1. **Objective**: To develop a neural network model capable of segmenting stones from the

background in a given set of images.

1. **Dataset**: The dataset of 20 images ((Imagexx.jpg)) and it’s corresponding Image true

labels (Imagexx\_label.jpg) containing stones with background.

1. **Coding Language**: python
2. **Explaining the approach**:

Step-1 Importing all the necessary libraries.

1. os: The os module provides a way to interact with the operating system. In this code, it's used to handle file paths and directory operations, such as joining paths and listing directory contents.
2. NumPy(np): NumPy is a fundamental library for numerical computing in Python. In this code, NumPy is used for numerical operations and array manipulations, such as loading images into arrays and preprocessing data.
3. TensorFlow(tf): In this code, TensorFlow is used for loading and preprocessing images, defining the U-Net model architecture, training the model, making predictions, and evaluating model performance.
4. matplotlib.pyplot (plt): Matplotlib is a plotting library for creating static, interactive, and animated visualizations in Python. Here, it's used to plot sample images, actual masks, and predicted masks, helping us to evaluate the model's performance visually.
5. From tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Conv2DTranspose, concatenate: Layer classes provided by Keras for constructing the U-Net model architecture. It's used here to define the convolutional layers, pooling layers, transposed convolutional layers, and concatenation operations necessary for the U-Net's encoder-decoder structure.
6. from tensorflow.keras.models import Model: It is used to instantiate the Keras model object, here it represents the entire U-Net architecture, including its inputs, outputs, and layers.
7. train\_test\_split (from sklearn.model\_selection): The train\_test\_split function from sklearn.model\_selection is used in the code to split the dataset into training and validation sets.
8. tensorflow.keras.metrics.BinaryIoU: This imports the Binary Intersection over Union (IoU) metric from the TensorFlow Keras API. IoU is a common evaluation metric for semantic segmentation tasks, measuring the overlap between predicted and ground truth masks.

Version of the above libraries:

1. numpy version= 1.26.4
2. tensorflow version= 2.15.0
3. matplotlib version= 3.8.2
4. scikit-learn version= 1.4.1.post1

Step2- Defining the path to the directory containing image dataset.

Step3- Data Loading and Preprocessing:

Images and their corresponding masks are loaded and pre-processed using the load\_data function. Images are resized to a target size of 256x256 pixels, then images are loaded using tf.keras.preprocessing.image.load\_img and converted to arrays using tf.keras.preprocessing.image.img\_to\_array. Masks are loaded in grayscale mode and normalizing their pixel values by dividing by 255. Data is splitted into training and validation sets using train\_test\_split.

Step4- Defining the model architecture:

The U-Net model architecture is defined using the unet\_model function, which constructs a convolutional neural network (CNN) consisting of convolutional, max-pooling, convolutional transpose (deconvolution), and concatenation layers.

The model's input shape is specified as (256,256, 3), representing the image dimensions and the number of channels (RGB).

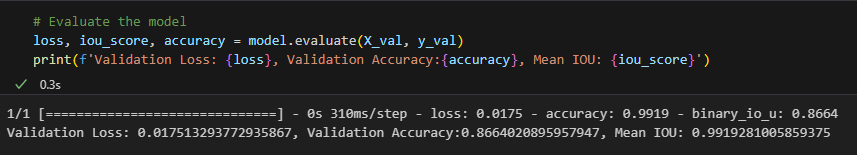
* 1. Model Definition: Defines a function named unet\_model that constructs the architecture of a U-Net model.
  2. Input Layer: Creates an input layer with the specified input\_shape.
  3. Encoder Section:
  4. Convolutional Blocks: Utilizes a series of convolutional layers (Conv2D) with increasing filter sizes (32, 64, 128, 256, 512) and ReLU activation function. Each block is followed by another convolutional layer with increasing filter size and activation function.
  5. Max Pooling: Incorporates max-pooling layers (MaxPooling2D) after each pair of convolutional layers to down sample the feature maps.
  6. Decoder Section:
  7. Upsampling and Concatenation: Transpose convolutional layers (Conv2DTranspose) to upsample the feature maps, followed by concatenation with the corresponding feature maps from the encoder section.
  8. Convolutional Blocks: Utilizes convolutional layers with decreasing filter sizes (256, 128, 64, 32) and ReLU activation function. Each block is followed by another convolutional layer with decreasing filter size and activation function.
  9. Output Layer: Generates the final output using a convolutional layer with a single filter and sigmoid activation function.
  10. Return Statement: Returns the constructed model.

Step5- Model compilation and Training: The model is compiled with the Adam optimizer and binary cross-entropy loss function. Additionally, BinaryIoU (Intersection over Union) metric is used to evaluate the model's performance.

The model is trained using the training data (X\_train and y\_train) with validation data (X\_val and y\_val) for monitoring performance. Training is performed for 20 epochs with a batch size of 16.

Step6- After training, the model is evaluated on the validation set to obtain loss, accuracy and Intersection over Union - IoU Evaluation Metric.

The Intersection over Union (IoU) score is calculated to measure the overlap between predicted and ground truth masks.



Step7- Predictions and Visualizations: Then, the trained model is used to make predictions on the entire set of images.

Plotting predictions:

* 1. The code creates a set of subplots to display original images, actual masks, and predicted masks.
  2. It iterates over a range of indices and displays the corresponding images and masks.
  3. Each subplot consists of three images: the original image, the actual mask, and the predicted mask.

