Step 1: Data

1. **Total number of data samples in the dataset**: There are 100 data samples included in the dataset.
2. **Problem the dataset addresses**: This dataset contains a comprehensive collection of retinal fundus images, meticulously annotated for blood vessel segmentation. Accurate segmentation of blood vessels is a critical task in ophthalmology as it aids in the early detection and management of various retinal pathologies, such as diabetic retinopathy and macular degeneration.
3. **Dimension range in the dataset**: All images have dimensions of 512x512 pixels. This is the raw dimension before any resizing is done during preprocessing.
4. **Missing information in the dataset**: There are no missing mask files in the dataset. This implies that for each image, there is a corresponding segmentation mask, and hence, no missing labels.
5. **Label of the dataset**: The labels are the segmentation masks corresponding to each image, and they take on values from 0 to 255, where 0 represents the background and values from 1 to 255 represent different objects or features to be segmented.
6. **Data split percentages**: Based on the hint, the data will be split into 70% for training, 15% for validation, and 15% for testing. This split is typical for ensuring that the model is trained on most of the data while still having enough unseen data to validate and test the model's generalizability.
7. **Data preprocessing used for the training dataset**: As per the code, the following preprocessing steps are applied:
    - The images and masks are normalized to have values in the range [0, 1]. This is done by casting the pixel values to a floating-point type and dividing them by 255.
    - The images and masks are resized to the target dimensions of 256x256 pixels. This step is necessary because the U-Net model requires a fixed input size.
    - Additionally, from the structure of the code (although not explicitly included in the preprocessing), it is likely that there would be shuffling of the data before training, and data augmentation could be added to improve the robustness of the model to variations in input images. However, these last two steps are not directly observable from the given code snippet and would typically be included in a complete training pipeline.

Step 2: Model

Model	IoU	Dice
U-Net 2 layers	0.58	0.41
U-Net 3 layers	0.64	0.47
U-Net 4 layers	0.67	0.51

Step 3: Objective

I am using “binary_crossentropy” as my loss function to train the model.

Step 4: Optimization

I have used the Adam optimizer. Here are some reasons for using Adam in this specific context:

1. **Adaptive Learning Rates**: Adam maintains a per-parameter learning rate that improves performance on problems with sparse gradients (like image data) or with noisy or different scales of parameters. This is particularly useful in image segmentation tasks, where different features may be present with varying frequencies.
2. **Efficiency**: Adam is computationally efficient, which is a considerable advantage when training large models like U-Net, which may have a substantial number of parameters.
3. **Little Memory Requirements**: Adam does not require a large amount of memory, which makes it suitable for applications like Google Colab, where the user may have limited memory resources.
4. **Suitable for Problems with Noisy/Non-stationary Objectives**: Image segmentation tasks often deal with a variety of input data, and the objective function landscape can be quite complex and noisy. Adam is robust to this type of objective landscape.
5. **Hyperparameters Have Intuitive Interpretation**: The default values for Adam's hyperparameters (learning rate, first and second moment decay rates) are generally recommended as good starting points, and in many cases, the default values work well, which reduces the need for extensive hyperparameter tuning.

Step 5: Model Selection

Model	Dice & IoU with normalization	Dice & IoU without normalization
U-Net 2 layers	Dice 0.41, Iou 0.58	Dice 0.71, Iou 0.55
U-Net 3 layers	Dice 0.47, Iou 0.64	Dice 0.67, Iou 0.51
U-Net 4 layers	Dice 0.51, Iou 0.67	Dice 0.60, Iou 0.42

Considering the normalization aspect:

1. **With Normalization:**
    - U-Net with **4 layers** has the highest Dice coefficient (0.51) and IoU (0.67), which means it has the best overlap between the predicted and actual segmentation masks after normalization.

2. **Without Normalization:**

    - The U-Net with **2 layers** has a Dice coefficient of 0.71 and an IoU of 0.55. Despite having a lower IoU compared to the 4-layer model with normalization, it has a higher Dice coefficient, indicating a very good overlap but potentially less consistency in the predictions (since IoU is lower).

The **U-Net with 4 layers and normalization** appears to perform the best overall, as it has the highest Dice and IoU scores among all models when normalization is applied. Normalization typically helps neural networks learn more efficiently and effectively, and the results here suggest that with normalization, the additional complexity of the 4-layer network allows it to learn better representations of the data, leading to a higher quality of segmentation.

Several strategies were used to mitigate overfitting and underfitting when training the U-Net model:

1. **Dropout Layers**: Dropout is a regularization technique where randomly selected neurons are ignored during training, which helps in preventing the model from becoming too dependent on any one feature. This is evident in the **build_unet** function where **Dropout(0.5)** is applied, meaning 50% of the nodes will be dropped at random during training, reducing overfitting.
2. **Data Augmentation**: While not directly shown in the code snippet you provided, data augmentation is a common technique to generate more training data from the existing samples by applying random transformations like rotation, scaling, and flipping. This helps the model generalize better, though it would need to be implemented separately in the data preprocessing pipeline.
3. **Batch Normalization**: This technique is used to normalize the input layer by adjusting and scaling the activations. It helps in speeding up the training as well as combating overfitting by adding a slight noise to each hidden layer's activations.
4. **Model Complexity**: The model complexity is adjusted according to the dataset size. A model with too many parameters might overfit a small dataset since it would learn the noise in the training set. Conversely, a model with too few parameters might underfit because it cannot capture the underlying pattern in the data. In the provided code, the depth of the U-Net is chosen to be suitable for the dataset size.
5. **Training/Validation Split**: The dataset is split into training and validation sets. During training, the model's performance on the validation set is monitored to ensure that it is learning general patterns rather than memorizing the training data. The code specifies a validation split of 0.1 (or 10%), which helps in detecting overfitting.
6. **Custom Loss Function**: The use of a combined loss function that incorporates binary cross-entropy and dice loss helps the model to not only penalize the wrong predictions but also to improve the overlap between the predicted and true masks (as the dice coefficient is a common metric for segmentation tasks).

7. **Performance Metrics**: By tracking performance metrics like the Dice coefficient and Intersection over Union (IoU), you can monitor if the model is learning the proper segmentation and generalizing well, rather than just reducing the loss on the training data.

Step 6: Model Performance

A. The Dice and IoU plots of the models I tried are included in the submitted notebook.
B. 4 input images with labels and their respective predicted masks are included in the submitted notebook.