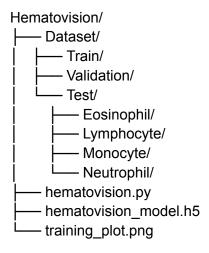
## HematoVision: Advanced Blood Cell Classification Using Transfer Learning

Date	27 July 2025
Track	Artificial intelligence and machine learning
Team id	LTVIP2025TMID32563
Project title	HematoVision: Advanced Blood Cell Classification Using Transfer Learning

## Complete Coding Steps Document

## Project Folder Structure



## **Coding**

# Import required libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification\_report, confusion\_matrix

import tensorflow as tf from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2 from tensorflow.keras.models import Model from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout from tensorflow.keras.optimizers import Adam

```
# Define constants
IMAGE SIZE = 224
BATCH SIZE = 32
EPOCHS = 10
NUM CLASSES = 4
# Dataset directory structure
TRAIN DIR = "/path/to/train"
VALID_DIR = "/path/to/valid"
# Image Preprocessing
train_datagen = ImageDataGenerator(
  rescale=1./255,
  rotation_range=15,
  zoom range=0.2,
  horizontal flip=True
)
valid datagen = ImageDataGenerator(rescale=1./255)
train data = train datagen.flow from directory(
  TRAIN_DIR,
  target_size=(IMAGE_SIZE, IMAGE_SIZE),
  batch size=BATCH SIZE,
  class_mode='categorical'
)
valid_data = valid_datagen.flow_from_directory(
  VALID DIR,
  target_size=(IMAGE_SIZE, IMAGE_SIZE),
  batch size=BATCH SIZE,
  class_mode='categorical'
# Load MobileNetV2 without top layer
base model = MobileNetV2(weights='imagenet', include top=False,
input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
# Freeze base model
```

```
base model.trainable = False
# Add custom layers
x = base model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
predictions = Dense(NUM_CLASSES, activation='softmax')(x)
model = Model(inputs=base model.input, outputs=predictions)
# Compile model
model.compile(optimizer=Adam(learning rate=0.0001), loss='categorical crossentropy',
metrics=['accuracy'])
# Train the model
history = model.fit(
  train_data,
  validation data=valid data,
  epochs=EPOCHS
)
# Evaluate the model
loss, accuracy = model.evaluate(valid data)
print(f"Validation Accuracy: {accuracy*100:.2f}%")
# Plot training results
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val accuracy'], label='Val Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy over Epochs')
plt.show()
# Predict on validation data
y pred = model.predict(valid data)
y_pred_classes = np.argmax(y_pred, axis=1)
y true = valid data.classes
# Confusion Matrix
cm = confusion matrix(y true, y pred classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=train_data.class_indices,
yticklabels=train data.class indices)
plt.xlabel('Predicted')
```

```
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

# Classification Report print("Classification Report:") print(classification\_report(y\_true, y\_pred\_classes, target\_names=list(train\_data.class\_indices.keys())))