**Assignment**

**Q1) How can the model achieve high accuracy when the available training images are insufficient? What strategies can be used to overcome this limitation?**

**Ans:** In This case, the model is showing high validation accuracy, like almost 0.999, even though I have trained on a small number of images — just around 30–40 samples. But this doesn’t mean the model is learning correctly. Most probably this high accuracy is misleading because the dataset is small and also might be imbalanced. In my ground truth masks, only few pixels are white (i.e., object area), and rest all are black. So even if the model predicts all black pixels (0s), still it will get high accuracy. This is a known issue with segmentation tasks when data is sparse or has low foreground content.

Still, if we want to genuinely improve performance with less data, there are few techniques that can help. First thing is data augmentation. In my model, I used ImageDataGenerator to apply flips, zoom, shift, etc. This creates synthetic variation in the training data. So the model sees rotated, shifted versions of same input, which helps in learning more robust features.

Second thing is using **transfer learning**. In my code I used a custom CNN built from scratch. But if I had used a pre-trained model like U-Net or even an encoder-decoder using ResNet or VGG pretrained on ImageNet, then it would have helped a lot because those models already learn good features from generic images, and we only fine-tune them for our use-case.

Also another approach is **patch-based training**. Instead of feeding whole 256x256 images, we can crop small patches like 64x64 where there is object presence, and train on that. That increases number of samples artificially.

Lastly, if the dataset is really small, techniques like **few-shot learning** or **semi-supervised learning** (e.g., using pseudo-labels on unlabeled data) can be explored. But in my current work, I have just used augmentation and resizing for data limitation.

**Q2) How can model overfitting and underfitting be addressed? What factors contribute to these issues?**

Ans: In my streak-star segmentation model, I faced signs of underfitting during training. The training accuracy stayed low (around 60–70%), while validation accuracy was very high (~0.999), which didn’t reflect real performance. This happens mostly because the mask data is sparse — meaning most pixels are 0 (background), so even wrong predictions can look accurate.

**Underfitting** happens when the model is too simple or not trained long enough. To fix that, I increased the number of epochs, adjusted the learning rate, and used BatchNormalization() to help the network learn better.

**Overfitting** would happen if the model starts memorising the small training dataset. To avoid that, I used data augmentation with ImageDataGenerator() (like flipping, shifting, rotating) so that the model learns more general patterns. Also, limiting model size (not too many layers or filters) helps prevent overfitting.

Another thing I kept in mind was splitting the data properly and using validation performance to check generalisation. If validation loss increases while training loss decreases, that’s a clear overfitting sign.

So, overall, I managed underfitting by improving model training and avoided overfitting through augmentation and balanced architecture.

**Q3) Can the developed model accurately detect streaks and stars in real-skyastronomical images, given varying signal-to-noise conditions (e.g., faint object detection)? How does this differ from simulated images?**

**Ans:** Detecting streaks and stars in real-sky images is definitely more challenging compared to the simulated ones I used. In my dataset, most images were clean and controlled, so the model learned to detect objects where contrast was clear. But in actual astronomical images, the noise level is high, objects can be faint, and background light might vary due to atmosphere, exposure, or telescope noise.

My model uses grayscale satellite images and binarized masks. It does decent segmentation on these, but real-sky images will have varying brightness and blurred or broken streaks. The model might fail to catch very faint stars or misclassify noise as objects.

Also, in real images, stars may overlap or be clustered. The current model uses a basic CNN and simple thresholding at 0.5, which might not be flexible enough to adapt to noise levels or fuzzy patterns. A more advanced approach like adaptive thresholding or postprocessing filters would help.

So, in short, while my model works okay on controlled inputs, for real-sky use it would need retraining with noise-heavy datasets and better postprocessing to improve faint object detection. The current one will struggle if directly applied to raw telescope data.

**Q4) What is the likelihood of false detections, such as detecting multiple blobs along a streak instead of a continuous line? Will the model correctly report false positives and generate an appropriate error matrix?**

**Ans:** Yes, there is a good chance that my model can produce false detections — especially in the form of detecting one long streak as multiple small blobs. This is because the model is trained on binary masks and uses a fixed threshold (0.5) to classify pixels. If the predicted mask has a broken line or discontinuity, the regionprops() function might detect each part separately.

Also, the model uses eccentricity to classify blobs. But sometimes a small part of a streak might still have low eccentricity and get wrongly labelled as star. These types of false positives are possible, especially when the prediction mask is faint or fragmented.

In terms of evaluation, my current code does not use a formal confusion matrix. It just prints test accuracy and visual output. To properly capture false positives and negatives, we would need to compute precision, recall, and F1 score on object-level (not pixel-wise), or use IoU (Intersection over Union) metrics for each detection.

So overall, the model can make false detections due to streak fragmentation, and it currently doesn’t quantify that in any matrix. For production use, we’ll need better postprocessing and evaluation methods to measure that.

**Algorithms Used:**

**U-Net**: Best for precise segmentation with limited data.

**Binary Cross-Entropy**: Ideal for binary masks.

**Connected Components**: Extracts objects/centroids from masks.

**Augmentation**: Boosts dataset diversity.