1. **Data Cleaning and Preprocessing**

Before training our deep learning model, we carefully cleaned and prepared the dataset to improve the quality and reliability of the training data. This step is crucial because machine learning models are very sensitive to the quality of input images. Even small issues in images can confuse the model, reduce accuracy, or cause it to learn noise instead of meaningful patterns.

**1. Loading and Checking Images**

We started by loading all images from the original dataset folder. Since the dataset contains different image formats (like .jpg, .png, .tif), we included all common image types.

**2. Detecting Corrupted or Low-Quality Images**

Not all images are good for training. Some images might be:

* **Corrupted or unreadable:** The file cannot be properly opened.
* **Completely black or with large black areas:** More than 50% of pixels are black, so no useful information.
* **Very low contrast:** Images look dull or flat, with little texture.
* **Too blurry or too noisy:** Images might be too smooth or have random noise.

To detect these issues, we wrote a function called is\_corrupted\_or\_noisy that checks:

* If the image file loads correctly.
* If more than half of the pixels are black.
* The standard deviation of pixel intensities (low values mean low contrast).
* The Laplacian variance to detect blurriness or noise.

If an image failed any of these tests, it was considered corrupted or low quality and excluded from the cleaned dataset.

**3. Image Preprocessing**

For good-quality images, we:

* Resized them all to a fixed size of **224×224 pixels** to maintain consistency.
* Converted any grayscale images to 3-channel RGB images to match the model input requirements.
* Applied a **Gaussian blur filter** (3×3 kernel) to reduce minor noise while keeping important details.

**4. Saving Cleaned Images**

All cleaned and preprocessed images were saved in a new folder called cleaned\_dataset, preserving their original folder and class structure.

**5. Results and Verification**

* Total processed (cleaned) images: **[number will appear after running code]**
* Total corrupted or noisy images skipped: **[number]**

We saved the list of corrupted and removed images in a file called corrupted\_images.txt to review later.

**6. Manual Inspection**

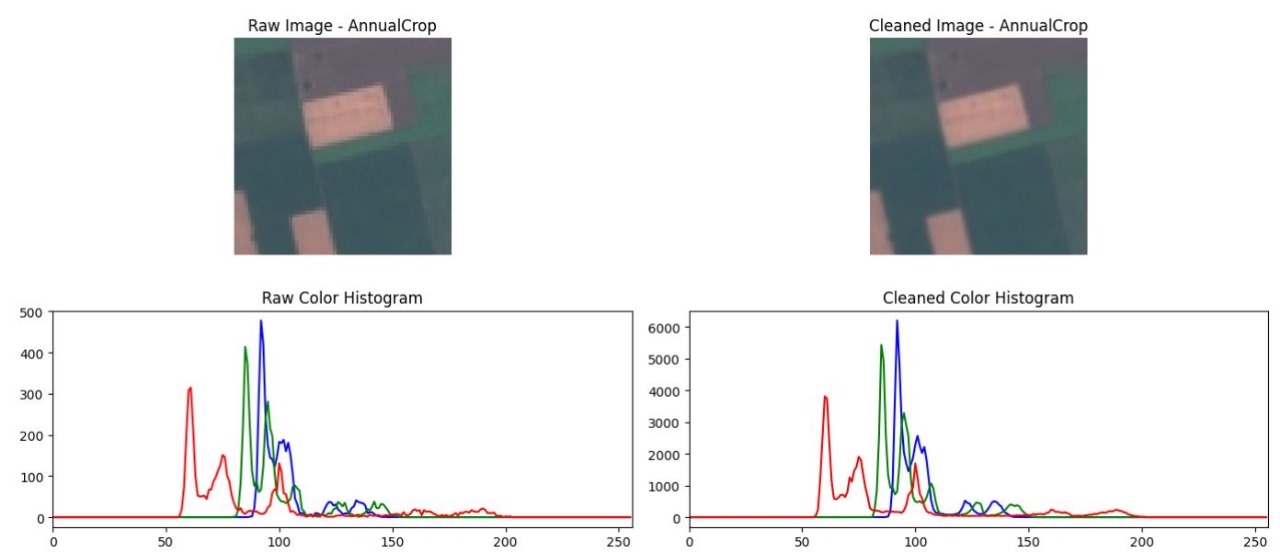
After automatic cleaning, we visually inspected samples of both the removed and cleaned images. This helped confirm that:

