

```
PS C:\Users\aarus\OneDrive\Desktop\SRM\DLT> & C:/Python313/python.exe c:/Users/aarus/OneDrive/Desktop/SRM/DLT/dltlab7.py
Epoch [1/10] Train Loss: 0.7439, Train Acc: 48.09% | Val Loss: 0.6924, Val Acc: 50.89%
Epoch [2/10] Train Loss: 0.6942, Train Acc: 47.87% | Val Loss: 0.6914, Val Acc: 49.11%
Epoch [3/10] Train Loss: 0.6960, Train Acc: 46.07% | Val Loss: 0.6907, Val Acc: 50.89%
Epoch [4/10] Train Loss: 0.6915, Train Acc: 52.58% | Val Loss: 0.6852, Val Acc: 63.39%
Epoch [5/10] Train Loss: 0.6904, Train Acc: 56.63% | Val Loss: 0.6799, Val Acc: 55.36%
Epoch [6/10] Train Loss: 0.6711, Train Acc: 58.43% | Val Loss: 0.6874, Val Acc: 52.68%
Epoch [7/10] Train Loss: 0.6598, Train Acc: 60.22% | Val Loss: 0.6445, Val Acc: 59.82%
Epoch [8/10] Train Loss: 0.6563, Train Acc: 65.39% | Val Loss: 0.6384, Val Acc: 68.75%
Epoch [9/10] Train Loss: 0.6482, Train Acc: 62.70% | Val Loss: 0.6427, Val Acc: 58.04%
Epoch [10/10] Train Loss: 0.6465, Train Acc: 62.70% | Val Loss: 0.6412, Val Acc: 59.82%
[✓] Training complete! Model saved as catdog_cnn.pth
PS C:\Users\aarus\OneDrive\Desktop\SRM\DLT>
```

```

1  ✓ import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  from torch.utils.data import DataLoader
5  from torchvision import datasets, transforms
6  import os
7
8  # =====
9  # 1. Data Preprocessing
10 # =====
11 data_dir = r"C:\Users\aarus\OneDrive\Desktop\SRM\DLT\archive (2)\train"
12
13 # Transformations (similar to ImageDataGenerator augmentations)
14 ✓ transform = transforms.Compose([
15     transforms.Resize((150, 150)),
16     transforms.RandomRotation(20),
17     transforms.RandomHorizontalFlip(),
18     transforms.RandomResizedCrop(150, scale=(0.8, 1.0)),
19     transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
20     transforms.ToTensor(),
21     transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]) # [-1, 1]
22 ])
23
24 # Training + Validation datasets
25 ✓ train_dataset = datasets.ImageFolder(
26     root=data_dir,
27     transform=transform
28 )
29
30 # Train-validation split
31 train_size = int(0.8 * len(train_dataset))
32 val_size = len(train_dataset) - train_size
33 train_data, val_data = torch.utils.data.random_split(train_dataset, [train_size, val_size])
34
35 # Data loaders
36 train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
37 val_loader = DataLoader(val_data, batch_size=32, shuffle=False)
38

```

```

38
39 # =====
40 # 2. Build CNN Model (from scratch)
41 # =====
42 class SimpleCNN(nn.Module):
43     def __init__(self):
44         super(SimpleCNN, self).__init__()
45         self.conv_layers = nn.Sequential(
46             nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
47             nn.ReLU(),
48             nn.MaxPool2d(kernel_size=2, stride=2),
49
50             nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
51             nn.ReLU(),
52             nn.MaxPool2d(kernel_size=2, stride=2),
53
54             nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
55             nn.ReLU(),
56             nn.MaxPool2d(kernel_size=2, stride=2),
57         )
58         self.fc_layers = nn.Sequential(
59             nn.Flatten(),
60             nn.Linear(128 * 18 * 18, 256), # Adjust based on image size
61             nn.ReLU(),
62             nn.Dropout(0.5),
63             nn.Linear(256, 1),
64             nn.Sigmoid()
65         )
66
67     def forward(self, x):
68         x = self.conv_layers(x)
69         x = self.fc_layers(x)
70         return x
71
72 # =====
73 # 3. Training Setup
74 # =====
75 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
76 model = SimpleCNN().to(device)

```

dtl7.py > ...

```
72 # =====
73 # 3. Training Setup
74 # =====
75 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
76 model = SimpleCNN().to(device)
77
78 criterion = nn.BCELoss() # Binary cross entropy
79 optimizer = optim.Adam(model.parameters(), lr=0.001)
80
81 # =====
82 # 4. Training Loop
83 # =====
84 epochs = 10
85 for epoch in range(epochs):
86     model.train()
87     running_loss = 0.0
88     correct, total = 0, 0
89
90     for images, labels in train_loader:
91         images, labels = images.to(device), labels.to(device).float().unsqueeze(1)
92
93         optimizer.zero_grad()
94         outputs = model(images)
95         loss = criterion(outputs, labels)
96         loss.backward()
97         optimizer.step()
98
99         running_loss += loss.item()
100         preds = (outputs > 0.5).float()
101         correct += (preds == labels).sum().item()
102         total += labels.size(0)
103
104     train_acc = 100 * correct / total
105
106     # Validation
107     model.eval()
108     val_loss, val_correct, val_total = 0, 0, 0
109     with torch.no_grad():
110         for images, labels in val_loader:
```

```

98
99     running_loss += loss.item()
100     preds = (outputs > 0.5).float()
101     correct += (preds == labels).sum().item()
102     total += labels.size(0)
103
104 train_acc = 100 * correct / total
105
106 # Validation
107 model.eval()
108 val_loss, val_correct, val_total = 0, 0, 0
109 with torch.no_grad():
110     for images, labels in val_loader:
111         images, labels = images.to(device), labels.to(device).float().unsqueeze(1)
112         outputs = model(images)
113         loss = criterion(outputs, labels)
114         val_loss += loss.item()
115
116         preds = (outputs > 0.5).float()
117         val_correct += (preds == labels).sum().item()
118         val_total += labels.size(0)
119
120 val_acc = 100 * val_correct / val_total
121 print(f"Epoch [{epoch+1}/{epochs}] "
122       f"Train Loss: {running_loss/len(train_loader):.4f}, Train Acc: {train_acc:.2f}% "
123       f"| Val Loss: {val_loss/len(val_loader):.4f}, Val Acc: {val_acc:.2f}%")
124
125 # =====
126 # 5. Save Model
127 # =====
128 torch.save(model.state_dict(), "catdog_cnn.pth")
129 print("✅ Training complete! Model saved as catdog_cnn.pth")
130

```