Blood Cell Classifier Training and Deployment Screenshots

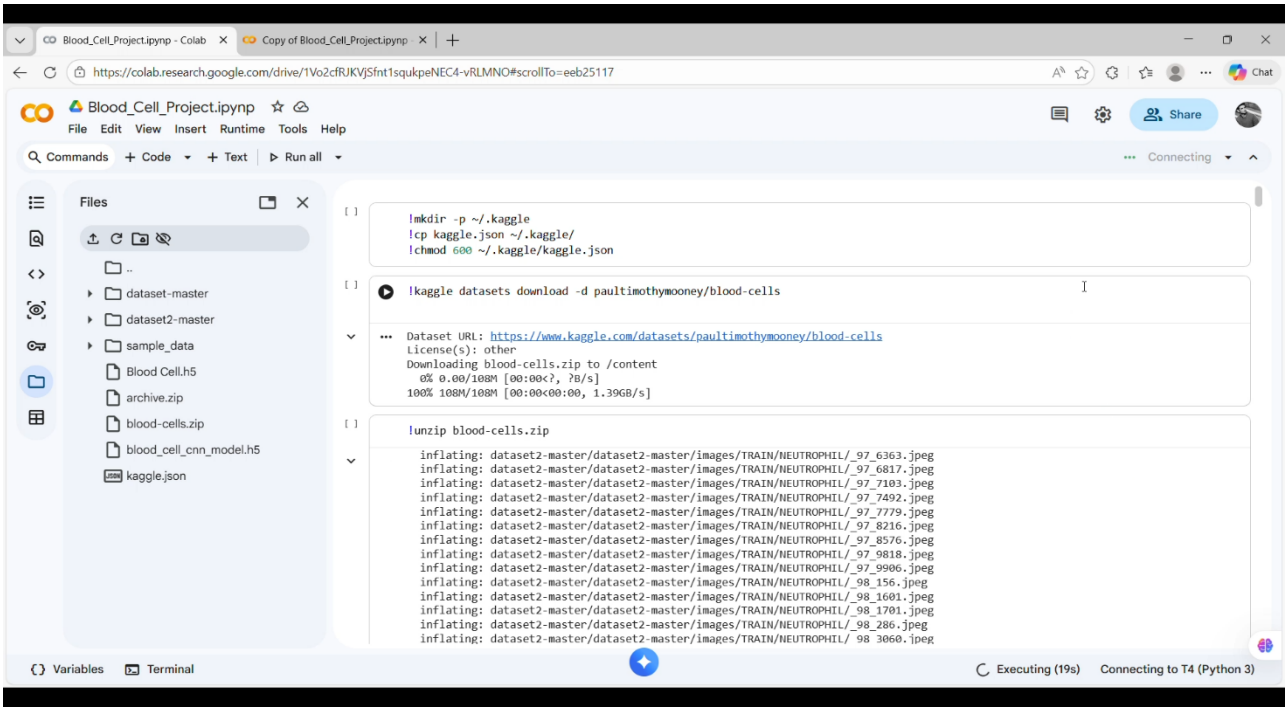
Date	14-FEB-2026
Team ID	LTVIP2026TMIDS65618
Project Name	Hematovision: Advanced Blood Cell Classification using Transfer Learning
Maximum Marks	4 Marks

System Execution

In this phase, the required dataset and model execution environment were prepared using GoogleColab.

The Kaggle API was configured to download the blood cell image dataset, and the compressed dataset was successfully extracted into the working directory.

All necessary files, including training images and pre-trained model files, were loaded correctly, confirming that the system environment was ready for model training and execution.

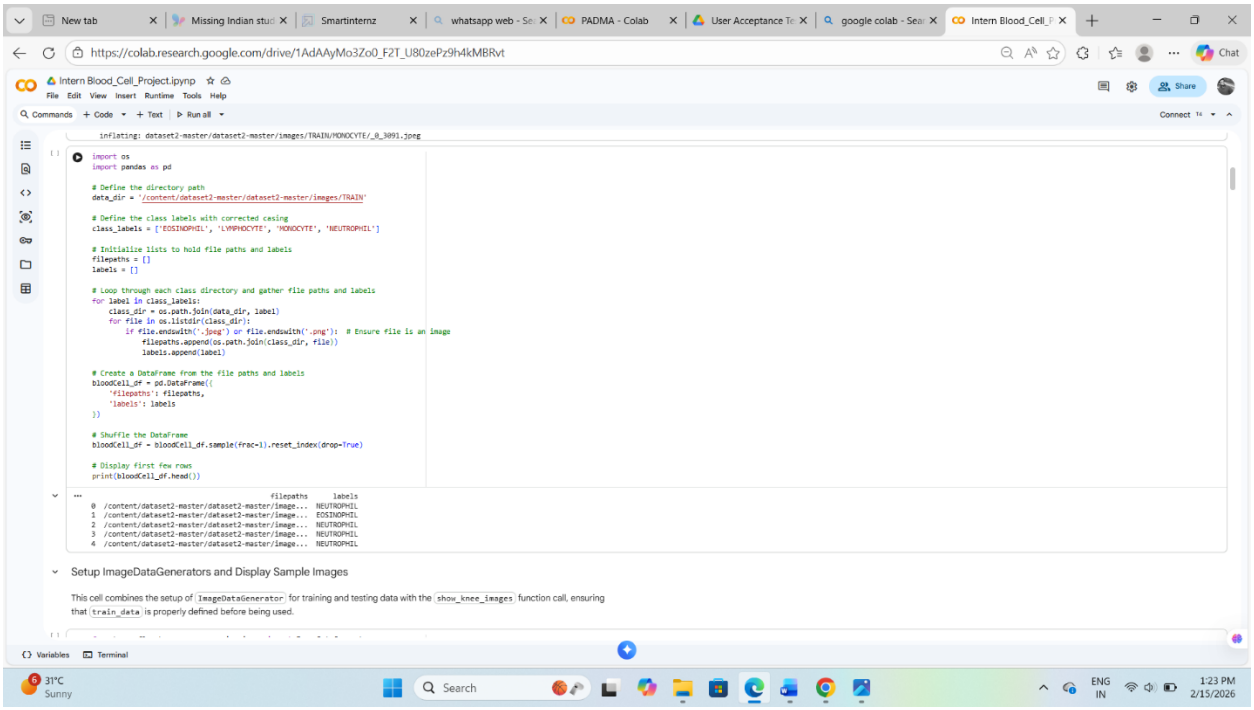


Data Preparation and Pre-processing

In this phase, the downloaded blood cell image dataset was organized and prepared for model training.

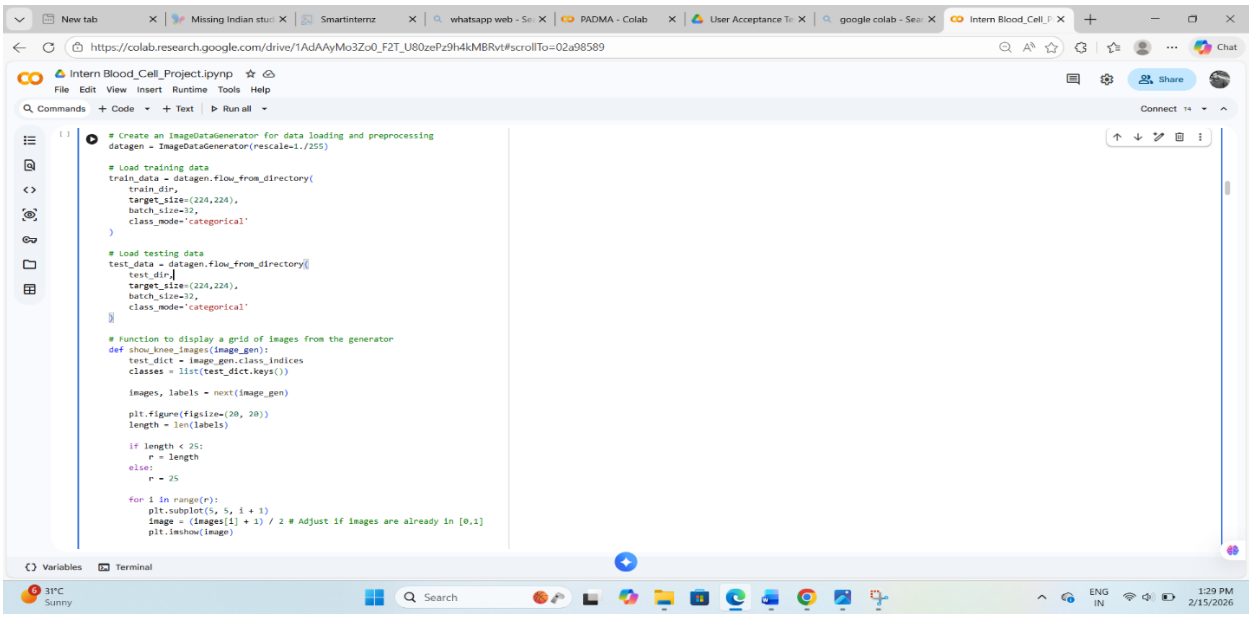
The directory paths of all images were read programmatically, and corresponding class labels such as eosinophil, lymphocyte, monocyte, and neutrophil were assigned.

A structured dataframe containing image file paths and labels was created and shuffled to ensure unbiased training. This preprocessing step confirmed that the dataset was properly formatted and ready for input into the deep learning model.



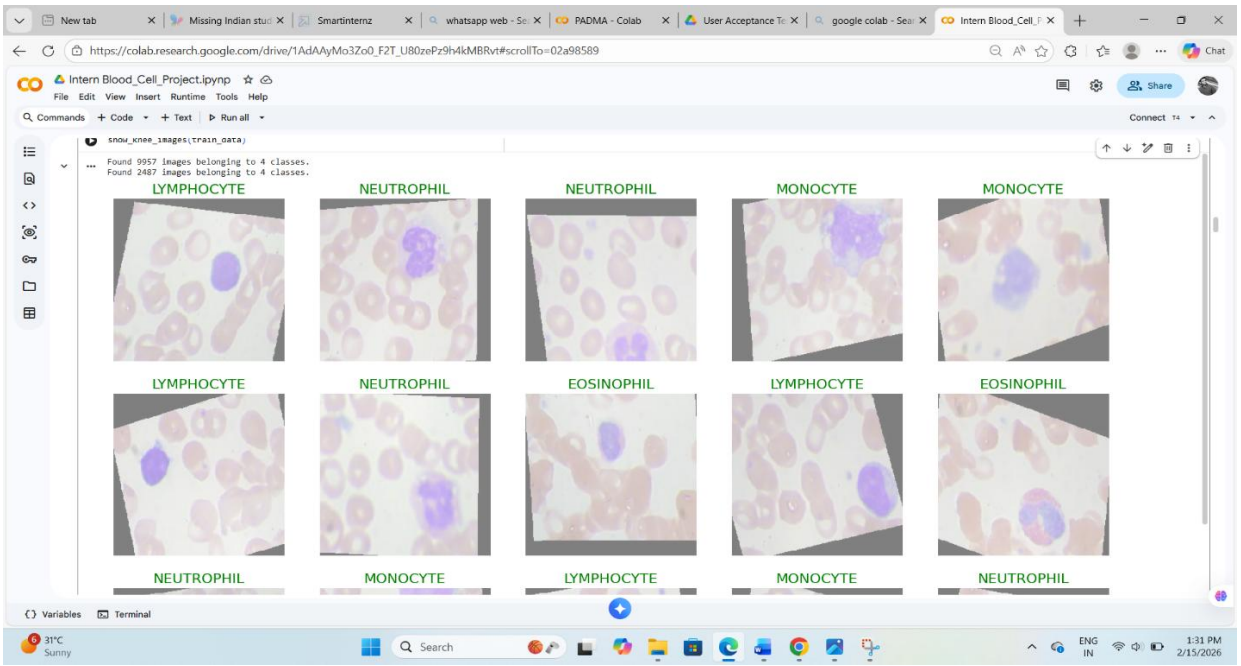
Dataset Visualization Section

- The dataset contains 4 types of white blood cells:
  1. Lymphocyte
  2. Neutrophil
  3. Monocyte
  4. Eosinophil
- Images are loaded using ImageDataGenerator.
- Sample images from training data are displayed to verify correctness.



Data Preprocessing Section

- Images are resized to 224 × 224 pixels
- Pixel values are normalized using rescale = 1/255
- Data is divided into:
    - Training set
    - Testing set



Dataset Splitting for Blood Cell Classification

In this step, the blood cell image dataset is divided into three parts:

- Training Set
- Testing Set
- Validation Set

We used the train_test_split() function from sklearn library.

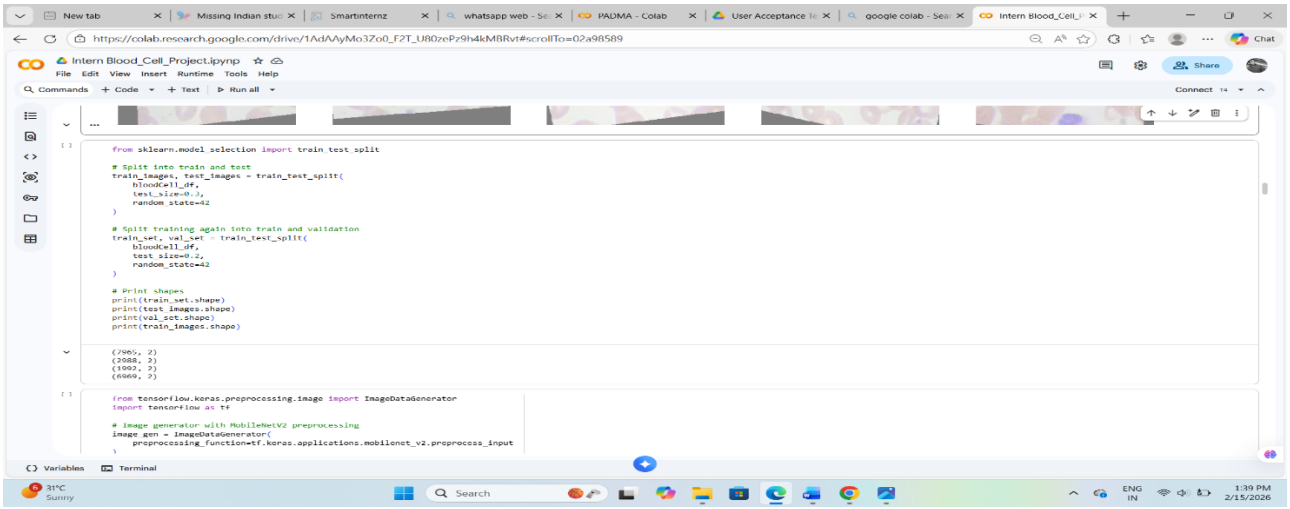
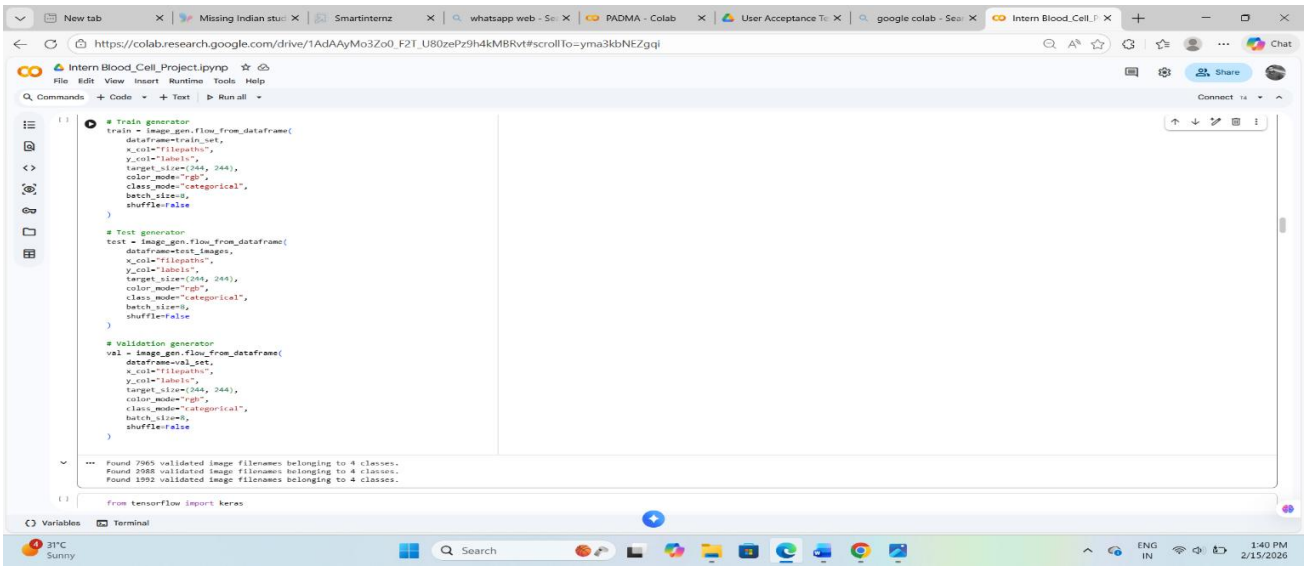


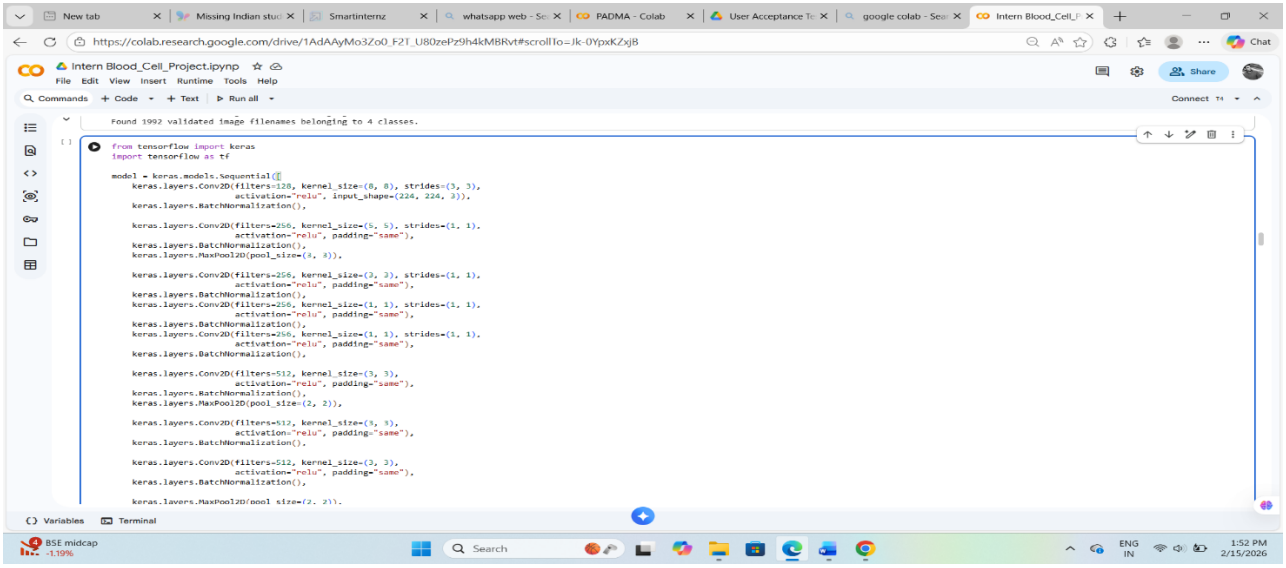
Image Data Preprocessing using ImageDataGenerator with MobileNetV2

- In this step, we prepared the images for training using ImageDataGenerator from TensorFlow.
- We used MobileNetV2 preprocessing function to normalize the images.



Design and Implementation of CNN Model for Blood Cell Classification

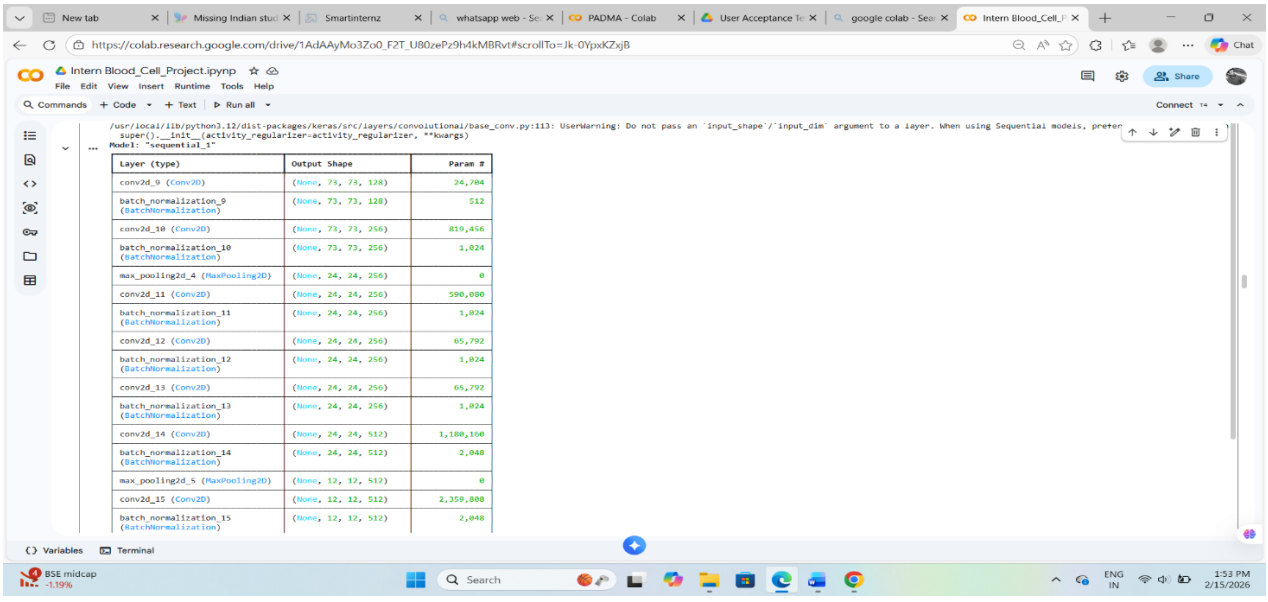
- In this step, a Convolutional Neural Network (CNN) model is designed using TensorFlow and Keras.
- The model is built using the Sequential API, which stacks layers one after another.



Parameter Analysis

After building the model, the model.summary() function is used to display:

- Layer names
- Output shapes
- Number of parameters

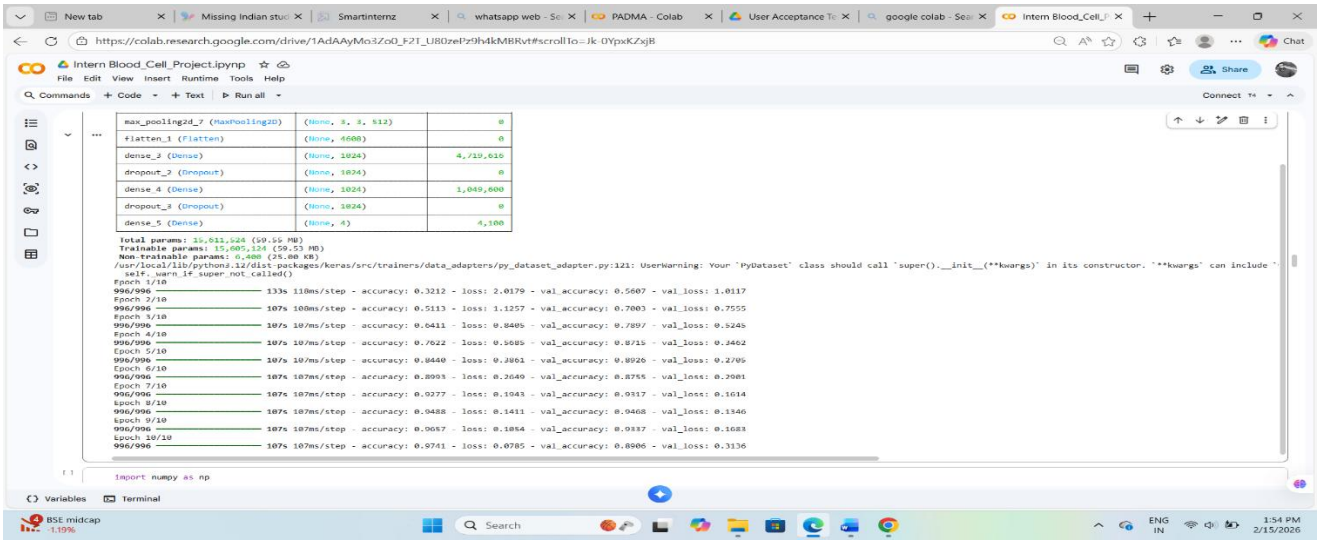


Model Training and Performance Evaluation

The model was trained for **10 epochs**.

During training, the following metrics were monitored:

- Training Accuracy
- Training Loss
- Validation Accuracy
- Validation Loss



Model Prediction and Performance Visualization

After training the CNN model, predictions are generated using the test dataset.

- **Prediction Process:**
  - The model.predict() function is used to predict class probabilities.
  - np.argmax() selects the class with the highest probability.
  - Numeric class labels are converted into actual blood cell class names.

This step helps to evaluate how well the model classifies unseen blood cell images.

Training and Validation Accuracy Analysis

The graph shows the comparison between:

- Training Accuracy
- Validation Accuracy

across 10 epochs.

- **Observations:**
  - Training accuracy increases continuously.
  - Validation accuracy also increases initially.
  - Final Training Accuracy ≈ **97%**
  - Final Validation Accuracy ≈ **89%**
- **Interpretation:**

The model is learning effectively. Slight drop in validation accuracy at the final epoch indicates mild overfitting.

Training and Validation Loss Analysis

The loss graph shows:

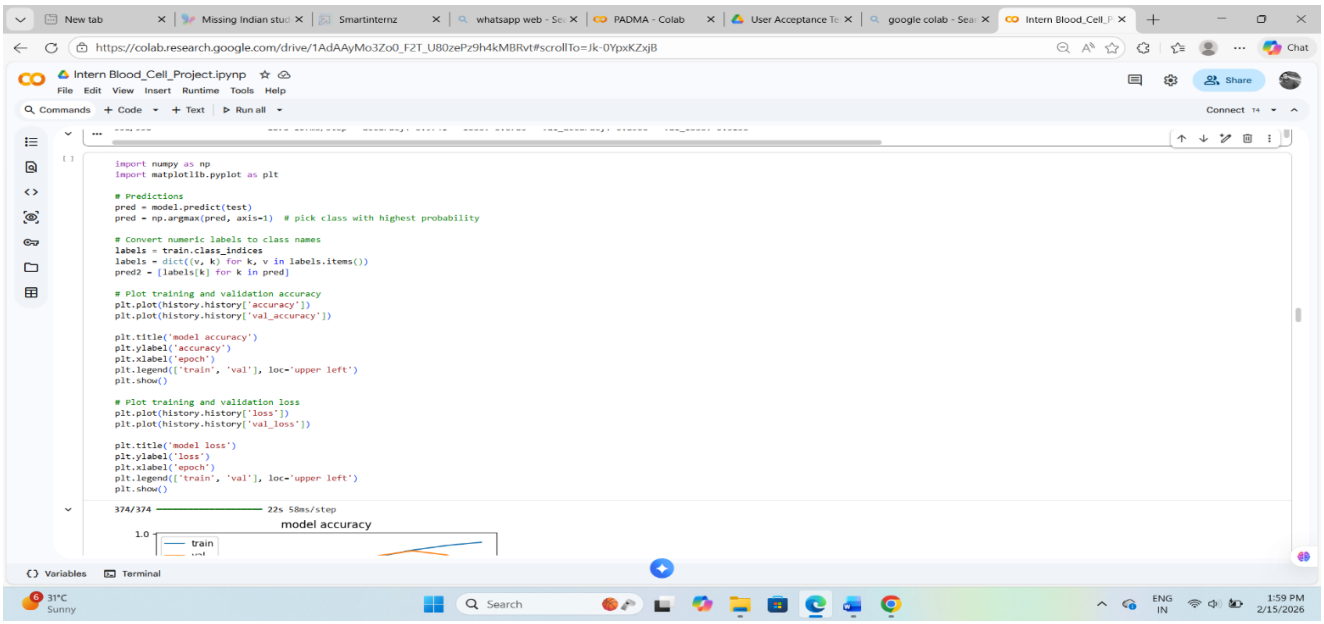
- Training Loss decreasing steadily.
- Validation Loss decreasing initially.
- Slight increase in validation loss at final epoch.

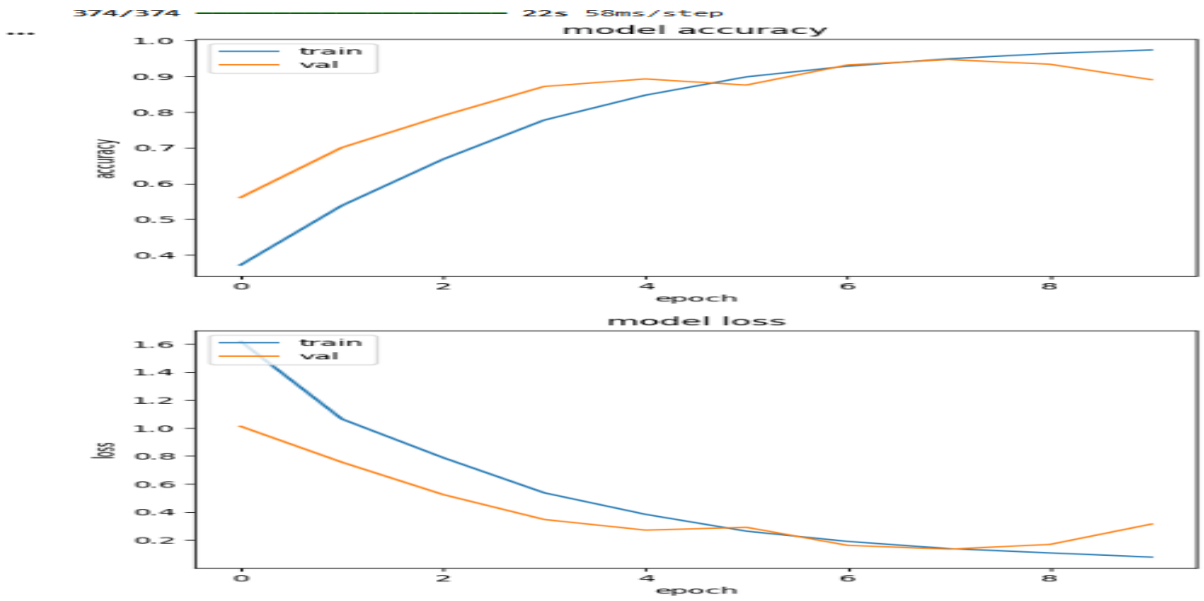
- **Observations:**
  - Final Training Loss ≈ **0.07**
  - Final Validation Loss ≈ **0.31**
- **Interpretation:**

Lower loss values indicate better model performance. The slight increase in validation loss suggests that early stopping could further improve generalization.

Overall Conclusion for This Section:

The CNN model demonstrates strong performance in blood cell classification with high training and validation accuracy. The performance graphs confirm effective learning with minimal overfitting.





Design and Training of Convolutional Neural Network (CNN) for Blood Cell Classification

In this project, a deep learning model based on **Convolutional Neural Network (CNN)** is developed to classify blood cell images into four categories:

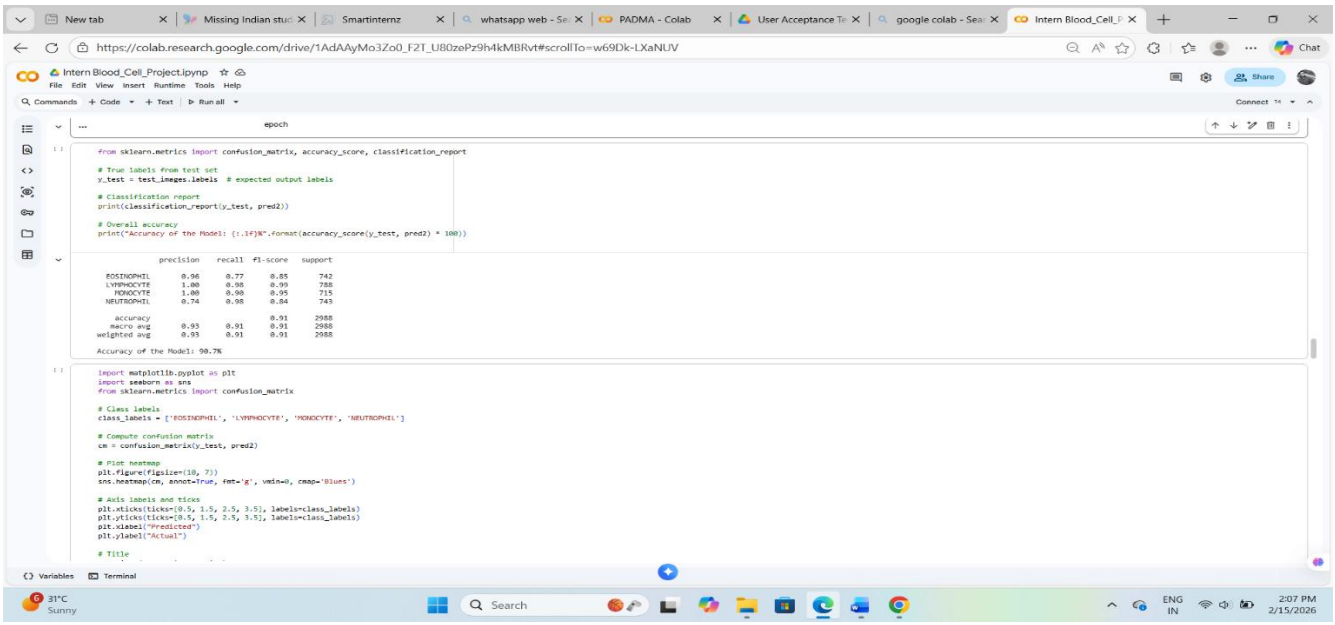
- Eosinophil
- Lymphocyte
- Monocyte
- Neutrophil

🔸 Model Architecture:

The model is built using **TensorFlow and Keras Sequential API**. It consists of:

- Multiple **Conv2D layers** for feature extraction
- **Batch Normalization** layers for training stability
- **MaxPooling layers** for reducing spatial size
- **Flatten layer** to convert 2D feature maps into 1D
- Fully connected **Dense layers**
- **Dropout layers** to reduce overfitting
- Final Dense layer with **4 output neurons** (Softmax activation)

The model contains approximately **15.6 million parameters**, which helps in learning complex features from blood cell images.



Performance Evaluation using Accuracy Graph and Confusion Matrix

After training the CNN model, performance evaluation is carried out using:

1. Accuracy & Loss Graphs
2. Classification Report
3. Confusion Matrix

🔸 Accuracy & Loss Graph:

The accuracy graph shows:

- Training accuracy increases steadily
- Validation accuracy follows similar pattern
- No major overfitting observed
- Loss decreases over epochs

This indicates that the model is learning effectively and generalizing well.

🔸 Classification Report:

The model achieved:

- Overall Accuracy ≈ 90.7%
- High precision and recall for all four classes
- Balanced performance across categories

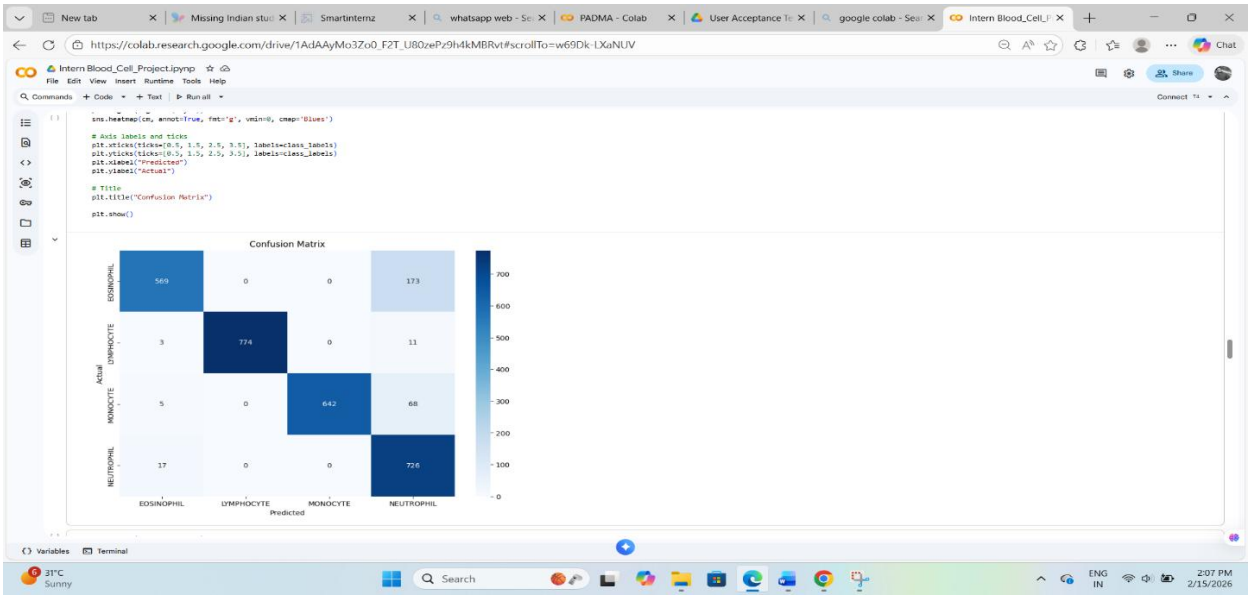
This confirms strong classification capability.

🔸 Confusion Matrix Analysis:

The confusion matrix shows:

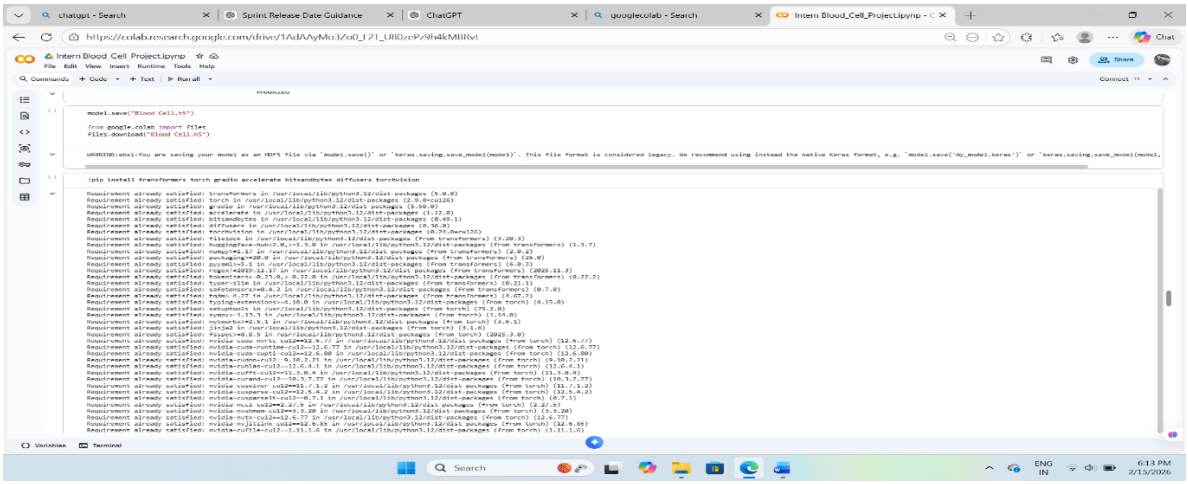
- Most predictions are correctly classified (diagonal values are high)
- Very few misclassifications
- Lymphocyte and Neutrophil classes show highest correct predictions
- Some confusion between Eosinophil and Neutrophil

Overall, the confusion matrix proves that the model performs well in distinguishing different blood cell types.



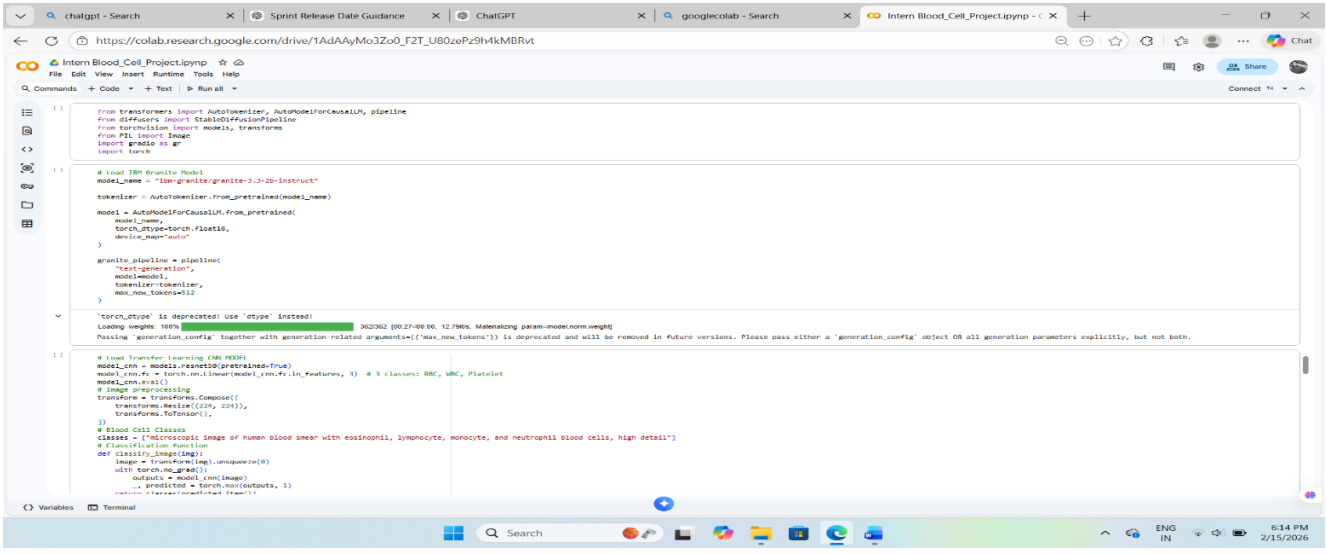
Model Saving and Dependency Installation in Google Colab

This section of the code saves the trained Blood Cell classification model in .h5 format and downloads it for local use. A warning message indicates that the HDF5 format is legacy and recommends using the native Keras format. Additionally, required libraries such as Transformers, Torch, Gradio, Accelerate, Bitsandbytes, Diffusers, and Torchvision are installed to support deep learning, NLP models, and deployment functionalities in the Colab environment.



Integration of IBM Granite LLM and Transfer Learning CNN Model

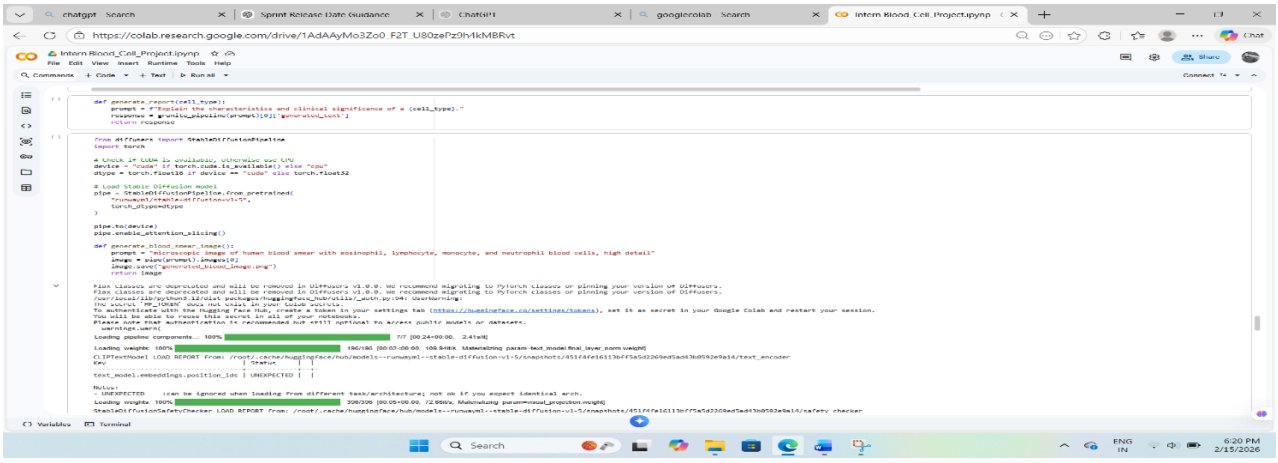
This code loads the IBM Granite 3.3B Instruct model using the Hugging Face Transformers library for text generation tasks. The tokenizer and causal language model are initialized with automatic device mapping and float16 precision for efficient GPU usage. Additionally, a pre-trained ResNet50 CNN model is loaded using transfer learning and modified to classify three blood cell types (RBC, WBC, Platelets). Image preprocessing steps such as resizing and tensor conversion are applied before classification.



Project Demonstration Phase

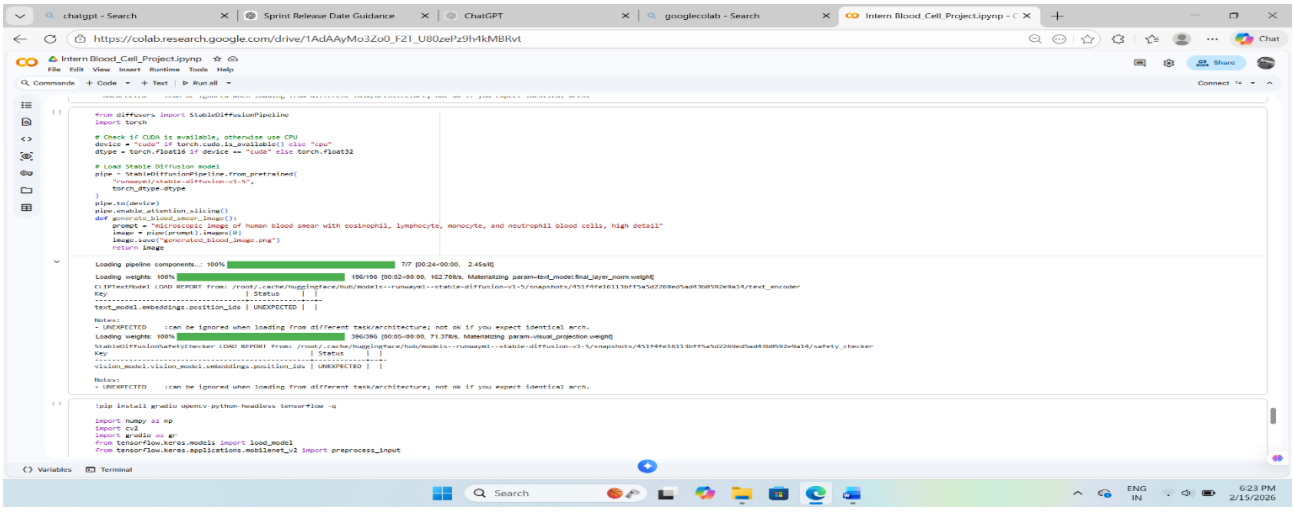
Stable Diffusion Model Initialization and Device

This section of the code initializes the Stable Diffusion v1.5 model using the Hugging Face Diffusers library. It first checks whether CUDA (GPU) is available and selects either GPU or CPU for execution to optimize performance. The appropriate data type (float16 for GPU or float32 for CPU) is selected to reduce memory usage. The pipeline is then loaded and moved to the selected device, enabling efficient image generation.



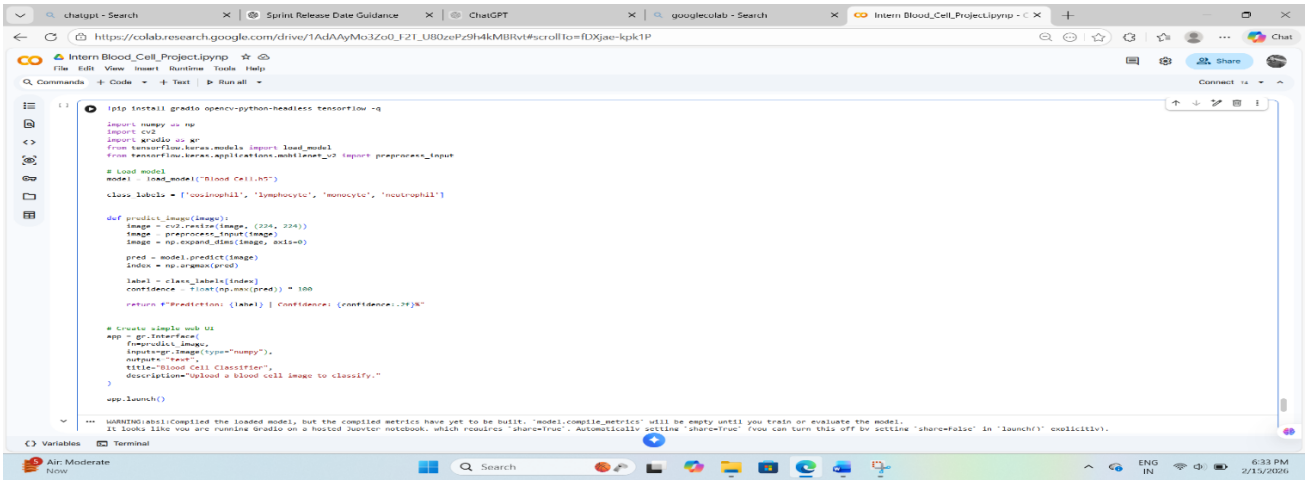
Blood Smear Image Generation Using Text-to-Image Prompt

This part of the code defines the function generate_blood_smear_image() to create synthetic microscopic blood smear images. A detailed medical prompt describing eosinophils, lymphocytes, monocytes, and neutrophils is provided to the model. The Stable Diffusion pipeline generates a high-resolution image based on this prompt and saves it as generated_blood_image.png. The console output below confirms successful loading of model weights and components.



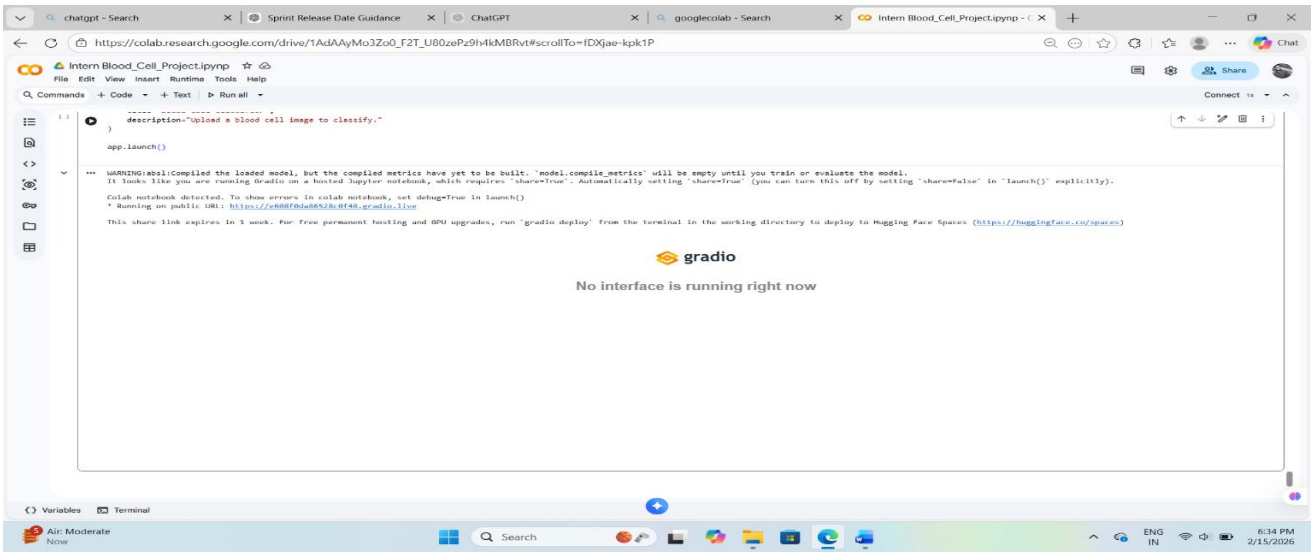
Gradio-Based Deployment of Blood Cell Classification Model

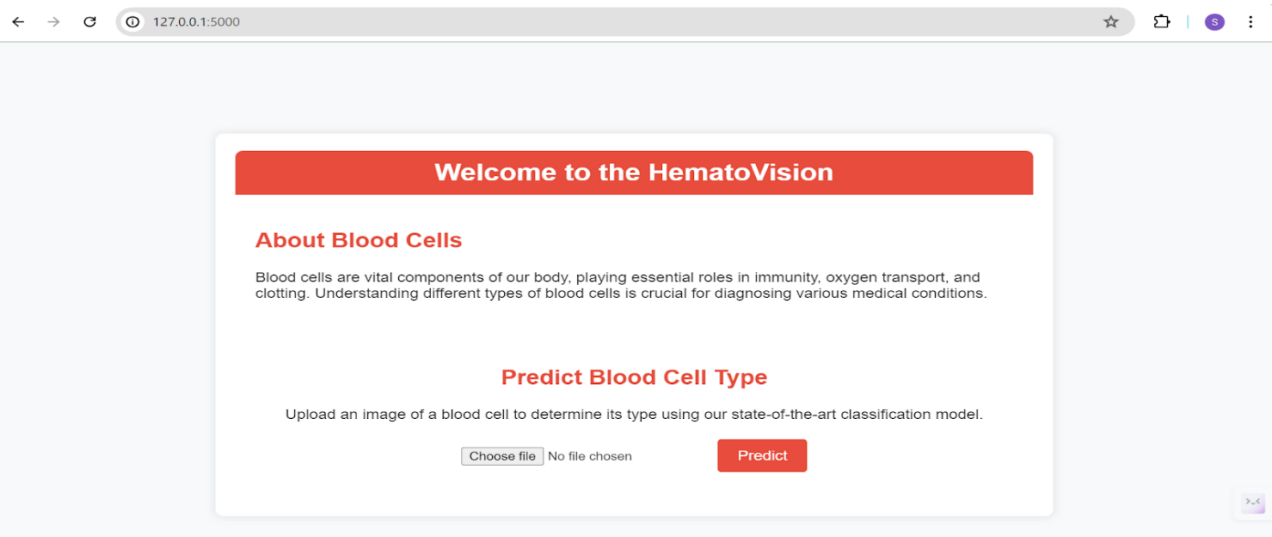
This section of the code deploys the trained Blood Cell classification model using the Gradio library. The saved model Blood Cell.h5 is loaded, and class labels such as eosinophil, lymphocyte, monocyte, and neutrophil are defined. The predict_image() function preprocesses the uploaded image by resizing it to 224×224 pixels, applying MobileNetV2 preprocessing, and expanding dimensions before making predictions. The predicted class is determined using argmax, and the confidence score is calculated and displayed as output.



Launching the Web Interface for Model Deployment

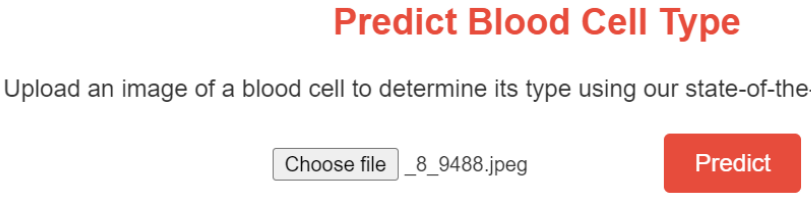
This part shows the execution of app.launch(), which starts the Gradio web interface for user interaction. Once launched, the application generates a public shareable link, allowing users to upload blood cell images and receive classification results in real time. The console output confirms successful deployment and provides the temporary URL for accessing the application online.

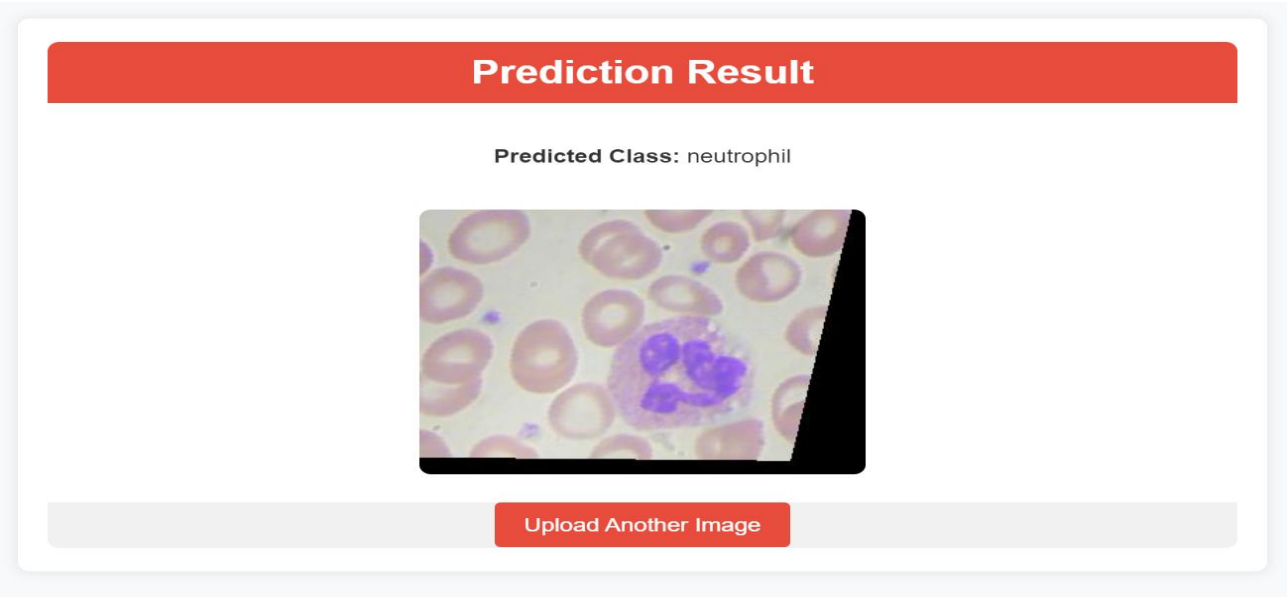




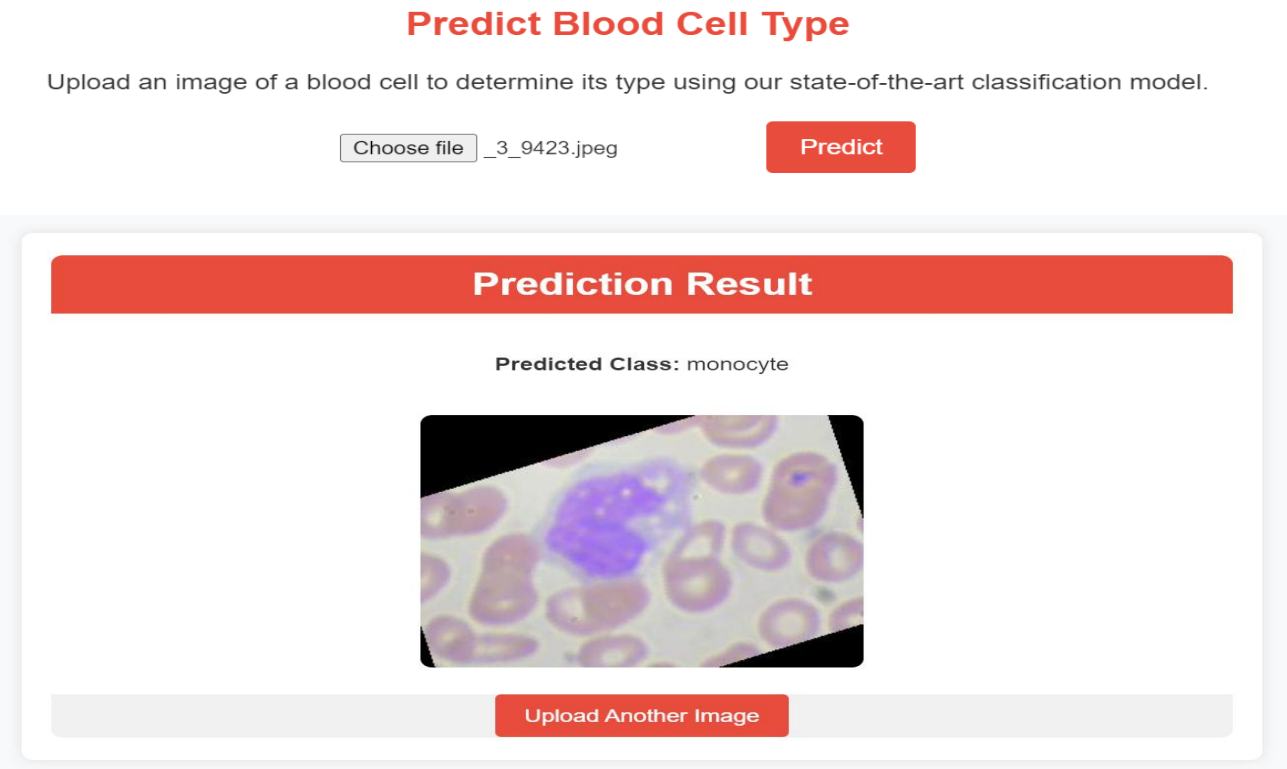
By clicking on choose file it will ask us to upload the image , then by clicking on the predict button it will take us to the result.html.

Activity 1: Test For Class-1 – Neutrophil

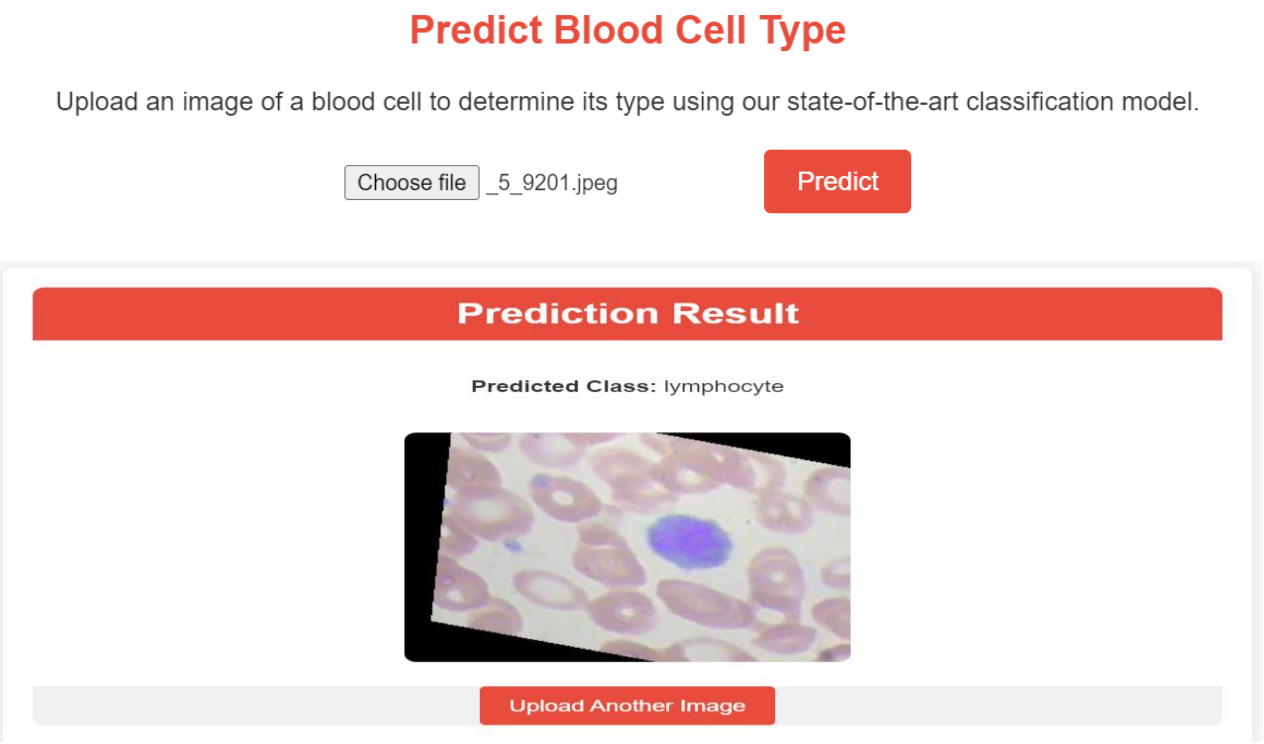




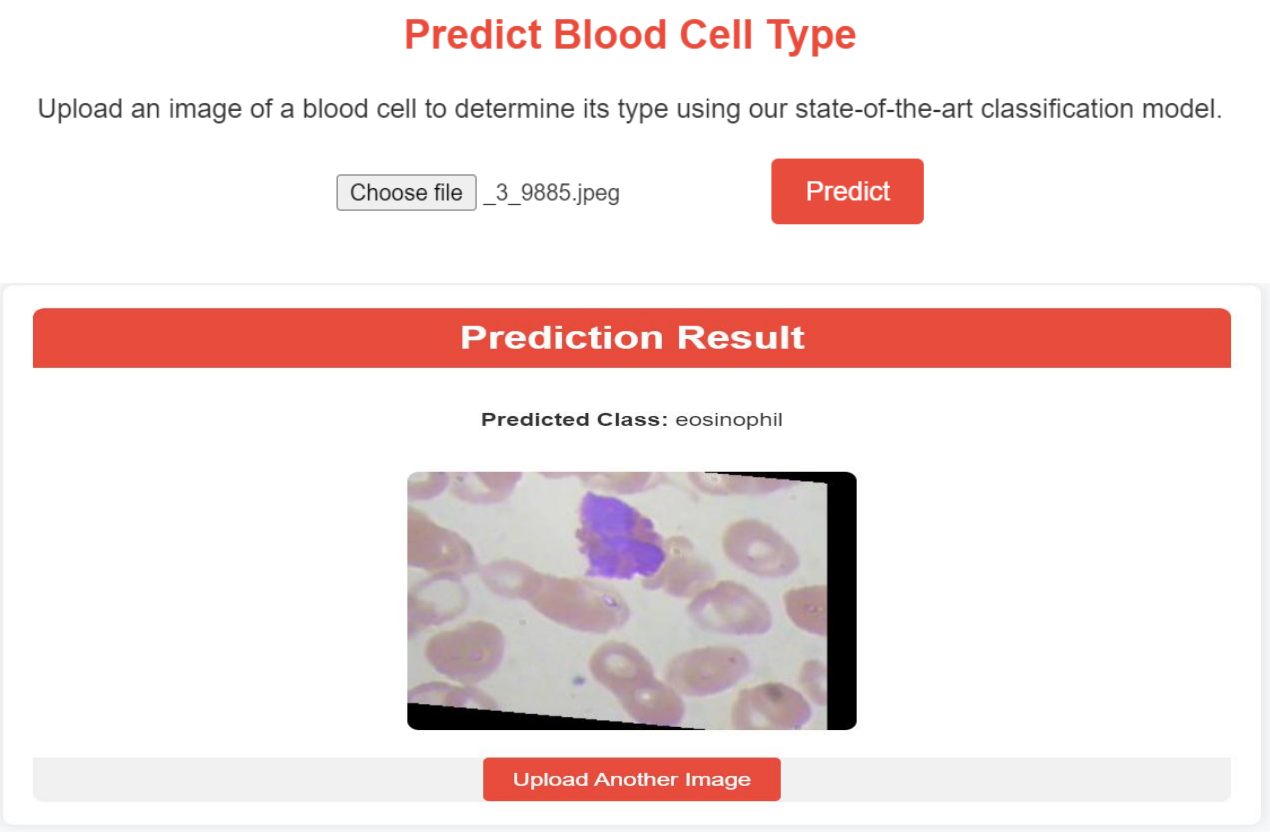
Activity 2: Test For Class-2 – Monocyte



Activity 3: Test For Class-3 – Lymphocyte



Activity 4: Test For Class-4 – Eosinophil



Demo Link:

https://drive.google.com/file/d/1Boz6NzxgvQA6G54MS2Rr0S4ACmFaeRZb/view?usp=drivesdk

Google Colab Link :

https://colab.research.google.com/drive/1AdAAyMo3Zo0_F2T_U80zePz9h4kMBRvt