Week 3 Sample Oral Assessment Questions

CSE151B SS1 2026

Below are questions that will be similar to what you'll see on this week's oral assessment. As mentioned in class, you'll be asked two questions–first an "understand" question, and then an "experiment" question. For questions where we imagine there are a few possible phrasings, there are multiple sample acceptable answers listed. Overall, if you completed the implementations for A1 and understood how and why they worked (either by thinking through or experimenting with different components), you should be well prepared.

Note that all of these contain excerpts from your assignment solution code for your reference, but you don't need to do a detailed read of every line of code to answer them–it's just there in case it helps as you answer the question!

Remember that we will only count your answer after you say **"My answer is..."** and we will only give hints after you answer "yes" to "Would you like a hint?" or answer incorrectly.

Understand:

1. [Topic: Convolutional Neural Network Metrics] -> probably stays as an exam question (might be too easy? Might wanna cut it?)

```python
def pixel_acc(pred, target):
    """
    Calculate pixel-wise accuracy between predictions and targets.

    Args:
        pred (tensor): Predicted output from the model. Shape: (N, n_class, H, W)
        target (tensor): Ground truth labels. Shape: (N, H, W)

    Returns:
        float: Pixel-wise accuracy.
    """
    pred = torch.argmax(pred, dim=1)  # (N, H, W)
    correct = (pred == target).sum().item()
    total = target.numel()
    return correct / total
```

Q: Above is an implementation of calculating the pixel accuracy in an image. What does the pixel accuracy measure?
Possible answer: My answer is: Pixel accuracy measures the number of correctly classified pixels divided by the total number of pixels.

Hints
Tier 1: "Walk through the last three lines — what are correct and total, and what does dividing them give?"
Tier 2: "correct counts pixels where prediction == ground truth; total is every pixel. So the ratio is…?"
Tier 3: "Fraction of pixels classified correctly = correct pixels ÷ total pixels."

2. [Topic: encoder downsampling] Every encoder conv uses stride=2, which halves the spatial resolution while the channel count keeps doubling (3→32→…→512). Why deliberately shrink the image like this. What does downsampling buy the model?

```
def __init__(self, n_class):
    super().__init__()
    self.n_class = n_class
    self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=2, padding=1, dilation=1)
    self.bnd1 = nn.BatchNorm2d(32)
    self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=2, padding=1, dilation=1)
    self.bnd2 = nn.BatchNorm2d(64)
    self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1, dilation=1)
    self.bnd3 = nn.BatchNorm2d(128)
    self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1, dilation=1)
    self.bnd4 = nn.BatchNorm2d(256)
    self.conv5 = nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1, dilation=1)
    self.bnd5 = nn.BatchNorm2d(512)
    self.relu = nn.ReLU(inplace=True)
    #Decoder
    self.deconv1 = nn.ConvTranspose2d(512, 256, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
    self.bn1 = nn.BatchNorm2d(256)
    self.deconv2 = nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
    self.bn2 = nn.BatchNorm2d(128)
    self.deconv3 = nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
    self.bn3 = nn.BatchNorm2d(64)
    self.deconv4 = nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
    self.bn4 = nn.BatchNorm2d(32)
    self.deconv5 = nn.ConvTranspose2d(32, 3, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
    self.bn5 = nn.BatchNorm2d(3)
    self.classifier = nn.Conv2d(3, self.n_class, kernel_size=1)
```

Possible answer: Downsampling rapidly grows each neuron's receptive field — after a few stride-2 layers a single neuron "sees" a large region of the original image, which is necessary to recognize objects (you can't tell "bird" from one pixel). It also lets the network build deeper, more abstract/semantic features (more channels) and is far cheaper to compute on smaller feature maps. In short: it trades spatial detail ("where") for semantic understanding ("what").

Hints:
Tier 1: "A single pixel isn't enough to recognize a bird. How does the network get each neuron to 'see' a larger region?"
Tier 2: "Stride-2 doubles the receptive field each layer. What does a bigger field of view let the network recognize — and what's the compute benefit?"

Tier 3: "It grows the receptive field so neurons see enough context to identify objects, builds deeper semantic features, and saves compute — trading 'where' for 'what.'"

Experiment:

1. [Topic: Convolutional Neural Network Architecture]

```
1  import torch.nn as nn
2  class FCN(nn.Module):
3
4      def __init__(self, n_class):
5          super().__init__()
6          self.n_class = n_class
7          self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=2, padding=1, dilation=1)
8          self.bnd1 = nn.BatchNorm2d(32)
9          self.deconv1 = nn.ConvTranspose2d(32, 3, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1)
10         self.bn1 = nn.BatchNorm2d(3)
11         self.relu = nn.ReLU(inplace=True)
12         self.classifier = nn.Conv2d(3, self.n_class, kernel_size=1)
13
14     def forward(self, x):
15         x1 = self.bnd1(self.relu(self.conv1(x)))
16         y1 = self.bn1(self.relu(self.deconv5(x1)))
17
18         score = self.classifier(y1)
19
20         return score  # size=(N, n_class, H, W)
```

Q: A student wants to scale up their CNN architecture from the image above. They change the first convolution to output 64 channels instead of 32 on line 7, **but they do not modify any other layer:**

self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1, dilation=1).

What would happen to the model?
Possible answer: My answer is: The model will fail at runtime. (Extra detail - bnd1 now gets the wrong channel count)

Hints
Tier 1: "Trace conv1's output through forward. It's 64 channels now — what's the very next layer that touches it, and how many channels does that layer expect?"
Tier 2: "The next layer is bnd1 = BatchNorm2d(32). It expects exactly 32 channels but now gets 64. What happens?"
Tier 3: "It crashes at runtime — bnd1 receives 64 channels and errors, because no downstream layer was updated to match."

2. [Topic: RNN Teacher Forcing]

```python
# Define RNN Model
class RNNModel(nn.Module):
    def __init__(self, vocab_size, hidden_size, embed_size, num_layers):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.rnn = nn.RNN(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, x):
        # x: (batch, seq_len)
        embed = self.embedding(x)                    # (batch, seq_len, embed_size)
        out, _ = self.rnn(embed)                     # (batch, seq_len, hidden_size)
        out = self.fc(out[:, -1, :])                 # (batch, vocab_size) — last timestep
        return out
```

Q: Above is an implementation of a RNN model. An experimenter is currently observing a test set loss of 1.4 when training this model from scratch using teacher forcing. If we train the same RNN model architecture from scratch (not continued training) without teacher forcing, what will happen to the model's test set loss?
Possible answer: My answer is: The loss will increase dramatically. This is reflected in your no teacher forcing experiments for the LSTM (loss increase from 1.4 to ~3).

Hints
Tier 1: "Without teacher forcing, what does the model consume as input during training — and how good are those inputs early on?"
Tier 2: "If it feeds its own bad early predictions back in, do errors get corrected or do they compound? What does that do to the loss?"
Tier 3: "Loss rises substantially — training on its own noisy outputs is much harder. Your LSTM run showed ~1.4 → ~3."