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*

```
Hematovision/
├── Dataset/
│   ├── Train/
│   ├── Validation/
│   └── Test/
│       ├── Eosinophil/
│       ├── Lymphocyte/
│       ├── Monocyte/
│       └── Neutrophil/
├── hematovision.py
├── hematovision_model.h5
└── training_plot.png
```

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())))
```