CENG403 - Computer Vision Quiz: Homework set THE-2
Coding Questions - Set 1

Student Name: ____________________

Problem 1

Bilinear Interpolation Implementation

Complete the bilinear interpolation function for deformable convolution:

```python
def bilinear_interpolate(feature_map, y, x):
    """
    feature_map: 2D numpy array (H, W)
    y, x: float coordinates
    """
    # Get integer coordinates
    y0, x0 = int(np.floor(y)), int(np.floor(x))
    y1, x1 = __________, __________

    # Calculate weights
    wy = y - y0
    wx = x - x0

    # Get corner values (assume boundary handling done)
    v00 = feature_map[y0, x0]
    v01 = feature_map[y0, x1]
    v10 = feature_map[y1, x0]
    v11 = feature_map[y1, x1]

    # Bilinear interpolation formula
    result = __________

    return result
```

Fix the bug in this boundary condition check:

```python
def check_bounds(y, x, height, width):
    if y >= 0 and y < height and x >= 0 and x < width:
        return True
    return False

# Usage in deformable conv
if check_bounds(sample_y, sample_x, H, W):
    value = bilinear_interpolate(input_map, sample_y, sample_x)
else:
    value = 0  # What's wrong with this approach?
```

What's the bug and how to fix it?

Problem 2

CNN Training Loop Implementation

Complete the training function:

```
def train_epoch(model, train_loader, optimizer, criterion, device):
    model.__________()  # Set correct mode

    running_loss = 0.0
    correct_top1 = 0
    total_samples = 0

    for batch_idx, (images, labels) in enumerate(train_loader):
        # Move to device
        images = __________
        labels = __________

        # Clear gradients
        __________

        # Forward pass
        outputs = __________
        loss = __________

        # Backward pass
        __________
        __________

        # Calculate accuracy
        _, predicted = torch.max(outputs.data, 1)
        total_samples += labels.size(0)
        correct_top1 += __________

        running_loss += loss.item()

    avg_loss = running_loss / len(train_loader)
    accuracy = 100.0 * correct_top1 / total_samples

    return avg_loss, accuracy
```

Identify and fix the error in this validation loop:

```
def validate(model, val_loader, criterion, device):
    model.eval()

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()  # <-- What's wrong here?

            # ... accuracy calculation
```

Error: ____

Fix: ____

Problem 3

Top-K Accuracy Implementation

Complete the top-5 accuracy calculation:

```
def calculate_topk_accuracy(outputs, labels, k=5):
    """
    outputs: tensor of shape (batch_size, num_classes)
    labels: tensor of shape (batch_size,)
    k: int, top-k accuracy
    """
    batch_size = labels.size(0)

    # Get top-k predictions
    _, topk_pred = torch.topk(outputs, k, dim=1)

    # Check if true labels are in top-k predictions
    correct = __________

    # Count correct predictions
    correct_count = __________

    accuracy = 100.0 * correct_count / batch_size
    return accuracy
```

Debug this accuracy calculation - what's wrong?

```
# In training loop
_, predicted = torch.max(outputs, 1)
correct = (predicted == labels).sum()
accuracy = correct / len(train_loader)  # Bug here!

print(f"Training␣accuracy:␣{accuracy:.2f}%")
```

Issue: ____

Correct version:

Problem 4

RNN Forward Pass Implementation

Complete the RNN forward pass:

```python
def rnn_forward(inputs, W_xh, W_hh, b_xh, b_hh, W_hy, b_y, hidden_size):
    """
    inputs: list of one-hot vectors
    W_xh: input-to-hidden weights (hidden_size, vocab_size)
    W_hh: hidden-to-hidden weights (hidden_size, hidden_size)
    """
    seq_len = len(inputs)
    h = torch.zeros(hidden_size)
    outputs = []

    for t in range(seq_len):
        # Current input
        x_t = inputs[t]

        # Update hidden state
        h = torch.tanh(__________ + __________ + __________ + __________)

        # Compute output
        logits = __________
        outputs.append(logits)

    return outputs, h
```

Fix the gradient computation bug:

```python
# Manual gradient computation
W_xh = torch.randn(H, V, requires_grad=True)
W_hh = torch.randn(H, H, requires_grad=True)

# ... forward pass and loss calculation

# Compute gradients
grad_W_xh = torch.autograd.grad(loss, W_xh)  # Bug!
grad_W_hh = torch.autograd.grad(loss, W_hh)  # Bug!

print(grad_W_xh.shape)  # This will error
```

What's wrong and how to fix?

