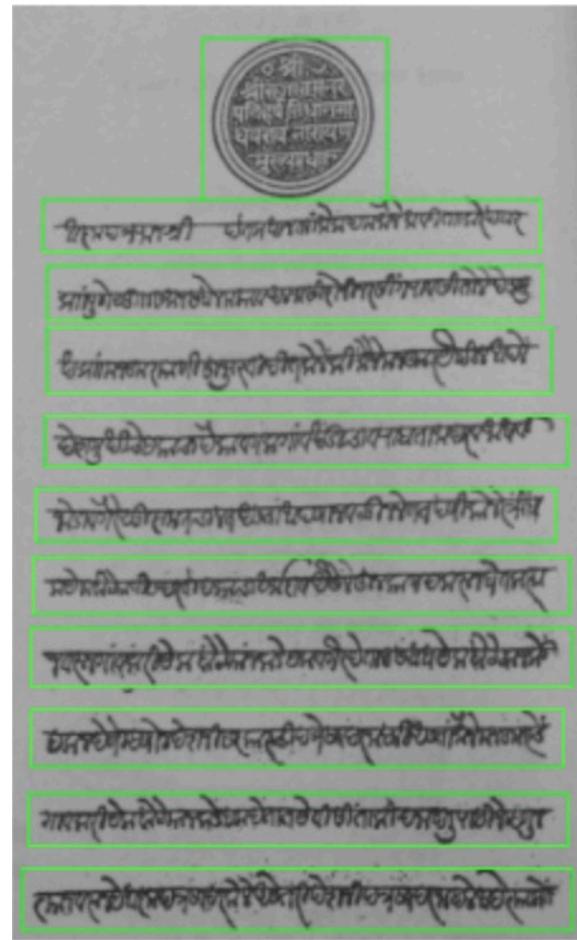
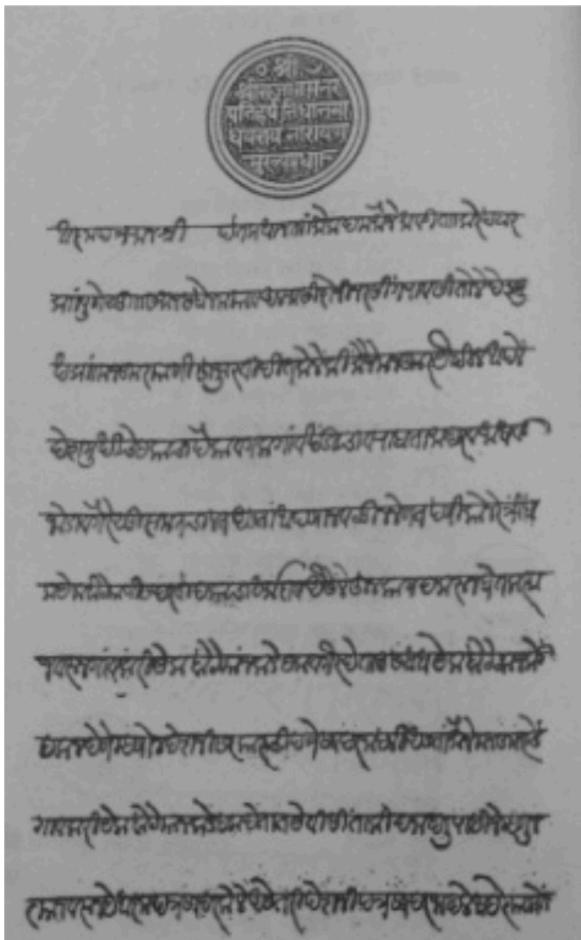


1. Given the following RGB input image (left), what image processing steps are required for obtaining the text regions as shown on the right (HINT: Morphological operations).



5

Solution:

1. Convert the image to grayscale.
 2. Binarise the image.
 3. Invert the binary mask.
 4. Dilate with a horizontal kernel.
 5. Detect contours and draw rectangles using cv2.rectangle.

2. Consider the following PyTorch architecture with Dropout.

```

import torch
import torch.nn as nn

class SimpleModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer = nn.Linear(10, 10)
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.layer(x)
        x = self.dropout(x)
        return x

# Initialize model and input
model = SimpleModel()
x_input = torch.randn(1, 10)

```

A:

```

model.eval()
with torch.no_grad():
    output = model(x_input)
    # Check output consistency...

```

B:

```

model.eval()
output = model(x_input)
# Check output consistency...

```

C:

```

with torch.no_grad():
    output = model(x_input)
    # Check output consistency...

```

D:

```

output = model(x_input)
# Check output consistency...

```

Answer the following questions (Select all cases that apply).

- If the code runs the `forward` pass multiple times on the exact same `x_input`, in which of the cases will the model produce different or inconsistent outputs across runs? [2]
- After the forward pass, if one attempts to calculate a loss and call `loss.backward()`, which of the cases will raise a RuntimeError? [2]
- You are writing a validation loop to check model accuracy on a test set. Which case represents the correct standard practice that you would use? [2]

Solution:

1. C and D
 2. A and C
 3. A
3. Consider the following PyTorch code intended to train a **Convolutional Neural Network (CNN)** for **binary image classification**.

```

import torch
import torch.nn as nn
import torch.optim as optim

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=0, stride=1)
        self.fc1 = nn.Linear(16 * 28 * 28, 64)
        self.fc2 = nn.Linear(64, 2)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x), dim=1)
        return x

model = CNN()
optimizer = optim.SGD(model.parameters(), lr=0.1)
criterion = nn.CrossEntropyLoss()

for epoch in range(3):
    for X, y in train_loader:
        optimizer.zero_grad()
        outputs = model(X, y)
        loss = criterion(outputs, y)

        optimizer.step()
        loss.backward()

```

Additional Information: `train_loader` yields mini-batches (X, y) . X has shape (batch_size, 1, 28, 28). y has shape (batch_size) with labels {0, 1}. Assume all hyperparameters are appropriate.

Identify all **buggy lines** in the code above (there are **exactly four**) and briefly explain the mistake in each.

Solution:

1. **Incorrect input dimension to the fully connected layer [2 marks]**

The line

```
self.fc1 = nn.Linear(16 * 28 * 28, 64)
```

is incorrect. After a 3×3 convolution with no padding and stride 1, the spatial dimensions reduce from 28×28 to 26×26 . Hence, the correct input size should be $16 \times 26 \times 26$.

2. **Incorrect forward call to the model** [1 mark]

The line

```
outputs = model(X, y)
```

is incorrect because the `forward()` method of the model expects only the input tensor X . The label tensor y should not be passed into the model.

3. **Incorrect order of backpropagation and parameter update** [1 mark]

The lines

```
optimizer.step() followed by loss.backward()
```

are in the wrong order. Gradients must be computed using backpropagation before updating the model parameters. The correct order is to call `loss.backward()` first and then `optimizer.step()`.

4. **Incorrect use of Softmax with Cross-Entropy Loss** [2 marks]

The line

```
x = torch.softmax(self.fc2(x), dim=1)
```

is incorrect when used with `nn.CrossEntropyLoss()`. `CrossEntropyLoss` internally applies `LogSoftmax`, so explicitly applying Softmax in the model leads to incorrect gradients.

The correct approach is to return raw logits from the final layer.

4. Consider the standard Batch Normalization operation applied to a neural network. The normalization process involves calculating batch statistics, normalizing the input, and finally applying a scale and shift operation.

1. Identify the specific parameters in a Batch Normalization layer that are updated via gradient descent (backpropagation).
2. **1D Batch Normalization (BatchNorm1d):** Consider a standard fully connected layer where the input tensor has shape (B, D) , with a batch size of B and feature dimension D . What is the total number of learnable parameters for this layer?
3. **2D Batch Normalization (BatchNorm2d):** Consider a convolutional layer output where the input tensor has shape (B, H, W, C) , representing a batch size of B , channel count C , height H , and width W . What is the total number of learnable parameters for this layer?

3

Solution:

1. The only learnable parameters are the scale (γ) and the shift (β).
2. $2 \times D$
3. $2 \times C$