Ahmedabad University

Course Name: Computer Vision

Weekly Report: 7

Group Name: Plain Vanilla Ice-cream

Submitted to faculty:

<u>Mehul Raval</u>

Date of Submission:

12th April 2025

Student Details

Roll No.	Name of the Student	Name of the Program
AU2240	Raj Koticha	B.Tech in CSE
AU2240	Dhruv Premani	B.Tech in CSE
AU2240085	Hariohm Bhatt	B.Tech in CSE

Table of Contents.

WORK DONE THIS WEEK

1. Introduction

This week, our focus shifted toward addressing challenges in class imbalance and multi-task learning for retinal imaging. The key advances include:

- **Segmentation:** We replaced our previous loss with a **Focal Loss** to better attend to hard-to-classify pixels, which yielded amazing segmentation results.
- **Disease Grading:** In parallel, we developed a separate disease grading network to predict the severity of retinal diseases.
- **Future Integration:** Building on these successes, our next step is to integrate localization and grading in a single U-Net framework.

Below, we detail our approach, code enhancements, and the outcomes observed.

---

2. Model Architecture

2.1 Improved U-Net with Focal Loss

To enhance segmentation quality, we modified our U-Net architecture to work seamlessly with Focal Loss. As before, our U-Net retains the encoder–decoder structure with skip connections. The change in the loss function better prioritizes the underrepresented pixels.

Residual Double Convolution Block

We continue using an enhanced double convolution block with residual connections to preserve spatial context:

```python
import torch
import torch.nn as nn

class ResidualDoubleConv(nn.Module):
        def __init__(self, in_channels, out_channels, drop_prob=0.1):
```

```python
        super(ResidualDoubleConv, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.dropout = nn.Dropout2d(drop_prob)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.residual_conv = None
        if in_channels != out_channels:
            self.residual_conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        identity = x if self.residual_conv is None else self.residual_conv(x)
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.dropout(out)
        out = self.bn2(self.conv2(out))
        out += identity
        out = self.relu(out)
        return out
```

2.2 Disease Grading Model

Alongside improved segmentation, we developed a separate disease grading network. This classifier processes retinal images to assess disease severity.

```python
class DiseaseGradingNet(nn.Module):
```

```
    def __init__(self, num_classes=3):
        super(DiseaseGradingNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu = nn.ReLU(inplace=True)
        self.pool = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.fc = nn.Linear(64 * 128 * 128, num_classes)

    def forward(self, x):
        x = self.pool(self.relu(self.bn1(self.conv1(x))))
        x = self.pool(self.relu(self.bn2(self.conv2(x))))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

2.3 Integrated Localization and Grading in U-Net

Building on our segmentation and grading advances, our new model integrates both localization and grading tasks. The U-Net now features two output branches:

- **Segmentation Head:** Delivers pixel-level lesion localization.
- **Grading Head:** Outputs disease grading based on features pooled from the bottleneck.

```python
class UNetLocalizationGrading(nn.Module):
    def __init__(self, in_channels=3, seg_channels=5, num_grading_classes=3, drop_prob=0.1):
```

```python
super(UNetLocalizationGrading, self).__init__()

# Encoder

self.enc1 = ResidualDoubleConv(in_channels, 64, drop_prob)

self.pool1 = nn.MaxPool2d(2)

self.enc2 = ResidualDoubleConv(64, 128, drop_prob)

self.pool2 = nn.MaxPool2d(2)

self.enc3 = ResidualDoubleConv(128, 256, drop_prob)

self.pool3 = nn.MaxPool2d(2)

self.enc4 = ResidualDoubleConv(256, 512, drop_prob)

self.pool4 = nn.MaxPool2d(2)


# Bottleneck

self.bottleneck = ResidualDoubleConv(512, 1024, drop_prob)


