A Sprint fixed period or duration in which a team works to complete a set of tasks

An **Epic** is a **big task or project** that is too large to complete in one sprint. It is broken down into **smaller tasks (stories)** that can be completed over multiple sprints.

A **Story** is a small task . It is part of an **Epic**.

A **Story Point** is a number that represents how much effort a story takes to complete. (usually in form of Fibonacci series)

1- Very Easy task
2- Easy task
3- Moderate task
5- Difficult task

Sprint 1: (5 Days)

Data Collection

Collection of Data **2**

To train HematoVision effectively, the project requires a high-quality, annotated dataset of blood cell images. Here's an overview of how the data is collected:

1. Source of Data

- Public Datasets:
  - BCCD Dataset (Blood Cell Count and Detection): Contains labeled images of four main types of white blood cells – neutrophils, eosinophils, monocytes, and lymphocytes.
  - ALL-IDB: Acute Lymphoblastic Leukemia Image Database, useful for detecting abnormal lymphocytes.
  - Kaggle Datasets: Various open-source blood smear image datasets are available, many with expert annotations.
- Clinical Collaborations *(optional for enhanced dataset)*:
  - Partnering with hospitals or pathology labs to obtain anonymized and ethically approved blood smear images.
  - Images captured using digital microscopes under controlled lighting and magnification settings.

Loading Data **1**

To load the dataset efficiently for a transfer learning model (e.g., using TensorFlow or PyTorch), follow these structured steps:

```
blood_cells/
 ├── train/
|    ├── eosinophil/
|    ├── lymphocyte/
|    ├── monocyte/
|   └── neutrophil/
 ├── val/
└── test/
```

Data Preprocessing

Handling Missing Values **3**

Type	Example	Handling Method
Missing Image Files	Image paths listed but files are missing	Remove from dataset
Corrupted Images	Images that cannot be opened/decoded	Detect and remove or log
Missing Labels	Images with no assigned class	Discard or manually label
Incomplete Metadata	Missing info like date, magnification, etc. (if used)	Fill with default or remove

Handling Categorical values **2**

In image classification tasks like HematoVision, **categorical values refer to the class labels** of each image, such as:

- eosinophil
- lymphocyte
- monocyte
- neutrophil

These categories must be converted to **numerical format** before feeding them into a model.

Sprint 2 (5 Days)

Model Building

Model Building **5**

Steps Involved

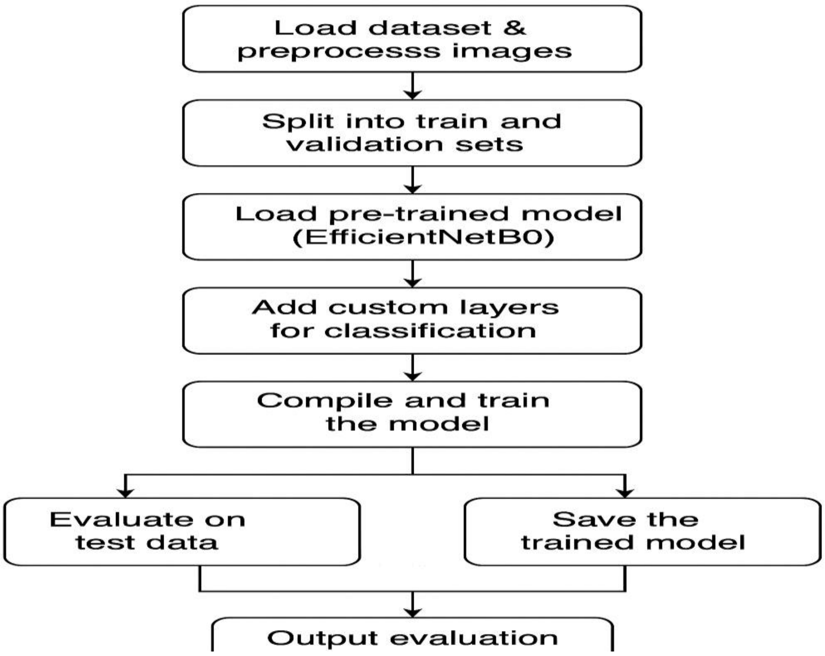
1. Import Libraries
2. Load Dataset & Preprocess Images
3. Split into Train and Validation Sets
4. Load Pre-trained Model (EfficientNetB0)
5. Add Custom Layers for Classification
6. Compile and Train the Model
7. Evaluate and Save the Model

modelbuilding.py:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
import os

# 1. Image Preprocessing and Augmentation
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_generator = train_datagen.flow_from_directory(
    'BloodCellDataset',  # folder with subfolders RBC, WBC, Platelets, etc.
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
val_generator = train_datagen.flow_from_directory(
    'BloodCellDataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
# 2. Load Pre-trained Model (EfficientNetB0)
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze base model layers
# 3. Add Custom Layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
# 4. Compile Model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
# 5. Train Model
model.fit(train_generator, validation_data=val_generator, epochs=10)
# 6. Save the Trained Model
model.save("blood_cell_model.h5")
```

Testing Model **3**



Deployment

Working HTML Pages **3**

Home.html:

```
<!DOCTYPE html>

<html>

<head>

  <title>Blood Cell Classifier</title>

</head>

<body>

  <h1>Upload a Blood Cell Image</h1>

  <form action="/predict" method="POST" enctype="multipart/form-data">

    <input type="file" name="file" required>

    <input type="submit" value="Predict">

  </form>

</body>

</html>
```

Result.html:

```
<!DOCTYPE html>

<html>

<head>

    <title>Prediction Result</title>

</head>

<body>

    <h1>Prediction Result</h1>

    <p><strong>Predicted Class:</strong> {{ prediction }}</p>

    <img src="{{ image_path }}" alt="Uploaded Image" width="300">

    <br><br>

    <a href="/">Try Another Image</a>

</body>

</html>
```

Flask deployment **5**

App.py:

```
from flask import Flask, render_template, request

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing import image

import numpy as np

import os

app = Flask(__name__)

model = load_model('blood_cell_model.h5')

class_names = ['Platelets', 'RBC', 'WBC']  # adjust to match your model

@app.route('/')

def index():

    return render_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():
    if 'file' not in request.files:
        return 'No file uploaded', 400
    file = request.files['file']
    if file.filename == '':
        return 'No file selected', 400
    filepath = os.path.join('static', 'upload.jpg')
    file.save(filepath)
    img = image.load_img(filepath, target_size=(224, 224))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]
    return render_template('index.html', prediction=predicted_class, img_path=filepath)
if __name__ == '__main__':
    app.run(debug=True)
```

Total Story Points

Sprint 1 = 8

Sprint 2 = 16

Velocity= Total Story Points Completed/ Number of Sprints

Total story Points= 16+8 =24

No of Sprints= 2

Velocity = (16+8)/2= 24/2

12 (Story Points per Sprint)

Your team’s velocity is 12 Story Points per Sprint.