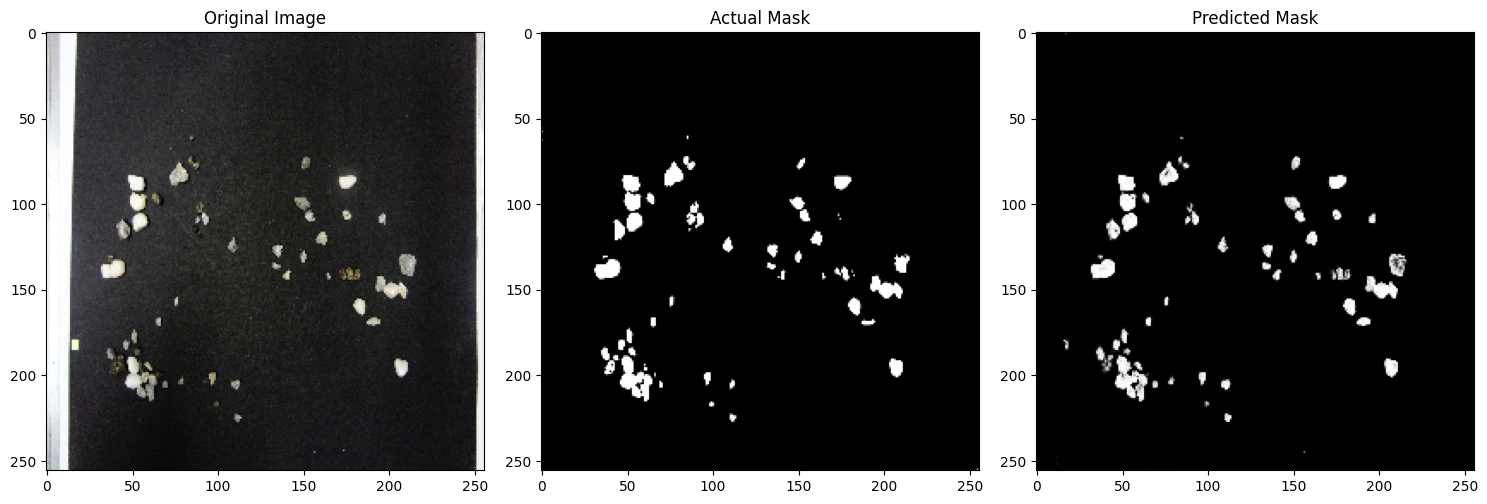


Fig-1

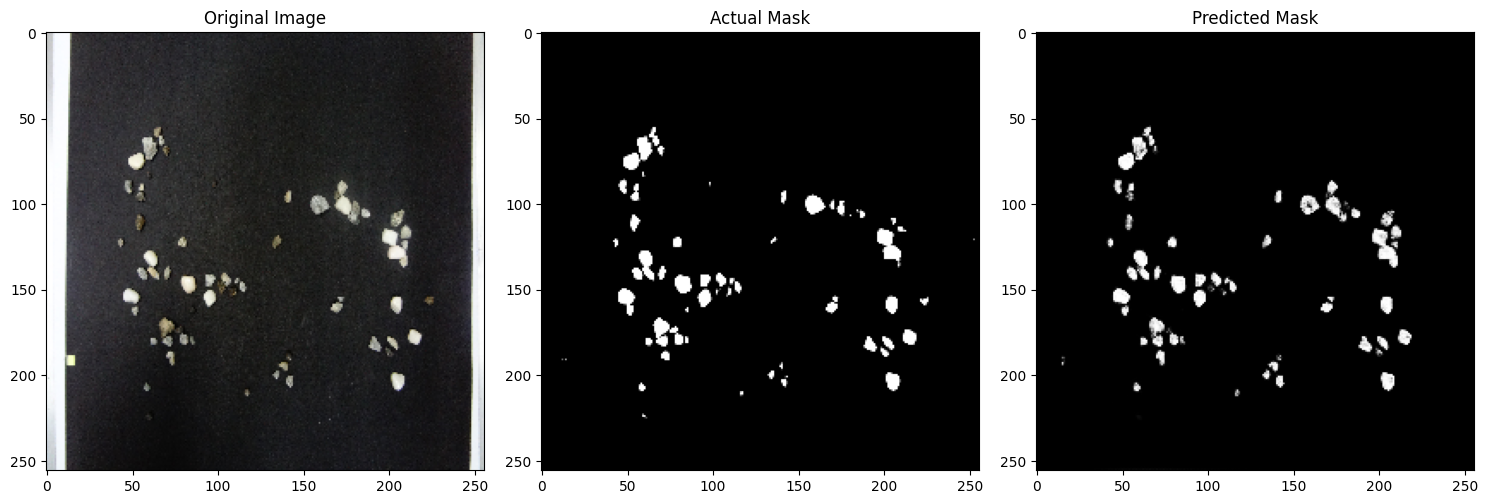


Fig-2

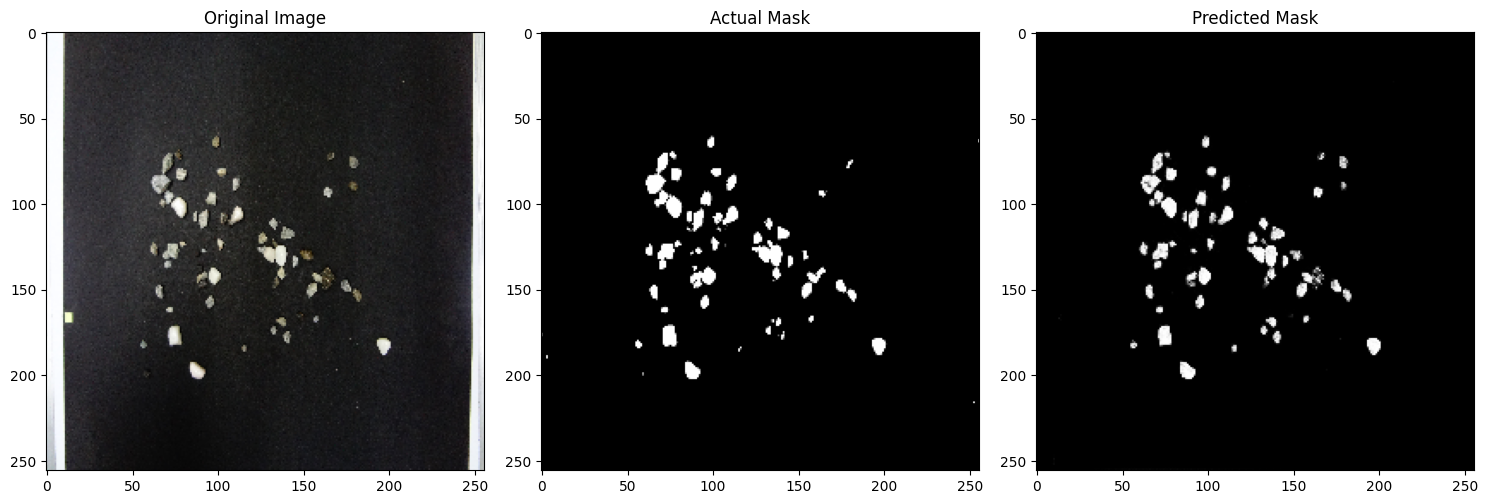


Fig-3

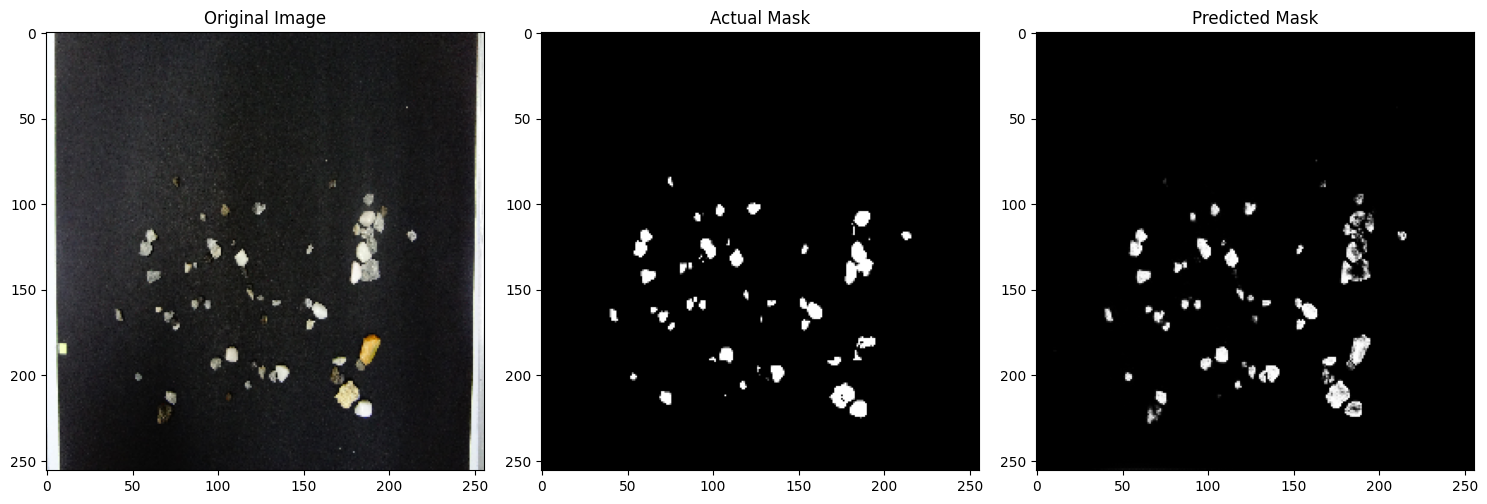


Fig-4

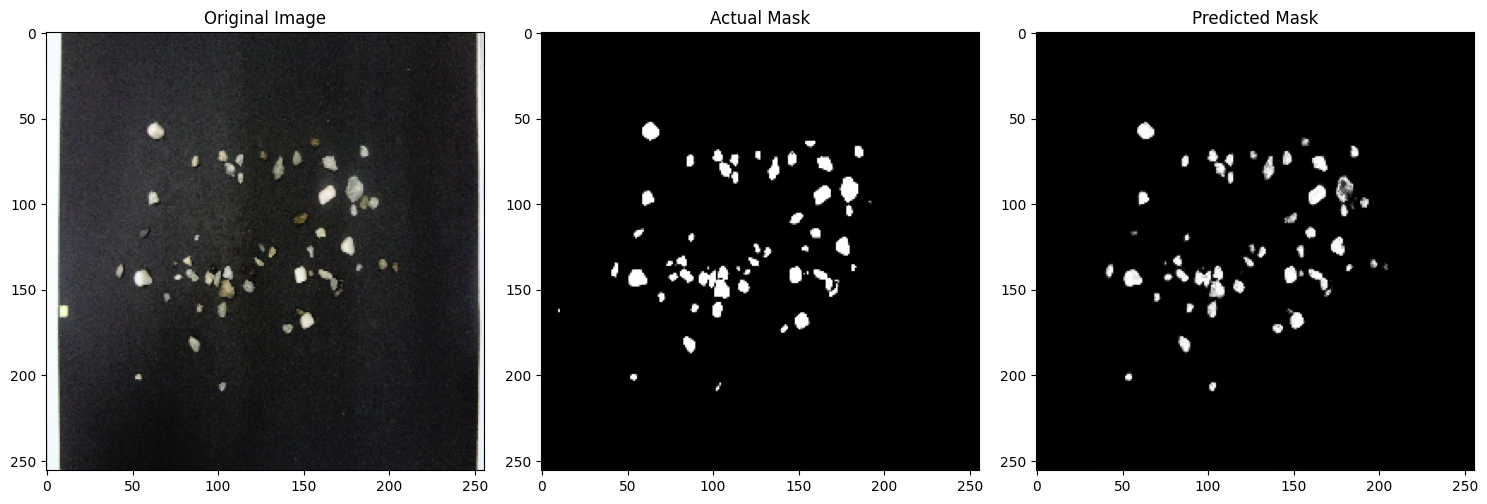


Fig-5

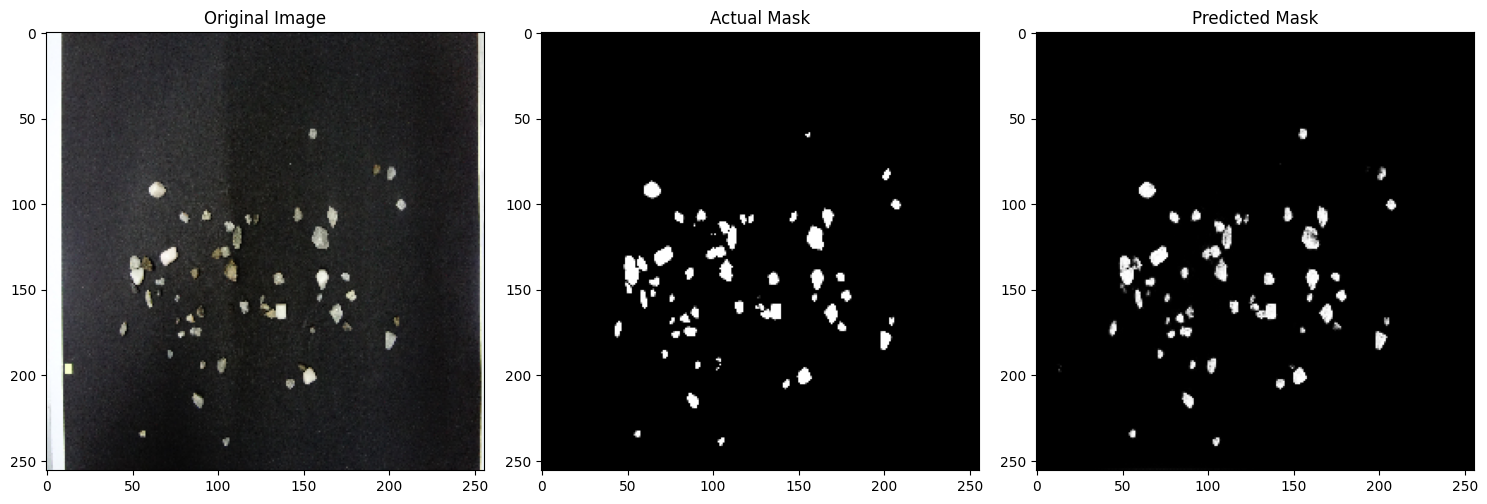


Fig-6

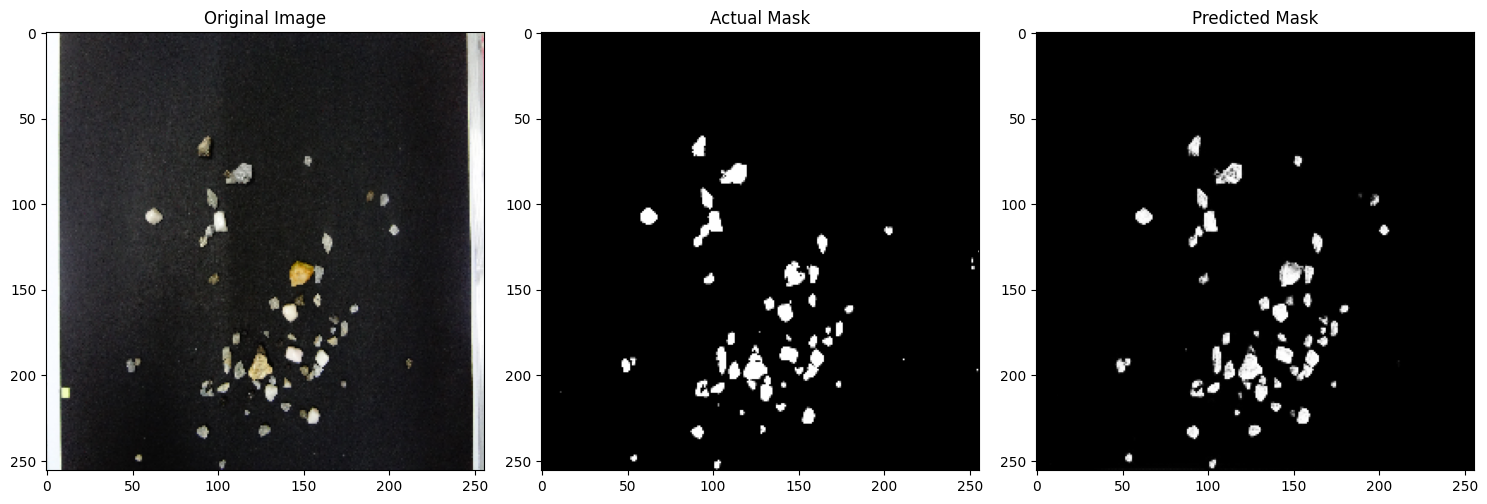


