baseline\_model.py

*import* torch

*import* torch.nn *as* nn

*import* torch.optim *as* optim

*from* torchvision *import* transforms

*from* torch.utils.data *import* DataLoader

*from* tqdm *import* tqdm

*import* os

*from* models.simple\_cnn *import* SimpleCNN

*from* data.load\_data *import* get\_cifar10\_dataloaders

*from* utils.metrics *import* compute\_accuracy, save\_results  *# Make sure this path is correct*

*# Set device*

device = torch.device("cuda" *if* torch.cuda.is\_available() *else* "cpu")

*# Load data*

transform = transforms.Compose([

    transforms.ToTensor(),

    transforms.Normalize((0.5,), (0.5,))

])

train\_loader, test\_loader = get\_cifar10\_dataloaders()

*# Model, loss, optimizer*

model = SimpleCNN().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), *lr*=0.001)

*# Training*

num\_epochs = 10

epoch\_accuracies = []

*for* epoch *in* range(num\_epochs):

    model.train()

    loop = tqdm(train\_loader, *desc*=f"Epoch [{epoch+1}/{num\_epochs}]", *leave*=False)

*for* inputs, labels *in* loop:

        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero\_grad()

        outputs = model(inputs)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

*# Evaluation*

    accuracy = compute\_accuracy(model, test\_loader, device)

    epoch\_accuracies.append(accuracy)

    print(f"✅ Epoch [{epoch+1}/{num\_epochs}] - Test Accuracy: {accuracy:.2f}%")

*# Save results*

metrics\_dict = {

    "num\_epochs": num\_epochs,

    "final\_accuracy": epoch\_accuracies[-1],

    "epoch\_accuracies": epoch\_accuracies

}

save\_results(metrics\_dict, *path*="results/baseline\_metrics.json")

Simple CNN

*import* torch.nn *as* nn

class SimpleCNN(nn.Module):

    def \_\_init\_\_(*self*, *num\_classes*=10):

        super(SimpleCNN, *self*).\_\_init\_\_()

*self*.net = nn.Sequential(

            nn.Conv2d(3, 32, 3, *padding*=1),

            nn.ReLU(),

            nn.Conv2d(32, 64, 3, *padding*=1),

            nn.ReLU(),

            nn.MaxPool2d(2, 2),

            nn.Conv2d(64, 128, 3, *padding*=1),

            nn.ReLU(),

            nn.MaxPool2d(2, 2),

            nn.Flatten(),

            nn.Linear(8 \* 8 \* 128, 256),

            nn.ReLU(),

            nn.Linear(256, *num\_classes*),

        )

    def forward(*self*, *x*):

*return* *self*.net(*x*)

Load Data

*import* torch

*import* torchvision

*import* torchvision.transforms *as* transforms

def get\_cifar10(*batch\_size*=64):

    transform = transforms.Compose(

        [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]

    )

    train\_set = torchvision.datasets.CIFAR10(

*root*="./data", *train*=True, *download*=True, *transform*=transform

    )

    train\_loader = torch.utils.data.DataLoader(

        train\_set, *batch\_size*=*batch\_size*, *shuffle*=True

    )

    test\_set = torchvision.datasets.CIFAR10(

*root*="./data", *train*=False, *download*=True, *transform*=transform

    )

    test\_loader = torch.utils.data.DataLoader(

        test\_set, *batch\_size*=*batch\_size*, *shuffle*=False

    )

*return* train\_loader, test\_loader

*if* \_\_name\_\_ == "\_\_main\_\_":

    train\_loader, test\_loader = get\_cifar10()

    print(f"Train batches: {len(train\_loader)}")

    print(f"Test batches: {len(test\_loader)}")

metrics.py

*import* torch

*import* json

*import* os

def compute\_accuracy(*model*, *dataloader*, *device*):

*model*.eval()

    correct = total = 0

*with* torch.no\_grad():

*for* inputs, labels *in* *dataloader*:

            inputs, labels = inputs.to(*device*), labels.to(*device*)

            outputs = *model*(inputs)

            \_, predicted = torch.max(outputs, 1)

            total += labels.size(0)

            correct += (predicted == labels).sum().item()

*return* 100 \* correct / total

def save\_results(*metrics\_dict*, *path*=None):

*# ✅ Set your absolute path here*

    results\_folder = r"C:\Users\Gunjan\Desktop\Research Paper\Self-Doubting AI\code\results"

    os.makedirs(results\_folder, *exist\_ok*=True)

*if* *path* is None:

*path* = os.path.join(results\_folder, "baseline\_metrics.json")

*with* open(*path*, "w") *as* f:

        json.dump(*metrics\_dict*, f, *indent*=2)

    print(f"✅ Saved results to: {*path*}")