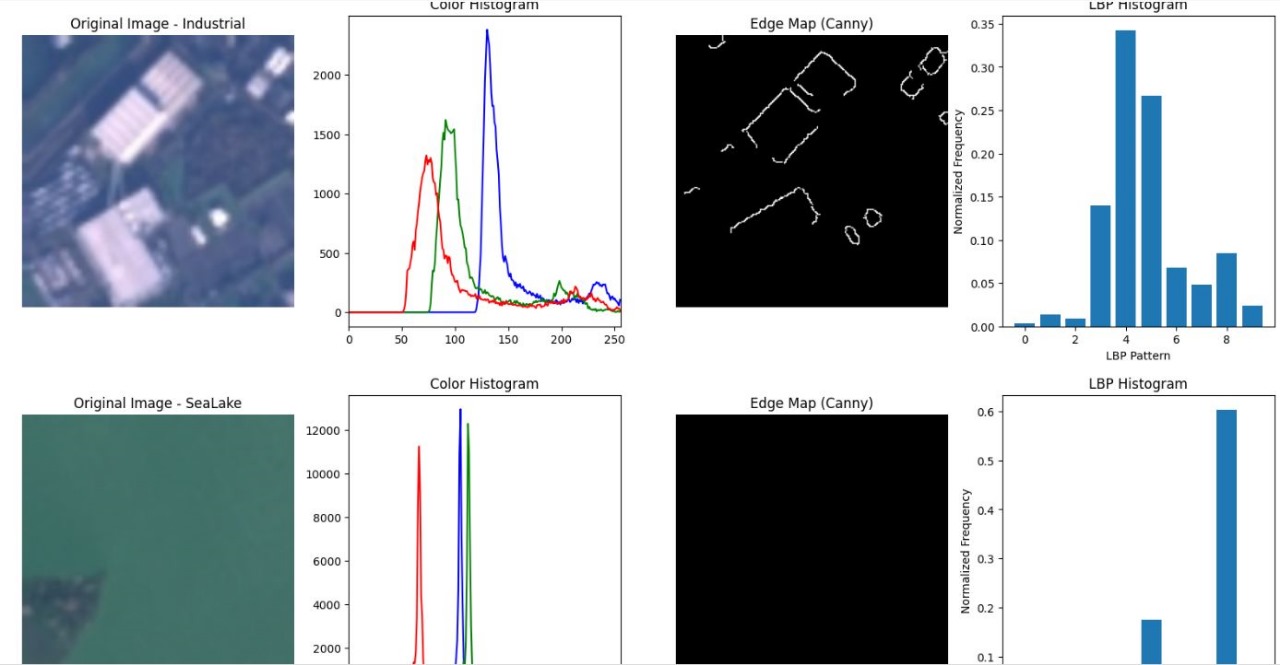
* The cleaning code correctly removed images that would likely harm the model training.
* Sometimes images look fine to the human eye but have subtle quality issues (like low texture or slight blur) that affect model learning. Our automated method caught these to improve training data quality.

