# CENG403 - Spring 2025: Homework set Write Code Blocks Practice

Your Name

## TASK 1: DEFORMABLE CNN - WRITE CODE BLOCKS

.1: Floor Operation

### Problem 1.1: Floor Operation

Write code to get the floor of a float value q_y and store it in y0:

.2: Ceiling Operation

### Problem 1.2: Ceiling Operation

Write code to get the ceiling of a float value q_x and store it in x1:

.3: Kernel Index Calculation

### Problem 1.3: Kernel Index Calculation

Write code to calculate linear kernel index from 2D position (kh, kw) with kernel width K_w:

.4: Y Offset Extraction

### Problem 1.4: Y Offset Extraction

Write code to extract Y offset from delta tensor for batch n, kernel index k, output position (h_out, w_out):

.5: X Offset Extraction

## Problem 1.5: X Offset Extraction

Write code to extract X offset from delta tensor for batch n, kernel index k, output position (h_out, w_out):

.6: Base Position Calculation

## Problem 1.6: Base Position Calculation

Write code to calculate base sampling position h_start from output position h_out and stride:

.7: Final Sampling Position

## Problem 1.7: Final Sampling Position

Write code to calculate final sampling position sample_y from h_start, kh, dilation, and delta_y:

.8: Bounds Check Condition

## Problem 1.8: Bounds Check Condition

Write code to check if coordinates (y, x) are within image bounds (H, W):

.9: Safe Pixel Access

## Problem 1.9: Safe Pixel Access

Write code to get pixel value at (y, x) from image img, returning 0.0 if out of bounds:

.10: Fractional Part Calculation

## Problem 1.10: Fractional Part Calculation

Write code to calculate fractional part dx from q_x and its floor x0:

.11: Linear Interpolation

## Problem 1.11: Linear Interpolation

Write code to linearly interpolate between v_left and v_right using weight dx:

.12: Bilinear Weight Calculation

## Problem 1.12: Bilinear Weight Calculation

Write code to calculate bilinear interpolation weight for points p and q:

.13: Value Accumulation

## Problem 1.13: Value Accumulation

Write code to accumulate weighted interpolated value to current_value using weight, mask, and interpolated:

.14: Corner Position Calculation

## Problem 1.14: Corner Position Calculation

Write code to calculate all four corner positions (y0, x0), (y0, x1), (y1, x0), (y1, x1) from q_y and q_x:

.15: Mask Application

## Problem 1.15: Mask Application

Write code to apply modulation mask m_k to an interpolated value:

# TASK 2: CNN PYTORCH - WRITE CODE BLOCKS

.1: Conv Layer Definition

## Problem 2.1: Conv Layer Definition

Write code to define a Conv2d layer with 32 input channels, 64 output channels, kernel size 3, padding 1:

.2: BatchNorm Layer Definition

## Problem 2.2: BatchNorm Layer Definition

Write code to define a BatchNorm2d layer for 128 channels:

.3: MaxPool Layer Definition

## Problem 2.3: MaxPool Layer Definition

Write code to define a MaxPool2d layer that halves spatial dimensions:

.4: Linear Layer Definition

## Problem 2.4: Linear Layer Definition

Write code to define a Linear layer with 4096 inputs and 512 outputs:

.5: Dropout Layer Definition

## Problem 2.5: Dropout Layer Definition

Write code to define a Dropout layer with probability 0.5:

.6: ReLU Activation

## Problem 2.6: ReLU Activation

Write code to apply ReLU activation to variable x:

.7: Tensor Flattening

## Problem 2.7: Tensor Flattening

Write code to flatten tensor x while preserving batch dimension:

.8: Device Transfer

## Problem 2.8: Device Transfer

Write code to move tensor images to device:

.9: Optimizer Zero Grad

## Problem 2.9: Optimizer Zero Grad

Write code to clear gradients from optimizer:

.10: Forward Pass

## Problem 2.10: Forward Pass

Write code to perform forward pass through model with input images:

.11: Loss Calculation

## Problem 2.11: Loss Calculation

Write code to calculate loss using criterion with outputs and labels:

.12: Backward Pass

## Problem 2.12: Backward Pass

Write code to perform backward pass on loss:

.13: Optimizer Step

## Problem 2.13: Optimizer Step

Write code to update model parameters using optimizer:

.14: Top-1 Prediction

## Problem 2.14: Top-1 Prediction

Write code to get top-1 predictions from outputs:

.15: Accuracy Calculation

## Problem 2.15: Accuracy Calculation

Write code to calculate accuracy from predicted and labels tensors:

.16: Model Training Mode

## Problem 2.16: Model Training Mode

Write code to set model to training mode:

.17: Model Evaluation Mode

## Problem 2.17: Model Evaluation Mode

Write code to set model to evaluation mode:

.18: No Gradient Context

## Problem 2.18: No Gradient Context

Write code to create context where gradients are disabled:

.19: Dataset Size Calculation

## Problem 2.19: Dataset Size Calculation

Write code to calculate training size as 80% of total dataset size:

.20: Random Split

## Problem 2.20: Random Split

Write code to split dataset into train_set and val_set with sizes train_size and val_size:

# TASK 3: RNN - WRITE CODE BLOCKS

.1: Character Set Creation

## Problem 3.1: Character Set Creation

Write code to create sorted list of unique characters from text:

.2: Character to Index Mapping

## Problem 3.2: Character to Index Mapping

Write code to create dictionary mapping each character to its index:

.3: Index to Character Mapping

## Problem 3.3: Index to Character Mapping

Write code to create dictionary mapping each index to its character:

.4: Input Sequence Creation

## Problem 3.4: Input Sequence Creation

Write code to create input sequence (all characters except last) from text:

.5: Target Sequence Creation

## Problem 3.5: Target Sequence Creation

Write code to create target sequence (all characters except first) from text:

.6: One-Hot Vector Creation

## Problem 3.6: One-Hot Vector Creation

Write code to create one-hot vector of given size with 1 at given index:

.7: Input-to-Hidden Weight Initialization

## Problem 3.7: Input-to-Hidden Weight Initialization

Write code to initialize W_xh weight matrix for RNN with hidden size H and vocab size V:

.8: Hidden-to-Hidden Weight Initialization

## Problem 3.8: Hidden-to-Hidden Weight Initialization

Write code to initialize W_hh weight matrix for RNN with hidden size H:

.9: Hidden Bias Initialization

## Problem 3.9: Hidden Bias Initialization

Write code to initialize bias vector b_xh for hidden layer with size H:

.10: Output Weight Initialization

## Problem 3.10: Output Weight Initialization

Write code to initialize W_hy weight matrix from hidden to output with vocab size V and hidden size H:

.11: Hidden State Initialization

## Problem 3.11: Hidden State Initialization

Write code to initialize hidden state vector with zeros of size H:

.12: Input Contribution Calculation

## Problem 3.12: Input Contribution Calculation

Write code to calculate input contribution to hidden state using W_xh and x_t:

.13: Hidden Recurrence Calculation

## Problem 3.13: Hidden Recurrence Calculation

Write code to calculate recurrent contribution to hidden state using W_hh and previous hidden state h:

.14: Hidden State Update

## Problem 3.14: Hidden State Update

Write code to update hidden state using tanh activation with all contributions and biases:

.15: Output Logits Calculation

## Problem 3.15: Output Logits Calculation

Write code to calculate output logits s_t from hidden state h using W_hy and b_y:

.16: Character Index Lookup

## Problem 3.16: Character Index Lookup

Write code to get index of character 'e' from char2idx dictionary:

.17: Input List Creation

## Problem 3.17: Input List Creation

Write code to convert input sequence to list of one-hot vectors:

.18: Target Tensor Creation

## Problem 3.18: Target Tensor Creation

Write code to convert target sequence to tensor of character indices:

.19: Logits Stacking

## Problem 3.19: Logits Stacking

Write code to stack list of logits tensors into single tensor:

.20: Log Softmax Calculation

## Problem 3.20: Log Softmax Calculation

Write code to calculate log softmax of logits along vocabulary dimension:

.21: NLL Loss Calculation

## Problem 3.21: NLL Loss Calculation

Write code to calculate negative log likelihood loss from log_probs and targets:

.22: Gradient Calculation

## Problem 3.22: Gradient Calculation

Write code to calculate gradient of loss with respect to W_xh:

# DATA PREPROCESSING - WRITE CODE BLOCKS

.1: CIFAR-100 Mean Values

## Problem 4.1: CIFAR-100 Mean Values

Write code to define CIFAR-100 normalization mean values:

.2: CIFAR-100 Std Values

## Problem 4.2: CIFAR-100 Std Values

Write code to define CIFAR-100 normalization standard deviation values:

.3: ToTensor Transform

## Problem 4.3: ToTensor Transform

Write code to create ToTensor transform:

.4: Normalization Transform

## Problem 4.4: Normalization Transform

Write code to create Normalize transform with CIFAR-100 mean and std:

.5: Random Crop Transform

## Problem 4.5: Random Crop Transform

Write code to create RandomCrop transform with size 32 and padding 4:

.6: Random Flip Transform

## Problem 4.6: Random Flip Transform

Write code to create RandomHorizontalFlip transform:

.7: Transform Composition

## Problem 4.7: Transform Composition

Write code to compose multiple transforms into single transform:

.8: CIFAR-100 Dataset Loading

## Problem 4.8: CIFAR-100 Dataset Loading

Write code to load CIFAR-100 training dataset with transform:

.9: DataLoader Creation

## Problem 4.9: DataLoader Creation

Write code to create DataLoader with batch size 128 and shuffle=True:

# OPTIMIZATION AND TRAINING - WRITE CODE BLOCKS

.1: CrossEntropy Loss Definition

## Problem 5.1: CrossEntropy Loss Definition

Write code to define CrossEntropy loss function:

.2: SGD Optimizer Definition

## Problem 5.2: SGD Optimizer Definition

Write code to define SGD optimizer with learning rate 0.01 and momentum 0.9:

.3: Adam Optimizer Definition

## Problem 5.3: Adam Optimizer Definition

Write code to define Adam optimizer with learning rate 0.001:

.4: Model Parameter Count

## Problem 5.4: Model Parameter Count

Write code to count total number of parameters in model:

.5: Learning Rate Update

## Problem 5.5: Learning Rate Update

Write code to multiply learning rate by 0.1 for all parameter groups:

.6: Model State Save

## Problem 5.6: Model State Save

Write code to save model state dictionary to file 'model.pth':

.7: Model State Load

## Problem 5.7: Model State Load

Write code to load model state dictionary from file 'model.pth':

.8: Gradient Clipping

## Problem 5.8: Gradient Clipping

Write code to clip gradients to maximum norm of 1.0:

3cm

.9: Top-5 Accuracy

## Problem 5.9: Top-5 Accuracy

Write code to calculate top-5 accuracy from outputs and labels:

## Problem 5.10: Loss Item Extraction

Write code to extract scalar value from loss tensor:

# DEBUGGING AND UTILITIES - WRITE CODE BLOCKS

.1: Tensor Shape Check

## Problem 6.1: Tensor Shape Check

Write code to print shape of tensor x:

.2: Tensor Device Check

## Problem 6.2: Tensor Device Check

Write code to check which device tensor x is on:

.3: Model Device Transfer

## Problem 6.3: Model Device Transfer

Write code to move entire model to GPU:

.4: Gradient Existence Check

## Problem 6.4: Gradient Existence Check

Write code to check if parameter has gradients:

.5: Memory Usage Check

## Problem 6.5: Memory Usage Check

Write code to check CUDA memory usage:

.6: Random Seed Setting

## Problem 6.6: Random Seed Setting

Write code to set PyTorch random seed to 42:

.7: Numpy Seed Setting

## Problem 6.7: Numpy Seed Setting

Write code to set numpy random seed to 42:

.8: Model Summary

## Problem 6.8: Model Summary

Write code to print model architecture:

.9: Batch Dimension Check

## Problem 6.9: Batch Dimension Check

Write code to get batch size from tensor x:

## Problem 6.10: Tensor Type Conversion

Write code to convert tensor x to float type:

# ANSWER BANK

Task 1 - Deformable CNN Answers:

## Problem Task 1 - Deformable CNN Answers:

1.1: `y0 = int(np.floor(q_y))`

1.2: `x1 = int(np.ceil(q_x))`

1.3: `k = kh * K_w + kw`

1.4: `delta_y = delta[n, 2 * k, h_out, w_out]`

1.5: `delta_x = delta[n, 2 * k + 1, h_out, w_out]`

1.6: `h_start = h_out * stride`

1.7: `sample_y = h_start + kh * dilation + delta_y`

1.8: `if 0 <= y < H and 0 <= x < W:`

1.9: `value = img[y, x] if (0 <= y < H and 0 <= x < W) else 0.0`

1.10: `dx = q_x - x0`

1.11: `result = v_left * (1 - dx) + v_right * dx`

1.12: `weight = (1 - abs(p_x - q_x)) * (1 - abs(p_y - q_y))`

1.13: `current_value += weight * mask * interpolated`

1.14:
```
y0, x0 = int(np.floor(q_y)), int(np.floor(q_x))
y1, x1 = y0 + 1, x0 + 1
```

1.15: `modulated_value = m_k * interpolated_value`

Task 2 - CNN PyTorch Answers:

## Problem Task 2 - CNN PyTorch Answers:

2.1: `self.conv = nn.Conv2d(32, 64, kernel_size=3, padding=1)`

2.2: `self.bn = nn.BatchNorm2d(128)`

2.3: `self.pool = nn.MaxPool2d(2, 2)`

2.4: `self.fc = nn.Linear(4096, 512)`

2.5: `self.dropout = nn.Dropout(0.5)`

2.6: `x = F.relu(x)`

2.7: `x = x.view(x.size(0), -1)`

2.8: `images = images.to(device)`

2.9: `optimizer.zero_grad()`

2.10: `outputs = model(images)`

2.11: `loss = criterion(outputs, labels)`

2.12: `loss.backward()`

2.13: `optimizer.step()`

2.14: `_, predicted = torch.max(outputs, 1)`

2.15: `accuracy = (predicted == labels).sum().item() / labels.size(0) * 100`

2.16: `model.train()`

2.17: `model.eval()`

2.18: `with torch.no_grad():`

2.19: `train_size = int(0.8 * len(dataset))`

2.20: `train_set, val_set = random_split(dataset, [train_size, val_size])`

Task 3 - RNN Answers:

## Problem Task 3 - RNN Answers:

3.1: `chars = sorted(list(set(text)))`

3.2: `char2idx = {ch: i for i, ch in enumerate(chars)}`

3.3: `idx2char = {i: ch for i, ch in enumerate(chars)}`

3.4: `input_seq = text[:-1]`

3.5: `target_seq = text[1:]`

3.6:
```
vec = torch.zeros(size)
vec[idx] = 1.0
```

3.7: `W_xh = torch.randn(H, V, requires_grad=True) * 0.1`

3.8: `W_hh = torch.randn(H, H, requires_grad=True) * 0.1`

3.9: `b_xh = torch.zeros(H, requires_grad=True)`

3.10: `W_hy = torch.randn(V, H, requires_grad=True) * 0.1`

3.11: `h = torch.zeros(H)`

3.12: `input_contrib = W_xh @ x_t`

3.13: `hidden_contrib = W_hh @ h`

3.14: `h = torch.tanh(W_xh @ x_t + b_xh + W_hh @ h + b_hh)`

3.15: `s_t = W_hy @ h + b_y`

3.16: `idx = char2idx['e']`

3.17: `inputs = [one_hot(char2idx[ch], V) for ch in input_seq]`

3.18: `targets = torch.tensor([char2idx[ch] for ch in target_seq], dtype=torch.long)`

3.19: `logits = torch.stack(logits_list)`

3.20: `log_probs = F.log_softmax(logits, dim=1)`

3.21: `loss = F.nll_loss(log_probs, targets)`

3.22: `grad_W_xh = torch.autograd.grad(loss, W_xh, retain_graph=True)[0]`

Additional Sections Answers:

## Problem Additional Sections Answers:

4.1: `mean = [0.5071, 0.4867, 0.4408]`

4.2: `std = [0.2675, 0.2565, 0.2761]`

4.3: `transform = transforms.ToTensor()`

4.4: `normalize = transforms.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565, 0.2761])`

4.5: `crop = transforms.RandomCrop(32, padding=4)`

4.6: `flip = transforms.RandomHorizontalFlip()`

4.7: `transform = transforms.Compose([transform1, transform2, ...])`

4.8: `dataset = CIFAR100(root='./data', train=True, transform=transform)`

4.9: `loader = DataLoader(dataset, batch_size=128, shuffle=True)`

5.1: `criterion = nn.CrossEntropyLoss()`

5.2: `optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)`

5.3: `optimizer = optim.Adam(model.parameters(), lr=0.001)`

5.4: `total_params = sum(p.numel() for p in model.parameters())`

5.5: `for param_group in optimizer.param_groups:  param_group['lr'] *= 0.1`

5.6: `torch.save(model.state_dict(), 'model.pth')`

5.7: `model.load_state_dict(torch.load('model.pth'))`

5.8: `torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)`

5.9: `_, top5_pred = outputs.topk(5, dim=1)`
`top5_acc = top5_pred.eq(labels.view(-1, 1).expand_as(top5_pred)).sum().item()`

5.10: `loss_value = loss.item()`

6.1: `print(x.shape)`

6.2: `print(x.device)`

6.3: `model = model.to('cuda')`

6.4: `if param.grad is not None:`

6.5: `print(torch.cuda.memory_allocated())`

6.6: `torch.manual_seed(42)`

6.7: `np.random.seed(42)`

6.8: `print(model)`

6.9: `batch_size = x.size(0)`

6.10: `x = x.float()`

Additional Answers

## Problem Additional Answers

C1: 32 x 32, C2: 8 x 8, C3: 4096, C4: 73856, C5: 256

S1: numpy, torch.nn, F, transforms

S2: nn.Module, super

S3: def forward(self, x)

S4: nn.CrossEntropyLoss()

S5: optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

M1: x = self.dropout(x) if self.training else x

M2: for param_group in optimizer.param_groups: param_group['lr'] *= 0.1

M3: torch.save(model.state_dict(), 'model.pth')

M4: outputs = model(images.to(device))

M5: correct = (predicted == labels).sum().item()