**Report on Medical Image Segmentation Using Deep Learning**

**1. Introduction**

The goal of this project was to train a **deep learning model for medical image segmentation** using **U-Net with a ResNet-34 encoder**. The model was trained to segment anatomical structures from grayscale medical images, using a dataset where segmentation masks were provided in **Run-Length Encoding (RLE)** format.

We applied **data augmentation, loss function tuning, learning rate scheduling, and early stopping** to optimize training performance. This report details the **dataset preparation, training process, model performance, challenges, and results**.

**2. Dataset Preparation**

**2.1 Data Loading**

* The dataset consisted of grayscale medical images in **TIFF format (.tif)**.
* Masks were provided in **RLE format**, requiring conversion into binary masks.

**2.2 RLE Encoding & Decoding**

* We used the rle\_to\_mask() function to **convert RLE to binary masks**.
* Similarly, mask\_to\_rle() was used to **convert predicted masks back to RLE format** for submission.

**2.3 Data Augmentation**

To **improve generalization and prevent overfitting**, **Albumentations library** was used with:

* **Resizing** → (448, 608) (to match U-Net’s expected input dimensions)
* **Elastic Transform** → Simulate tissue deformations.
* **Gaussian Blur** → Enhance model robustness to noise.
* **Horizontal Flip** → To handle orientation variations.
* **Random Brightness & Contrast** → Mimic lighting differences.
* **Coarse Dropout** → Simulate missing data in the images.

**Effect:** Augmentation increased dataset variability, improving model robustness.

**3. Model Architecture & Training**

**3.1 Model Choice: U-Net with ResNet-34 Encoder**

* **Encoder:** ResNet-34 (pretrained on ImageNet) for feature extraction.
* **Decoder:** U-Net architecture to upsample feature maps to original resolution.
* **Activation Function:** Sigmoid (for binary segmentation).
* **Dropout:** 30% to reduce overfitting.

python

CopyEdit

model = smp.Unet(

encoder\_name="resnet34",

encoder\_weights="imagenet",

in\_channels=1, # Grayscale images

classes=1, # Binary segmentation

activation="sigmoid",

decoder\_dropout=0.3, # Dropout to prevent overfitting

)

**3.2 Loss Function**

A **hybrid loss function** was used:

* **Dice Loss** (good for segmentation accuracy)
* **Binary Cross Entropy (BCE) Loss** (stabilizes learning)

python

CopyEdit

def combined\_loss(y\_pred, y\_true):

return 0.5 \* dice\_loss(y\_pred, y\_true) + 0.5 \* bce\_loss(y\_pred, y\_true)

**3.3 Optimizer & Learning Rate Scheduler**

* **Optimizer:** AdamW with weight decay (1e-4) to prevent overfitting.
* **Scheduler:** ReduceLROnPlateau to **halve the learning rate** if validation loss stagnates.

python

CopyEdit

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4, weight\_decay=1e-4)

scheduler = torch.optim.lr\_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=5)

**3.4 Training & Validation Strategy**

* **80% Training - 20% Validation split**
* **Batch Size:** 32
* **Early Stopping:** Training stops if validation loss does not improve for 10 epochs.

python

CopyEdit

early\_stopping\_patience = 10 # Stop if no improvement for 10 epochs

**4. Model Performance**

**4.1 Training Loss vs Validation Loss**

| **Epoch** | **Train Loss** | **Validation Loss** | **Best Model Saved?** |
| --- | --- | --- | --- |
| 0 | **0.9106** | **0.8703** | ✅ |
| 5 | **0.6543** | **0.6414** | ✅ |
| 10 | **0.5060** | **0.5915** | ⚠️ No improvement |
| 15 | **0.4671** | **0.5654** | ⚠️ No improvement |
| 20 | **0.4436** | **0.5550** | ✅ |
| 30 | **0.4157** | **0.5716** | ❌ Early Stopping |

**4.2 Findings**

* **Train loss improved consistently** over 30 epochs.
* **Validation loss stopped improving** after ~20 epochs.
* **Early stopping** was triggered to prevent overfitting.

**4.3 Possible Issues**

1. **Validation loss stagnated around 0.55** → Potential data leakage or overfitting.
2. **Predicted masks were fully white (1 243600 for all images)**:
   * Overfitting to training set (model classifying everything as foreground).
   * Poor thresholding before converting to RLE.

**5. Debugging & Manual Verification**

**5.1 Visual Inspection**

We manually **plotted the predicted masks over the original images** using plt.imshow().

python

CopyEdit

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

ax[0].imshow(image, cmap="gray")

ax[0].set\_title("Original Image")

ax[1].imshow(mask, cmap="gray")

ax[1].set\_title("Predicted Mask")

plt.show()

**5.2 Fixing the "All White" Prediction Issue**

**Cause:** The model’s **threshold was too low** (0.5), leading to many false positives.

✅ **Fix:** Adjusted threshold in post-processing:

python

CopyEdit

mask\_resized = mask\_resized > 0.6 # Increased threshold to reduce false positives

**6. Final Submission Generation**

Once the model was corrected, we generated predictions for the test set and saved the submission file.

**Key Steps**

1. **Resize images to (448, 608) before inference.**
2. **Apply threshold (0.6) to reduce false positives.**
3. **Resize masks back to (420, 580) after inference.**
4. **Convert binary masks to RLE format for submission.**

python

CopyEdit

def mask\_to\_rle(mask):

pixels = mask.flatten(order='F')

pixels = np.concatenate([[0], pixels, [0]])

runs = np.where(pixels[1:] != pixels[:-1])[0] + 1

runs[1::2] -= runs[::2]

return " ".join(str(x) for x in runs) if len(runs) > 0 else ""

**7. Conclusion & Next Steps**

**✅ Achievements**

* **Trained a U-Net model** using ResNet-34 for medical image segmentation.
* **Applied advanced data augmentation** to improve generalization.
* **Implemented hybrid loss (Dice + BCE)** for better segmentation accuracy.
* **Used ReduceLROnPlateau & early stopping** for efficient training.
* **Manually validated predictions** to detect issues early.
* **Fixed thresholding problems** to ensure valid RLE submissions.

**❌ Challenges**

1. **Validation loss stagnation** → Requires more regularization or larger dataset.
2. **Overfitting to training data** → Could try dropout, weight decay tuning.
3. **Poor thresholding** → Required manual tuning to prevent "all-white" masks.

**🚀 Future Work**

* **Test different U-Net architectures** (EfficientNet, ResNet-50, etc.).
* **Experiment with different loss functions** (Tversky loss, Lovasz loss).
* **Try Test Time Augmentation (TTA)** to improve generalization.
* **Fine-tune thresholding dynamically** based on dataset characteristics.

**🔹 Final Thoughts**

This project demonstrated **deep learning-based medical image segmentation** using U-Net. The model performed well on the training set but required **manual debugging** to fix submission errors. **Future work** can focus on **better regularization, dynamic thresholding, and improved architectures** to enhance segmentation quality.

🔥 **This was a great learning experience in deep learning, segmentation, and Kaggle-style competitions!** 🚀