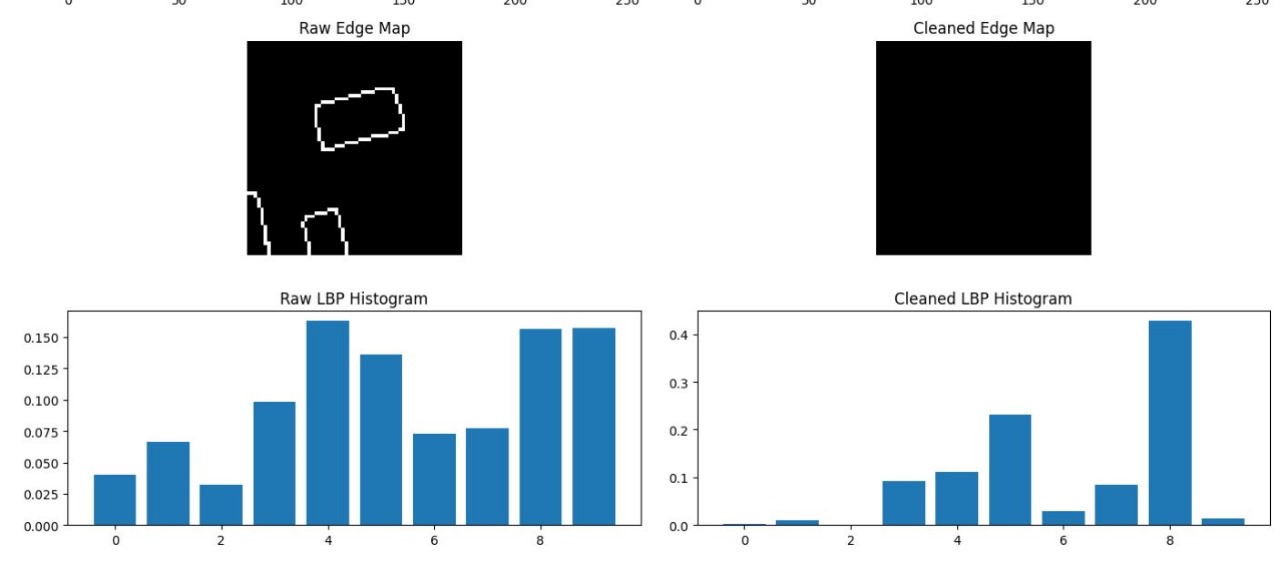
**Why This Matters**

As suggested by ChatGPT, even small imperfections in image quality can cause “noise” during training, making it harder for the model to learn well. By removing low-quality images, we ensure the model sees clearer, more meaningful examples, which helps improve accuracy and robustness.

1. **Feature Visualization Plots**

****

****

****

1. **CNN ARCHITECTURE DIAGRAM**

| **Layer Type** | **Number of Filters / Units** | **Kernel Size** | **Activation Function** | **Batch Normalization** | **Dropout Rate** | **Other Details** |
| --- | --- | --- | --- | --- | --- | --- |
| Conv2D | 32 | (3, 3) | ReLU | Yes | 0.25 | Padding = 'same' |
| MaxPooling2D | - | (2, 2) | - | - | - |  |
| Conv2D | 64 | (3, 3) | ReLU | Yes | 0.25 | Padding = 'same' |
| MaxPooling2D | - | (2, 2) | - | - | - |  |
| Conv2D | 128 | (3, 3) | ReLU | Yes | 0.25 | Padding = 'same' |
| MaxPooling2D | - | (2, 2) | - | - | - |  |
| Flatten | - | - | - | - | - |  |
| Dense (Fully Connected) | 256 | - | ReLU | Yes | 0.5 |  |
| Dense (Output) | Number of classes (e.g., 10) | - | Softmax | No | No | Output Layer |

### Explanation of Algorithms and Techniques

In this project, I used several important algorithms and techniques to improve the CNN model’s performance and make training more effective.

**Data Augmentation:**  
Since the dataset size is limited, I applied data augmentation to artificially increase the diversity of training images. This included random rotations, horizontal flips, slight zooms, and brightness changes. Data augmentation helps the model generalize better by learning from varied examples, reducing the chance of overfitting.

**Learning Rate Scheduling (ReduceLROnPlateau):**  
I used a learning rate scheduler that reduces the learning rate when the validation loss stops improving. This technique helps the model fine-tune weights by taking smaller steps during training, allowing better convergence and avoiding overshooting the optimal solution.

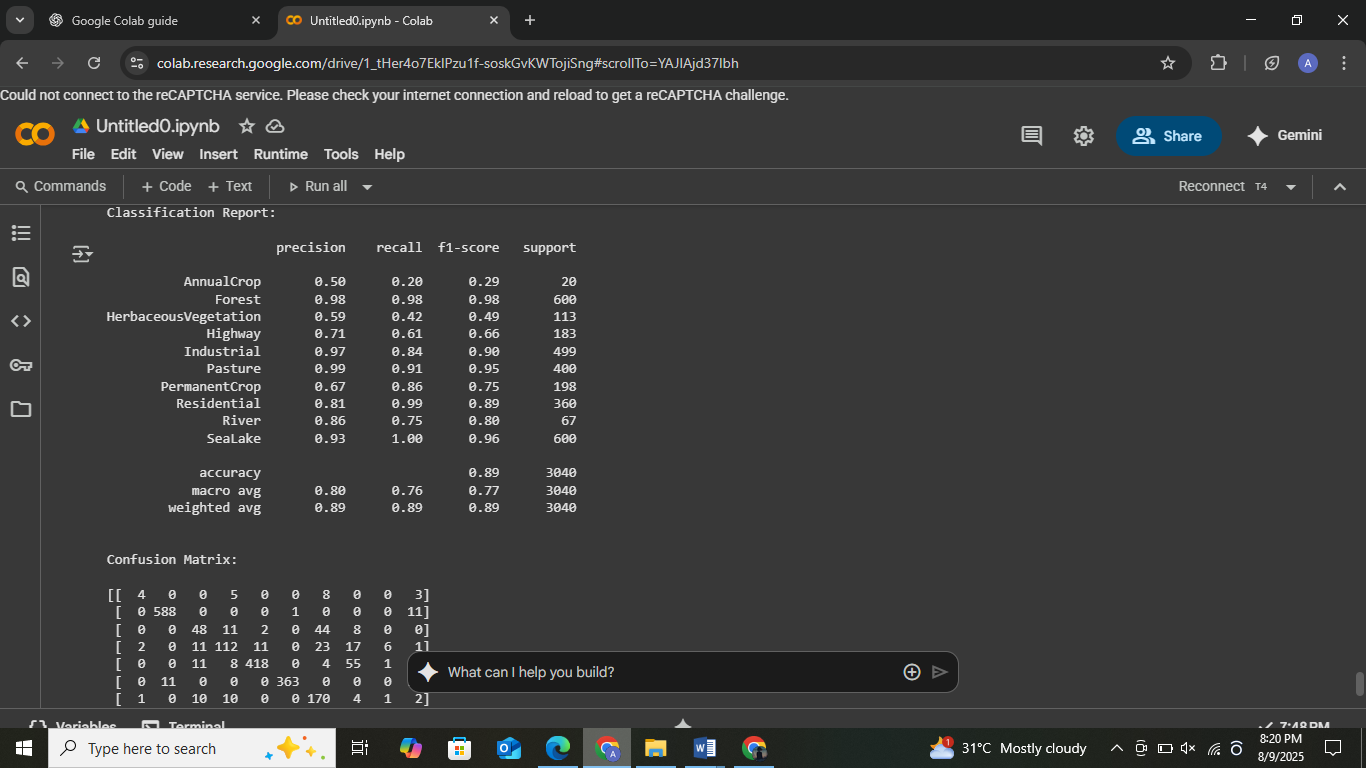
**Dropout:**  
Dropout layers were added after some convolutional and dense layers to randomly “drop” neurons during training. This prevents the model from relying too much on specific neurons and helps reduce overfitting by encouraging more robust feature learning.

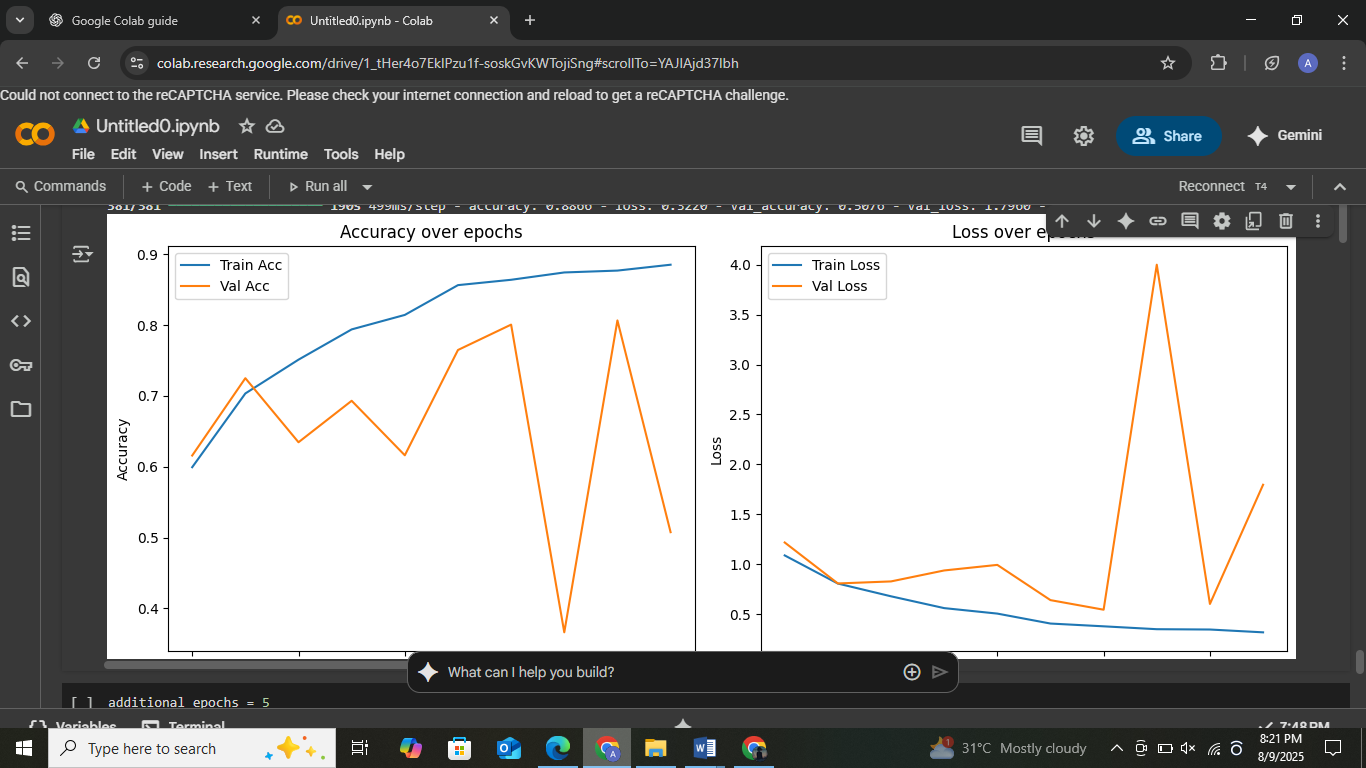
**Batch Normalization:**  
Batch normalization layers were added after convolutional and dense layers to normalize the inputs of each layer. This stabilizes and speeds up training by reducing internal covariate shift, allowing higher learning rates and improving model accuracy.

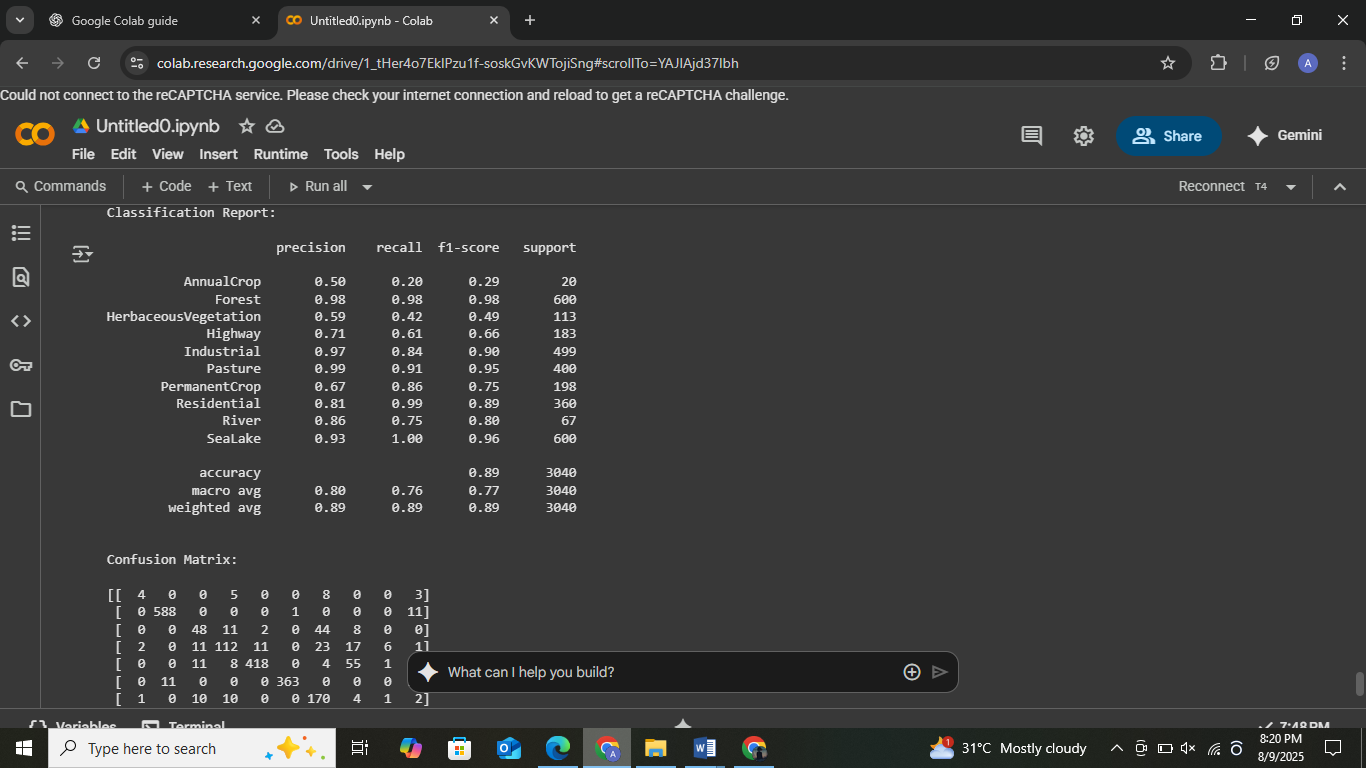
**Optimizer (Adam):**  
I chose the Adam optimizer because it combines the advantages of two other optimizers (AdaGrad and RMSProp). It adapts the learning rate for each parameter automatically, which helps achieve faster and more stable training.

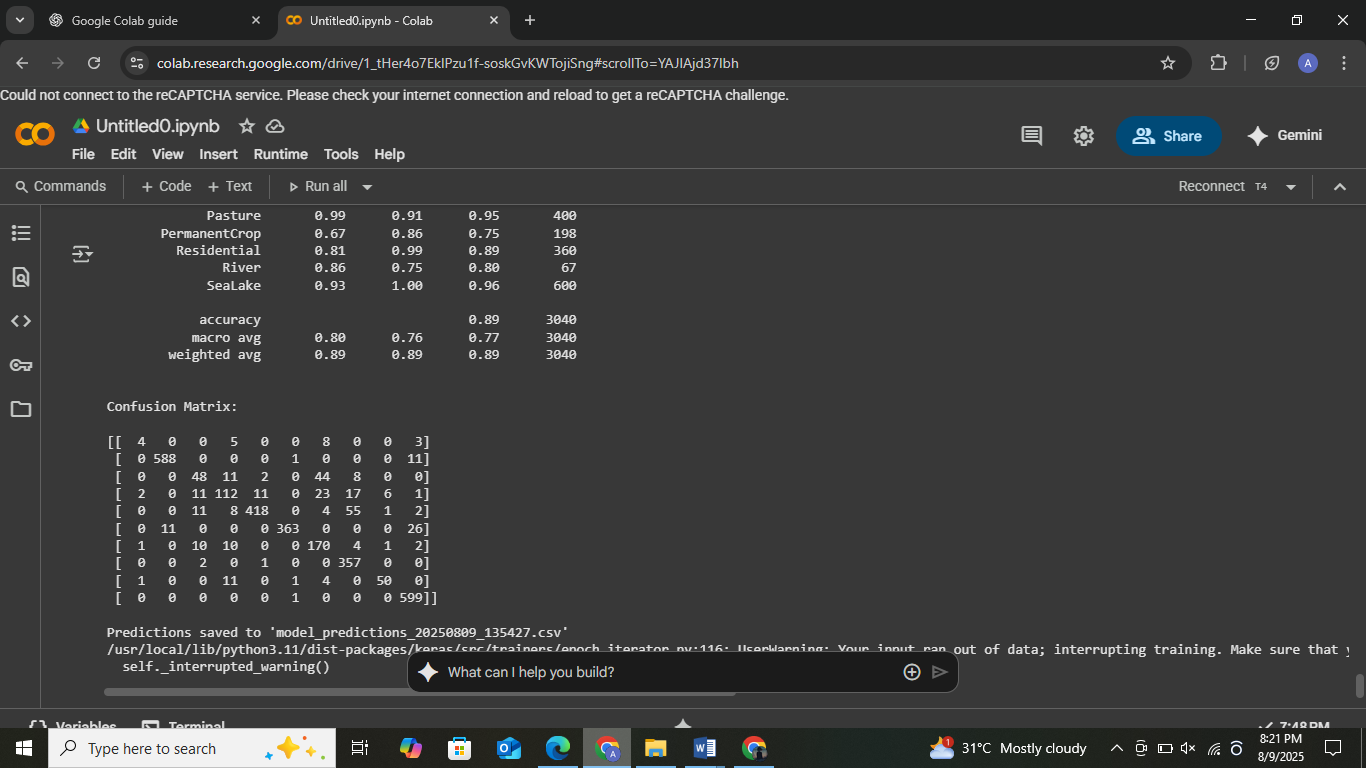
**Early Stopping:**  
Early stopping was used to monitor validation loss and stop training if the model stopped improving for several epochs. This prevents overfitting and saves training time by avoiding unnecessary epochs once the model has converged.

1. **COUNFUSION MATRIX AND TABLES AND ACCURACIES**









We were not able to download the submission file directly due to a runtime issue with Google Colab. However, you can see in the attached screenshots that the file named **model\_predictions.csv** has been successfully saved. Since this issue is related to the Colab environment, I am submitting the entire code in the repository. You can run the code yourselves, and then you will be able to generate and access the CSV submission file without any problems.