 - Loss: 0.0500
Epoch 25/30 - Loss: 0.0533
Epoch 26/30 - Loss: 0.0499
Epoch 27/30 - Loss: 0.0474
Epoch 28/30 - Loss: 0.0466
Epoch 29/30 - Loss: 0.0455
Epoch 30/30 - Loss: 0.0435
Finished Training
Accuracy on the 10000 test images: 75.90%
```

[2]:
```
import tensorflow as tf
from keras import datasets, layers, models, optimizers
import numpy as np
import os

# Set seeds for reproducibility
tf.random.set_seed(42)
np.random.seed(42)

# GPU settings
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)

# Load and preprocess CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.
 ↪load_data()

# Normalize pixel values to be between -1 and 1
train_images = (train_images / 127.5) - 1
test_images = (test_images / 127.5) - 1

# Model
def create_model():
    model = models.Sequential([
        # First convolutional block
        layers.Conv2D(32, (3, 3), padding='same', activation='relu',␣
 ↪input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),

        # Second convolutional block
        layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # Third convolutional block
        layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
        layers.GlobalAveragePooling2D(),

        # Fully connected layers
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dense(10)
    ])
```

```
    return model

# Create and compile the model
model = create_model()
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

# Display model summary
model.summary()

# Training parameters
BATCH_SIZE = 64
EPOCHS = 30

# Training
history = model.fit(
    train_images, train_labels,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=(test_images, test_labels),
    verbose=1
)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Accuracy on the 10000 test images: {test_acc * 100:.2f}%")

# Save model
model_path = "model/tensorflow_cifar10.h5"
os.makedirs(os.path.dirname(model_path), exist_ok=True)
model.save(model_path)
print(f"Model saved to {model_path}")
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071             95s
1us/step
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73,856
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 256)	33,024
dense_1 (Dense)	(None, 10)	2,570

Total params: 128,842 (503.29 KB)

Trainable params: 128,842 (503.29 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/30
782/782             10s 8ms/step -
accuracy: 0.2972 - loss: 1.8513 - val_accuracy: 0.4815 - val_loss: 1.4033
Epoch 2/30
782/782             6s 5ms/step -
accuracy: 0.4959 - loss: 1.3799 - val_accuracy: 0.5692 - val_loss: 1.2024
Epoch 3/30
782/782             3s 4ms/step -
accuracy: 0.5736 - loss: 1.1834 - val_accuracy: 0.6184 - val_loss: 1.0701
Epoch 4/30
782/782             4s 4ms/step -
accuracy: 0.6231 - loss: 1.0595 - val_accuracy: 0.6507 - val_loss: 0.9895
Epoch 5/30
782/782             4s 5ms/step -
accuracy: 0.6526 - loss: 0.9723 - val_accuracy: 0.6712 - val_loss: 0.9344
```

```
Epoch 6/30
782/782            5s 5ms/step -
accuracy: 0.6790 - loss: 0.9037 - val_accuracy: 0.6882 - val_loss: 0.8946
Epoch 7/30
782/782            3s 4ms/step -
accuracy: 0.6989 - loss: 0.8469 - val_accuracy: 0.6962 - val_loss: 0.8738
Epoch 8/30
782/782            4s 5ms/step -
accuracy: 0.7181 - loss: 0.7983 - val_accuracy: 0.7089 - val_loss: 0.8486
Epoch 9/30
782/782            5s 4ms/step -
accuracy: 0.7351 - loss: 0.7531 - val_accuracy: 0.7137 - val_loss: 0.8408
Epoch 10/30
782/782            6s 5ms/step -
accuracy: 0.7504 - loss: 0.7141 - val_accuracy: 0.7184 - val_loss: 0.8238
Epoch 11/30
782/782            4s 5ms/step -
accuracy: 0.7635 - loss: 0.6754 - val_accuracy: 0.7311 - val_loss: 0.7946
Epoch 12/30
782/782            5s 5ms/step -
accuracy: 0.7750 - loss: 0.6400 - val_accuracy: 0.7363 - val_loss: 0.7753
Epoch 13/30
782/782            4s 5ms/step -
accuracy: 0.7895 - loss: 0.6062 - val_accuracy: 0.7442 - val_loss: 0.7605
Epoch 14/30
782/782            5s 4ms/step -
accuracy: 0.8015 - loss: 0.5741 - val_accuracy: 0.7472 - val_loss: 0.7528
Epoch 15/30
782/782            5s 5ms/step -
accuracy: 0.8098 - loss: 0.5449 - val_accuracy: 0.7489 - val_loss: 0.7559
Epoch 16/30
782/782            5s 4ms/step -
accuracy: 0.8205 - loss: 0.5174 - val_accuracy: 0.7502 - val_loss: 0.7582
Epoch 17/30
782/782            3s 4ms/step -
accuracy: 0.8290 - loss: 0.4912 - val_accuracy: 0.7579 - val_loss: 0.7432
Epoch 18/30
782/782            6s 5ms/step -
accuracy: 0.8386 - loss: 0.4665 - val_accuracy: 0.7603 - val_loss: 0.7376
Epoch 19/30
782/782            3s 4ms/step -
accuracy: 0.8473 - loss: 0.4415 - val_accuracy: 0.7588 - val_loss: 0.7467
Epoch 20/30
782/782            3s 4ms/step -
accuracy: 0.8565 - loss: 0.4187 - val_accuracy: 0.7584 - val_loss: 0.7557
Epoch 21/30
782/782            4s 5ms/step -
accuracy: 0.8634 - loss: 0.3996 - val_accuracy: 0.7597 - val_loss: 0.7678
```

```
Epoch 22/30
782/782             5s 4ms/step -
accuracy: 0.8689 - loss: 0.3811 - val_accuracy: 0.7591 - val_loss: 0.7808
Epoch 23/30
782/782             3s 4ms/step -
accuracy: 0.8740 - loss: 0.3637 - val_accuracy: 0.7610 - val_loss: 0.7935
Epoch 24/30
782/782             4s 5ms/step -
accuracy: 0.8792 - loss: 0.3478 - val_accuracy: 0.7602 - val_loss: 0.8121
Epoch 25/30
782/782             5s 5ms/step -
accuracy: 0.8833 - loss: 0.3343 - val_accuracy: 0.7606 - val_loss: 0.8265
Epoch 26/30
782/782             3s 4ms/step -
accuracy: 0.8865 - loss: 0.3250 - val_accuracy: 0.7574 - val_loss: 0.8495
Epoch 27/30
782/782             4s 5ms/step -
accuracy: 0.8858 - loss: 0.3213 - val_accuracy: 0.7592 - val_loss: 0.8386
Epoch 28/30
782/782             3s 4ms/step -
accuracy: 0.8909 - loss: 0.3134 - val_accuracy: 0.7570 - val_loss: 0.8656
Epoch 29/30
782/782             6s 5ms/step -
accuracy: 0.8952 - loss: 0.3012 - val_accuracy: 0.7578 - val_loss: 0.8672
Epoch 30/30
782/782             4s 5ms/step -
accuracy: 0.8998 - loss: 0.2867 - val_accuracy: 0.7620 - val_loss: 0.8798
313/313 - 1s - 4ms/step - accuracy: 0.7620 - loss: 0.8798

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

Accuracy on the 10000 test images: 76.20%
Model saved to model/tensorflow_cifar10.h5
```

[3]:
```python
#  1. Imports & Device
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader

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

```
#  2. Data transforms & loaders
# CIFAR-10 images are 32×32; VGG expects 224×224
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]

train_transform = transforms.Compose(
    [
        transforms.Resize(256),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ]
)

val_transform = transforms.Compose(
    [
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),
    ]
)

train_ds = datasets.CIFAR10(
    root="./data", train=True, download=True, transform=train_transform
)
val_ds = datasets.CIFAR10(
    root="./data", train=False, download=True, transform=val_transform
)

train_loader = DataLoader(train_ds, batch_size=64, shuffle=True, num_workers=4)
val_loader = DataLoader(val_ds, batch_size=64, shuffle=False, num_workers=4)

#  3. Load & modify VGG16
model = models.vgg16(weights=models.VGG16_Weights.IMAGENET1K_V1).to(device)

# Freeze convolutional backbone
for param in model.features.parameters():
    param.requires_grad = False

# Replace classifier head
n_classes = 10
model.classifier = nn.Sequential(
    nn.Linear(25088, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
```

```
    nn.Linear(4096, 1024),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(1024, n_classes),
).to(device)

#  4. Loss & optimizer (only head params)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),␣
 ↪lr=1e-4)


#  5. Training / validation functions
def train_epoch(model, loader, optimizer, criterion):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for imgs, labels in loader:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(imgs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * imgs.size(0)
        _, preds = outputs.max(1)
        correct += preds.eq(labels).sum().item()
        total += labels.size(0)
    return running_loss / total, correct / total


def eval_epoch(model, loader, criterion):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for imgs, labels in loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            running_loss += loss.item() * imgs.size(0)
            _, preds = outputs.max(1)
            correct += preds.eq(labels).sum().item()
            total += labels.size(0)
    return running_loss / total, correct / total
```

```
#  6. Full training loop
n_epochs = 10
for epoch in range(1, n_epochs + 1):
    train_loss, train_acc = train_epoch(model, train_loader, optimizer,␣
 ↪criterion)
    val_loss, val_acc = eval_epoch(model, val_loader, criterion)
    print(
        f"Epoch {epoch}/{n_epochs}  "
        f"Train: loss={train_loss:.4f}, acc={train_acc:.4f}  "
        f"Val:   loss={val_loss:.4f}, acc={val_acc:.4f}"
    )

print(" Fine-tuning complete!")
```

```
Using device: cuda

/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624:
UserWarning: This DataLoader will create 4 worker processes in total. Our
suggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker
number to avoid potential slowness/freeze if necessary.
  warnings.warn(

Epoch 1/10  Train: loss=1.0300, acc=0.6302  Val:   loss=0.4779, acc=0.8305
Epoch 2/10  Train: loss=0.8603, acc=0.6948  Val:   loss=0.4396, acc=0.8458
Epoch 3/10  Train: loss=0.8142, acc=0.7126  Val:   loss=0.3994, acc=0.8615
Epoch 4/10  Train: loss=0.7776, acc=0.7254  Val:   loss=0.3789, acc=0.8711
Epoch 5/10  Train: loss=0.7645, acc=0.7296  Val:   loss=0.3824, acc=0.8680
Epoch 6/10  Train: loss=0.7406, acc=0.7390  Val:   loss=0.3711, acc=0.8750
Epoch 7/10  Train: loss=0.7186, acc=0.7451  Val:   loss=0.3745, acc=0.8726
Epoch 8/10  Train: loss=0.7071, acc=0.7495  Val:   loss=0.3608, acc=0.8773
Epoch 9/10  Train: loss=0.6956, acc=0.7544  Val:   loss=0.3520, acc=0.8818
Epoch 10/10  Train: loss=0.6837, acc=0.7592  Val:   loss=0.3466, acc=0.8839
 Fine-tuning complete!
```