# Decoder for segmentation

self.up4 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)

self.dec4 = ResidualDoubleConv(1024, 512, drop_prob)

self.up3 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)

self.dec3 = ResidualDoubleConv(512, 256, drop_prob)

self.up2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)

self.dec2 = ResidualDoubleConv(256, 128, drop_prob)

self.up1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)

self.dec1 = ResidualDoubleConv(128, 64, drop_prob)

self.out_seg = nn.Conv2d(64, seg_channels, kernel_size=1)


# Classification head for grading

self.global_pool = nn.AdaptiveAvgPool2d((1, 1))
```

```python
        self.fc_grading = nn.Linear(1024, num_grading_classes)


    def forward(self, x):
        enc1 = self.enc1(x)
        enc2 = self.enc2(self.pool1(enc1))
        enc3 = self.enc3(self.pool2(enc2))
        enc4 = self.enc4(self.pool3(enc3))
        bottleneck = self.bottleneck(self.pool4(enc4))


        # Decoder branch for segmentation
        d4 = self.up4(bottleneck)
        d4 = torch.cat([d4, enc4], dim=1)
        d4 = self.dec4(d4)
        d3 = self.up3(d4)
        d3 = torch.cat([d3, enc3], dim=1)
        d3 = self.dec3(d3)
        d2 = self.up2(d3)
        d2 = torch.cat([d2, enc2], dim=1)
        d2 = self.dec2(d2)
        d1 = self.up1(d2)
        d1 = torch.cat([d1, enc1], dim=1)
        d1 = self.dec1(d1)
        seg_out = self.out_seg(d1)


        # Classification branch for grading
        pooled = self.global_pool(bottleneck)
        pooled = pooled.view(pooled.size(0), -1)
```

```
        grading_out = self.fc_grading(pooled)

        return seg_out, grading_out
```

3. Loss Functions

3.1 Focal Loss for Segmentation

To specifically mitigate class imbalance and improve the detection of difficult pixels, we implemented the Focal Loss. The following code snippet illustrates our implementation:

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


class FocalLoss(nn.Module):

    def __init__(self, alpha=1, gamma=2, logits=True, reduction='mean'):

        super(FocalLoss, self).__init__()

        self.alpha = alpha

        self.gamma = gamma

        self.logits = logits

        self.reduction = reduction


    def forward(self, inputs, targets):

        if self.logits:

            BCE_loss = F.binary_cross_entropy_with_logits(inputs, targets, reduction='none')

        else:

            BCE_loss = F.binary_cross_entropy(inputs, targets, reduction='none')
```

```python
        pt = torch.exp(-BCE_loss)
        focal_loss = self.alpha * (1 - pt) ** self.gamma * BCE_loss

        if self.reduction == 'mean':
            return focal_loss.mean()
        else:
            return focal_loss.sum()
```

3.2 Multi-Task Loss for Integrated Model

For our new integrated model, we plan to combine segmentation and grading losses. One straightforward approach is to use a weighted sum of the focal loss (for segmentation) and cross-entropy loss (for grading):

```python
def combined_loss(seg_pred, seg_target, grade_pred, grade_target, focal_loss_fn, ce_weight=1.0, focal_weight=1.0):
    seg_loss = focal_loss_fn(seg_pred, seg_target)
    grade_loss = F.cross_entropy(grade_pred, grade_target)
    return focal_weight * seg_loss + ce_weight * grade_loss
```

4. Training & Evaluation

Data Preprocessing:

- Images resized to 512×512
- Normalization and contrast enhancement (CLAHE applied on the L channel)

Dataloader Setup:

- Batch size: 4
- Shuffle: True
- Number of workers: 2

Training Loop Snippet (for Segmentation with Focal Loss):

```python
model = UNetLocalizationGrading()  # For integrated segmentation and grading

focal_loss_fn = FocalLoss(alpha=1, gamma=2, logits=True)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

num_epochs = 5



for epoch in range(num_epochs):

    model.train()

    for images, seg_masks, grade_labels in train_loader:

        seg_preds, grade_preds = model(images)

        loss = combined_loss(seg_preds, seg_masks, grade_preds, grade_labels, focal_loss_fn)

        optimizer.zero_grad()

        loss.backward()

        optimizer.step()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
```

Evaluation Metrics:

- **Dice Coefficient & IoU:** For segmentation quality
- **Accuracy & Cross-Entropy:** For disease grading performance

A typical evaluation function computes the final segmentation Dice, IoU, and grading accuracy.

5. Results

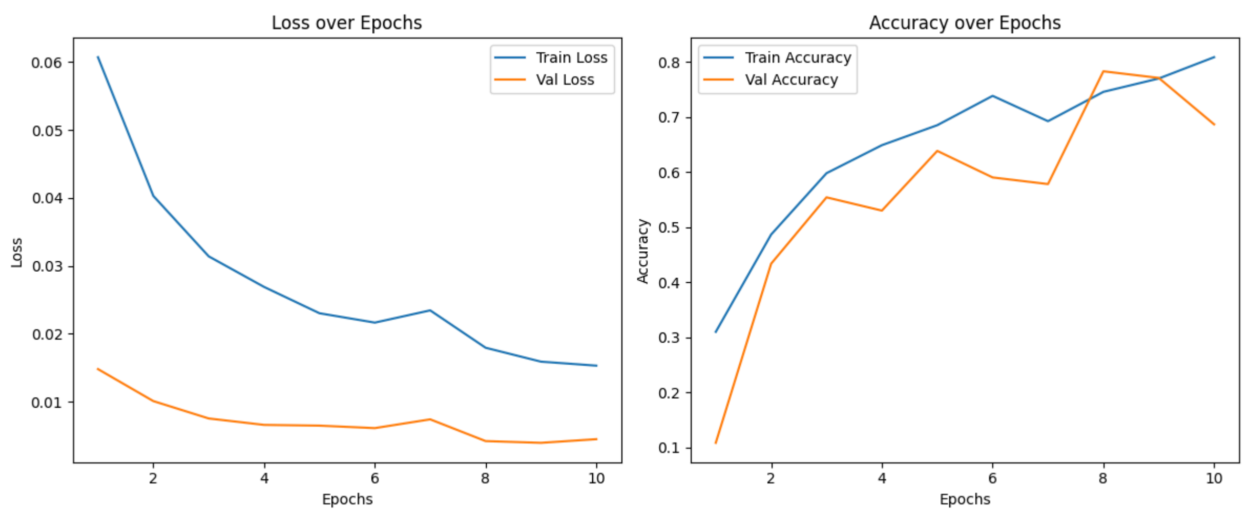
The experimental outcomes this week are promising:

- **Segmentation:** By using the Focal Loss, the segmentation network achieved remarkably higher IoU and Dice scores—especially for challenging pixels.

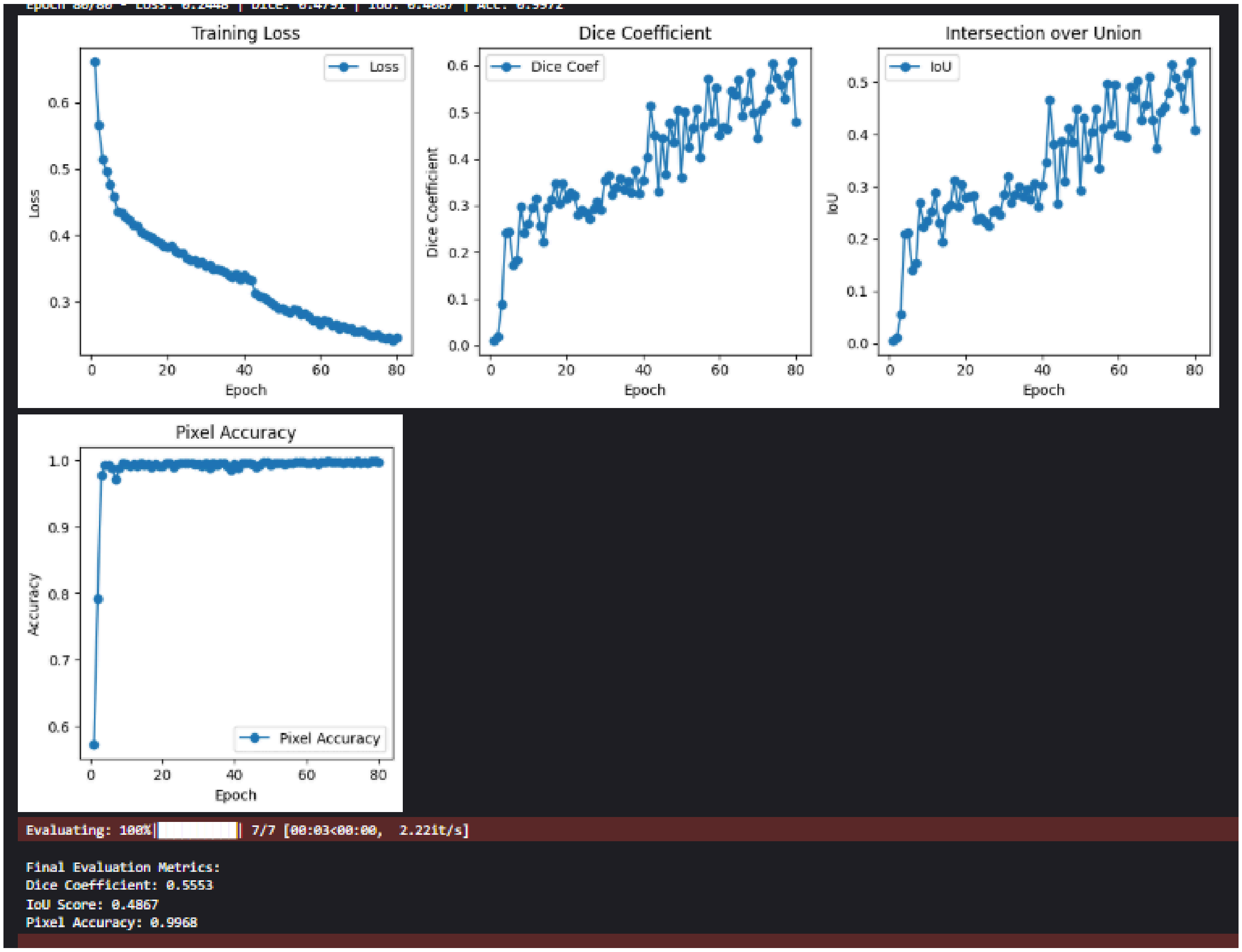
- **Disease Grading:** The standalone grading model produced robust classification performance on disease severity.