Fig-7



Fig-8

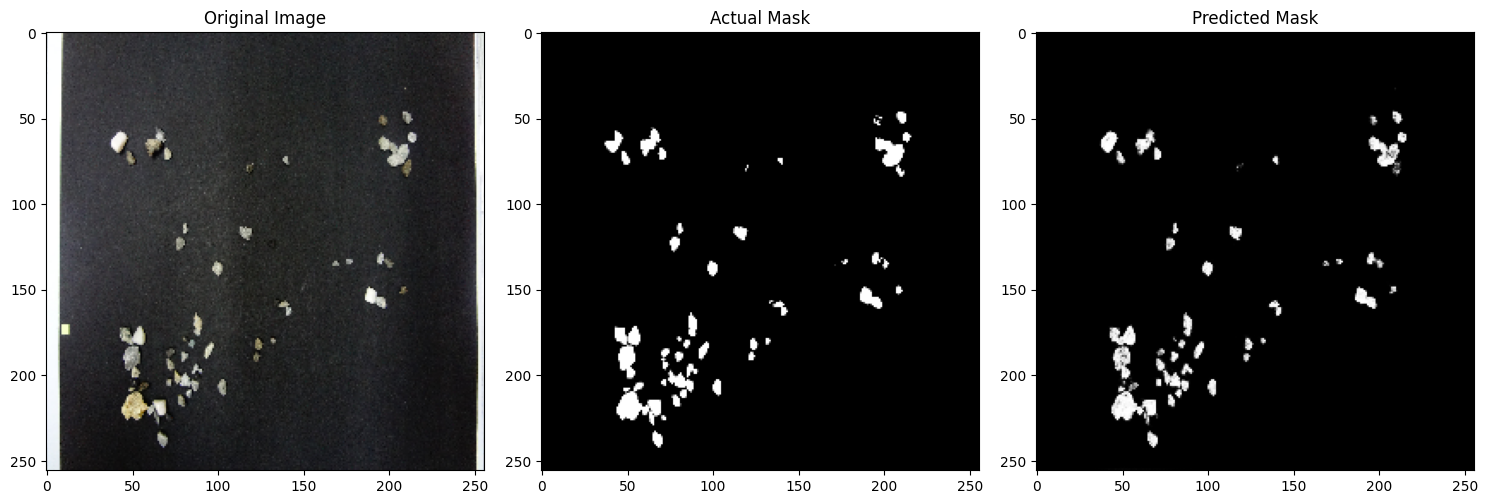


Fig-9

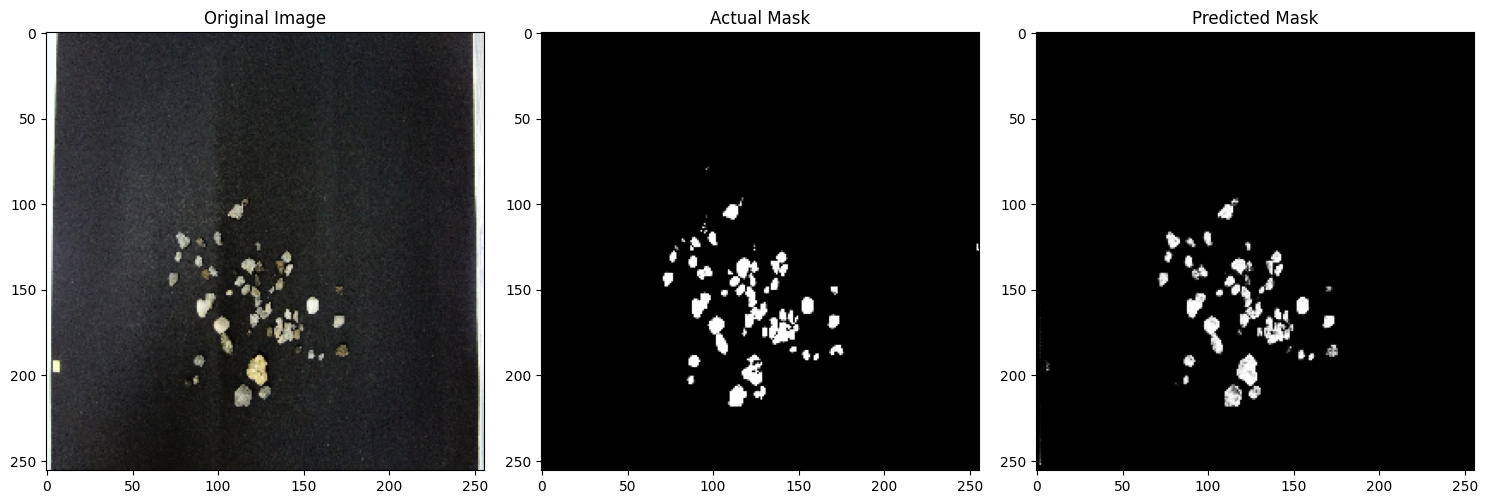


Fig-10

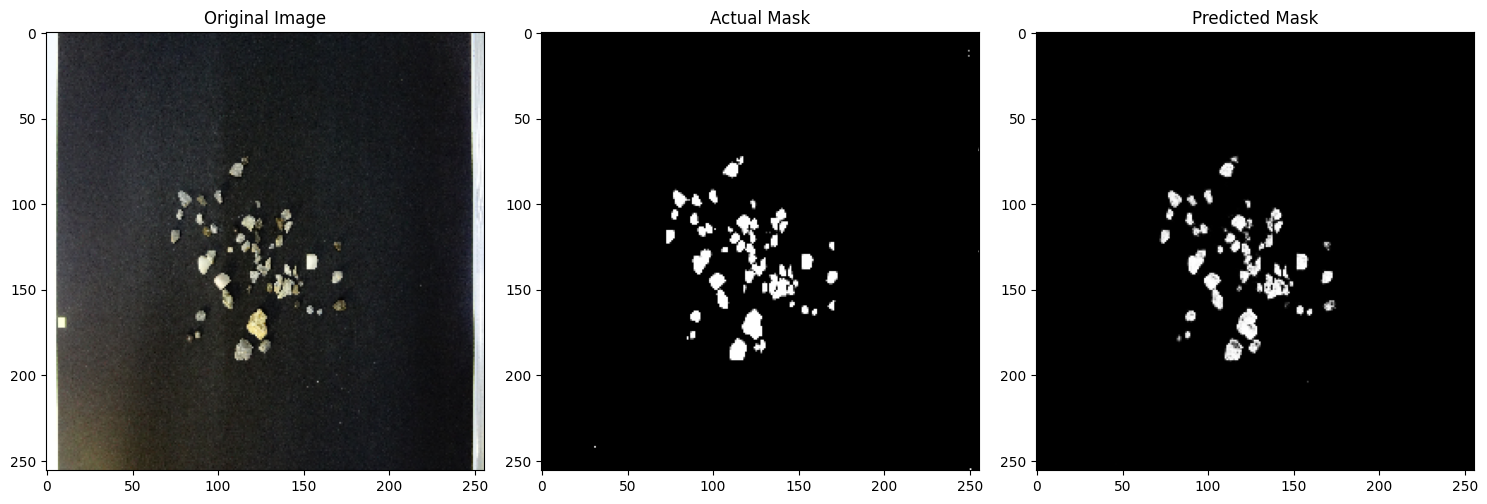


Fig-11

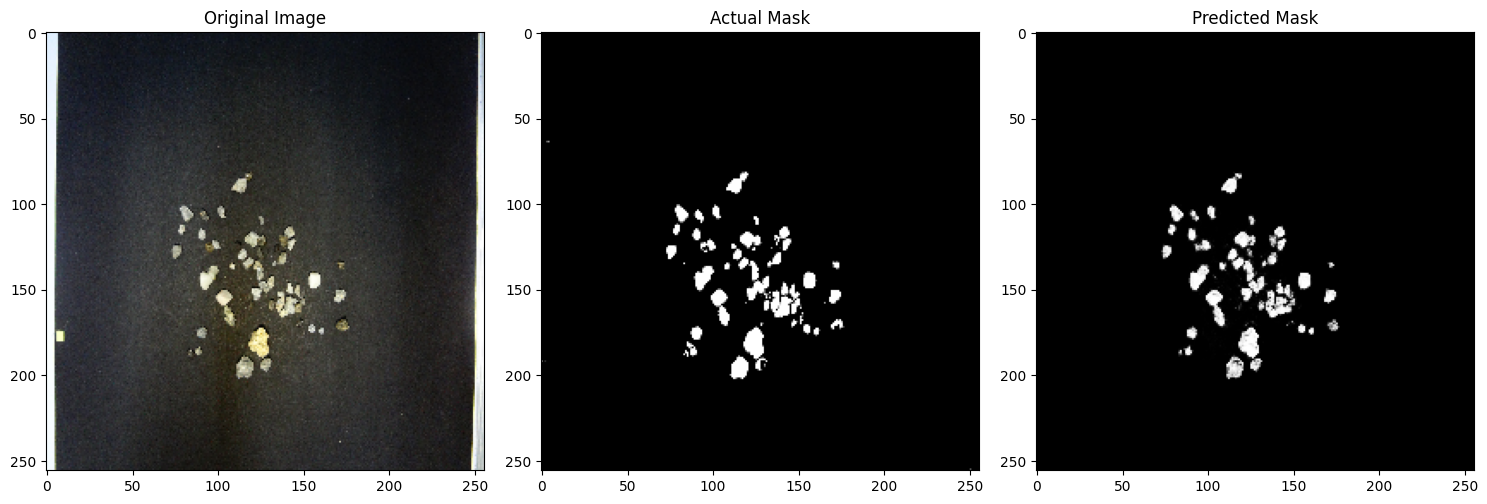


Fig-12

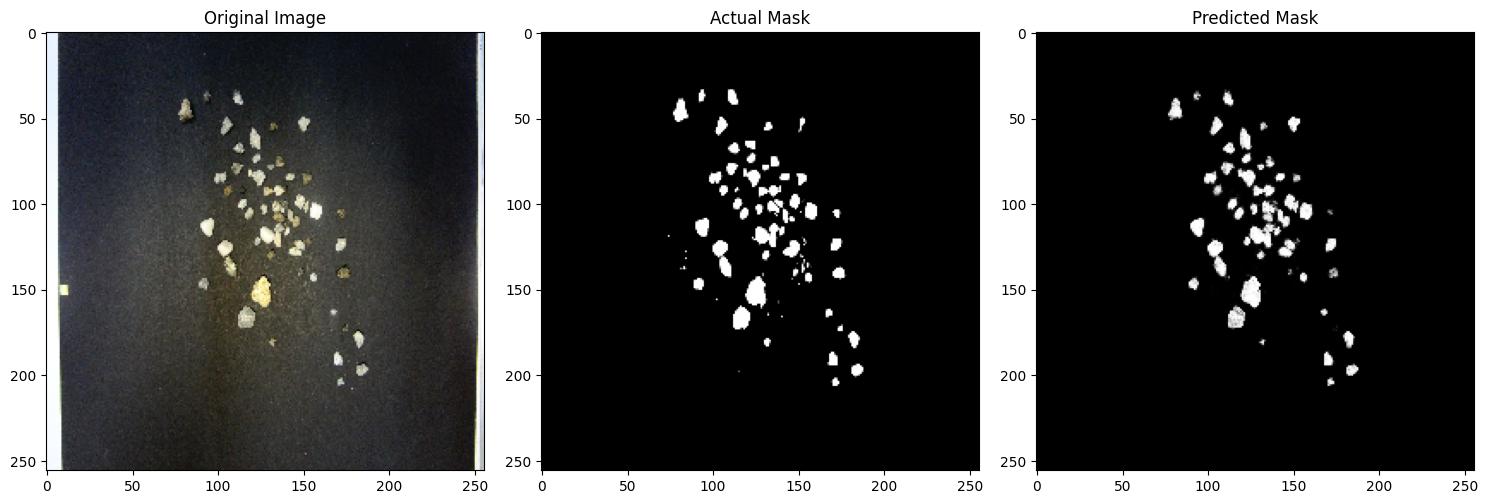


Fig-13

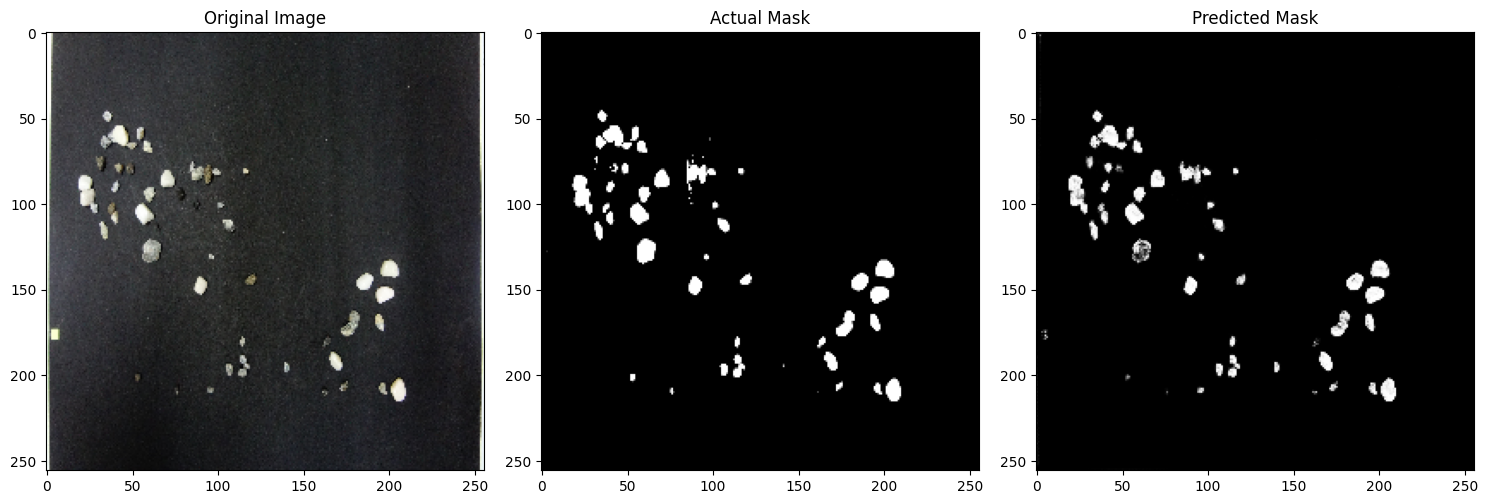


Fig-14