Problem 5

Data Preprocessing and Loading

Complete the CIFAR100 data loading setup:

```
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, random_split

# Define transforms
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(10),
    __________,  # Convert to tensor
    __________   # Normalize (use CIFAR100 stats if needed)
])

test_transform = transforms.Compose([
    __________,
    transforms.Normalize((0.5071, 0.4867, 0.4408),
                         (0.2675, 0.2565, 0.2761))
])

# Load dataset
full_train_dataset = datasets.CIFAR100(
    root='./data', train=__________,
    download=True, transform=train_transform
)

# Split dataset: 80% train, 20% validation
train_size = int(0.8 * len(full_train_dataset))
val_size = __________

train_dataset, val_dataset = random_split(
    full_train_dataset, [__________, __________]
)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=__________)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=__________)
```

What's wrong with this validation dataset setup?

```
train_dataset, val_dataset = random_split(full_train_dataset, [train_size,
    val_size])
# Problem: val_dataset still uses train_transform with data augmentation!
```

How to fix: ____

Problem 6

Model Architecture Implementation

Complete the CNN model definition:

```python
import torch.nn as nn
import torch.nn.functional as F

class CustomCNN(nn.Module):
    def __init__(self, num_classes=100):
        super(CustomCNN, self).__init__()

        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)

        # Pooling
        self.pool = nn.MaxPool2d(2, 2)

        # Fully connected layers
        # CIFAR100 images are 32x32, after 3 pooling operations: 32->16->8->4
        self.fc1 = nn.Linear(__________, 512)
        self.fc2 = nn.Linear(512, num_classes)

        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        # Conv block 1
        x = F.relu(self.conv1(x))
        x = self.pool(x)

        # Conv block 2
        x = __________
        x = __________

        # Conv block 3
        x = F.relu(self.conv3(x))
        x = self.pool(x)

        # Flatten
        x = x.view(x.size(0), -1)

        # Fully connected
        x = F.relu(self.fc1(x))
        x = __________  # Apply dropout
        x = self.fc2(x)

        return x
```

Calculate the input size for fc1 layer:

Calculation: ____

Problem 7

Loss Function and Optimizer Setup

Complete the training setup:

```
# Model setup
model = CustomCNN(num_classes=100)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = __________

# Loss function for multi-class classification
criterion = __________

# Optimizer setup
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=0.001,
    weight_decay=__________  # L2 regularization
)

# Learning rate scheduler (optional)
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=10,
    gamma=__________  # Decay factor
)
```

Debug this optimizer issue:

```
# Training loop
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()

        # What's missing here?

        if batch_idx % 100 == 0:
            print(f'Loss:␣{loss.item():.6f}')
```

Missing line: ____

Problem 8

Character-Level RNN Setup

Complete the character preprocessing:

```
text = "Deep␣Learning"

# Create vocabulary
chars = sorted(list(set(text)))
char2idx = __{char: i for __________ in __________}__
idx2char = __{i: char for __________ in __________}__

# Create input and target sequences
input_seq = text[:-1]  # "Deep Learnin"
target_seq = text[1:]  # "eep Learning"

# Convert to one-hot vectors
def char_to_onehot(char, vocab_size):
    vec = torch.zeros(vocab_size)
    vec[__________] = 1
    return vec

# Convert sequences
inputs = [char_to_onehot(char, len(chars)) for char in __________]
targets = [char2idx[char] for char in __________]

print(f"Vocab␣size:␣{len(chars)}")
print(f"Input␣sequence␣length:␣{len(inputs)}")
print(f"Target␣sequence␣length:␣{len(targets)}")
```

Complete the RNN parameter initialization:

```
import torch
import torch.nn.functional as F

# Hyperparameters
V = len(chars)    # Vocabulary size
H = 16            # Hidden size
seq_len = len(input_seq)

# Initialize parameters
W_xh = torch.randn(H, V, requires_grad=True) * 0.01
W_hh = torch.randn(__________, requires_grad=True) * 0.01
b_xh = torch.zeros(__________, requires_grad=True)
b_hh = torch.zeros(H, requires_grad=True)
W_hy = torch.randn(__________, requires_grad=True) * 0.01
b_y = torch.zeros(__________, requires_grad=True)

print(f"W_xh␣shape:␣{W_xh.shape}")
print(f"W_hh␣shape:␣{W_hh.shape}")
print(f"W_hy␣shape:␣{W_hy.shape}")
```