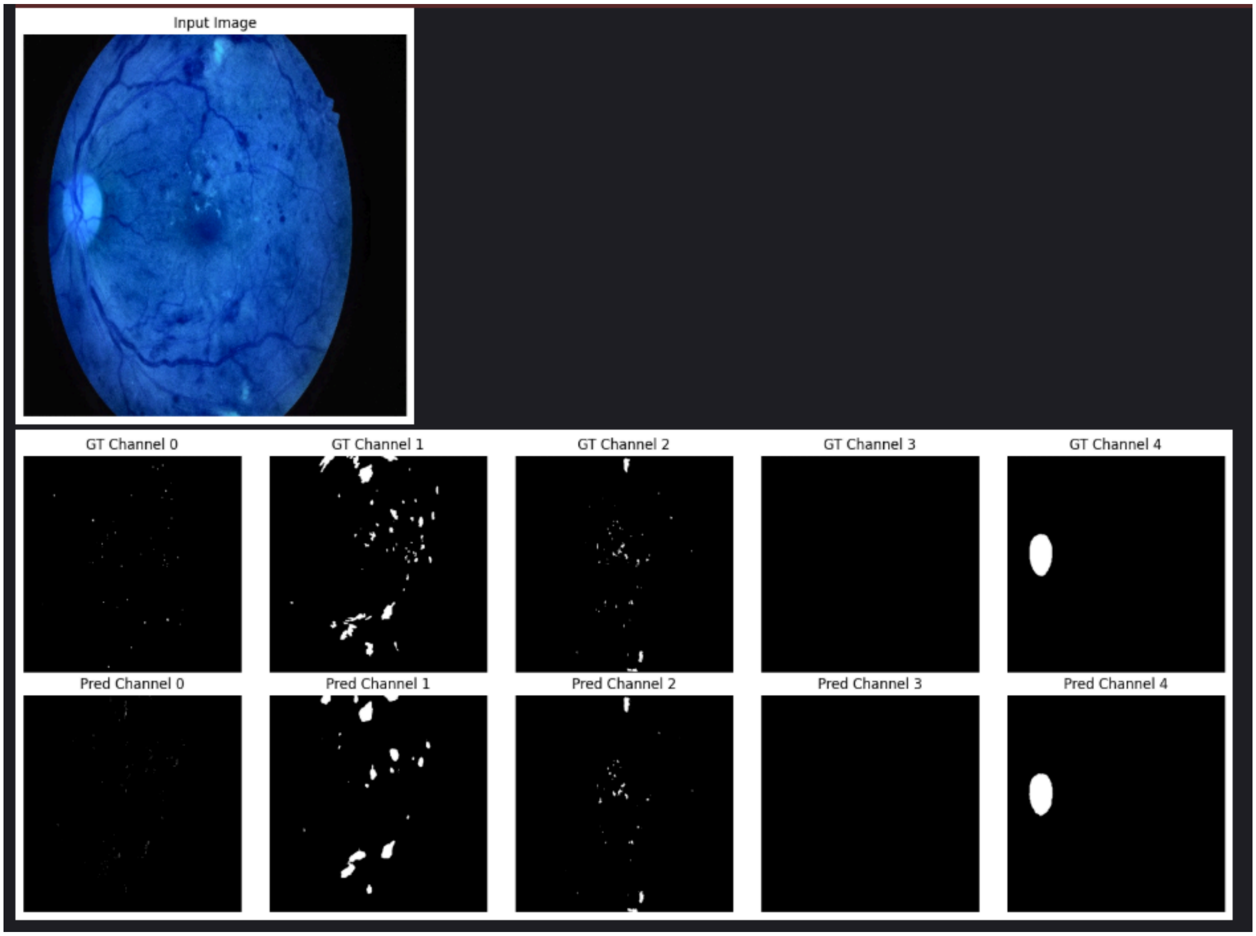
- **Overall:** Early experiments with the integrated model indicate that combining localization and grading tasks can streamline further improvements while preserving critical spatial information.

Disease Grading outcome:



Segmentation with Focal Loss:





6. Observations & Future Work

- **Segmentation:** Focal Loss has significantly improved handling of hard-to-classify pixels. Future work may explore varying the alpha and gamma parameters to further optimize performance.

- **Grading:** While the grading model shows good initial accuracy, incorporating more diverse data and fine-tuning the network may further enhance predictions.

- **Integration:** We will continue to refine our integrated U-Net model. Plans include exploring multi-scale feature aggregation and more sophisticated attention mechanisms to better fuse localization and grading information.

- **Post-Processing:** Techniques such as Conditional Random Fields (CRFs) remain on the agenda to sharpen segmentation boundaries.

WORK TO BE DONE NEXT WEEK

1. Implementing Focal loss.
2. Finding probable solutions for the leaky dataset and class imbalance.