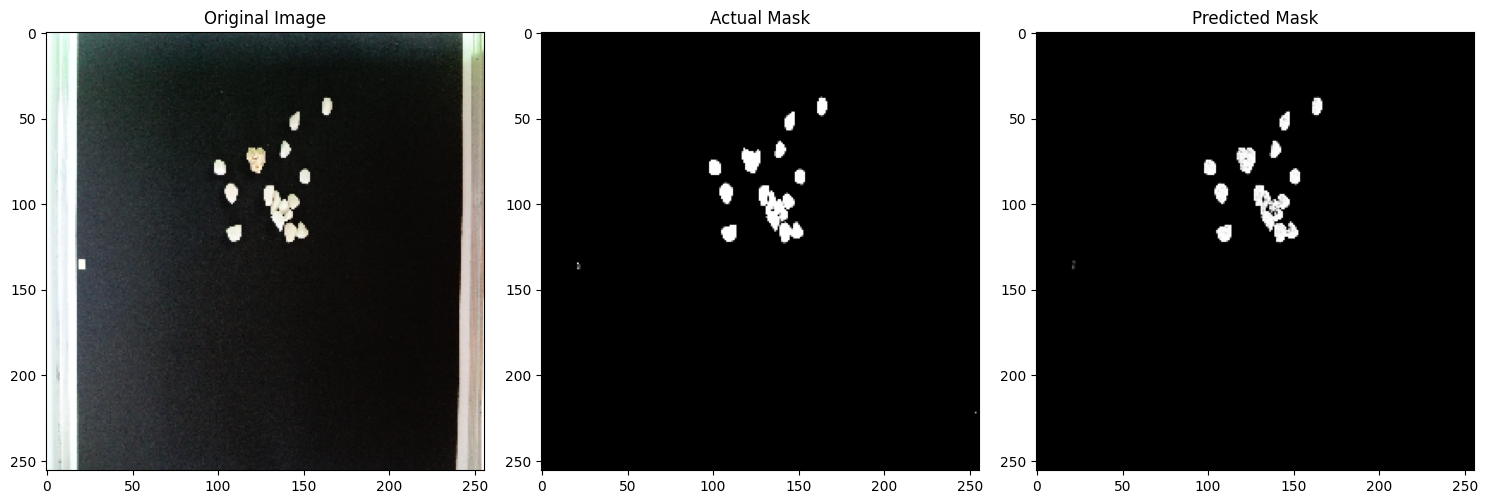


Fig-15

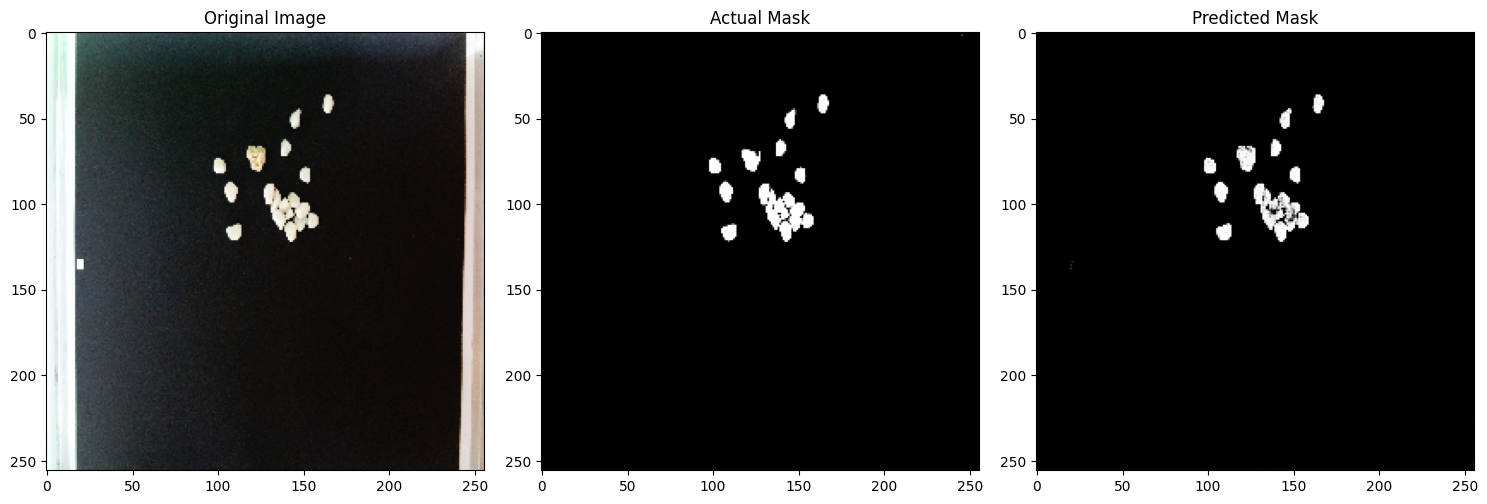


Fig-16



Fig-17

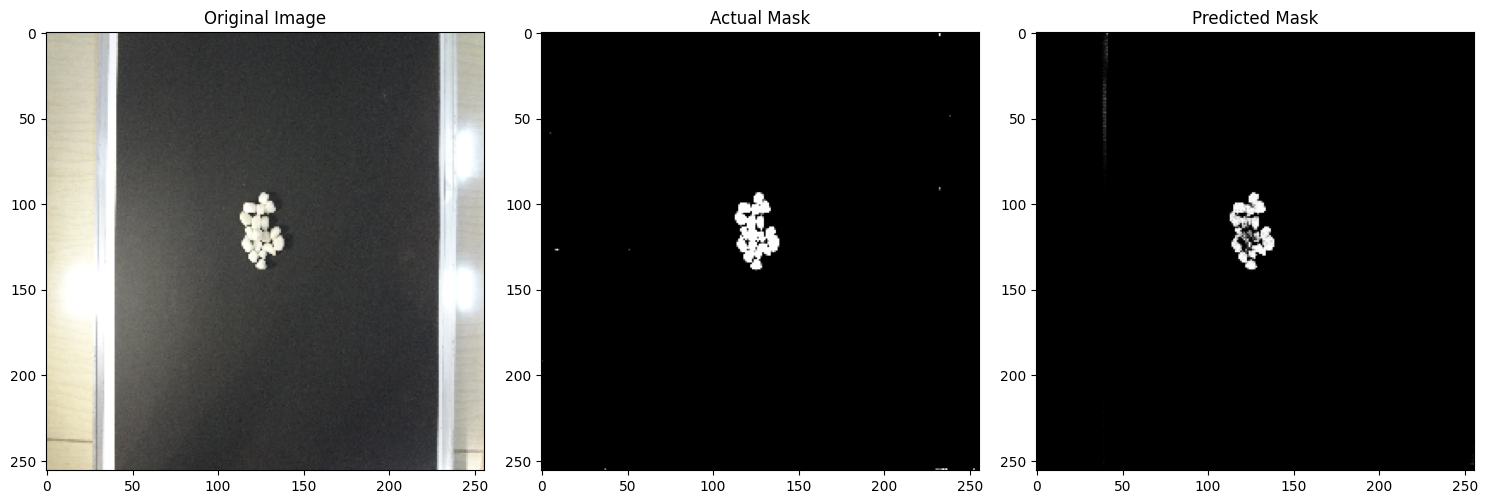


Fig-18

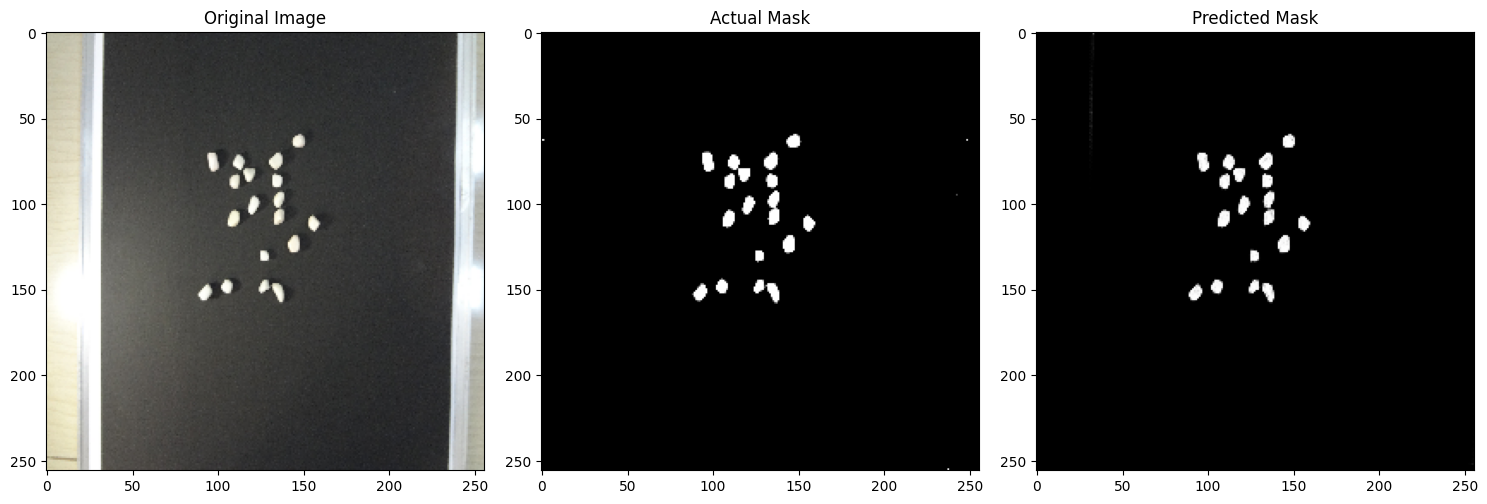


Fig-19



Fig-20

1. **Reason for selecting the U-net architecture**:
2. Encoder-Decoder Structure with Skip Connections:

what makes U-Net so good at image segmentation is skip connections and decoder networks. What we have done till now is similar to any CNN. The skip connections and decoder network separate the U-Net from other CNNs.

The encoder effectively captures features from the input image at different resolutions.

The decoder leverages transposed convolutions to recover spatial information while up sampling feature maps.

Skip connections directly concatenate feature maps from corresponding encoder and decoder levels. This is crucial because:

It preserves fine-grained details that might be lost during down sampling in the encoder.

It allows the decoder to make more accurate predictions by incorporating higher-level semantic information from the encoder.

1. The combination of the two paths enables U-net to learn both global and local features and to achieve high accuracy in segmentation tasks. One of the strengths of U-net is its versatility in accepting different types of input data, such as grayscale, color, and multi-channel images.
2. Data Efficiency: U-Net excels at learning from limited data, a common challenge in segmentation tasks. Its architecture uses a relatively low number of parameters, enabling effective learning without overfitting on smaller datasets.
3. **Hardware requirements for running the code**: