HEMOTOVISION:ADVANCED BLOOD CELL CLASSIFICATION USING TRANSFER LEARNING.

Final Report

1. INTRODUCTION

1.1 Project Overview

The project focuses on the automated classification of white blood cells (WBCs) using deep learning techniques to assist in hematological analysis. The EfficientNetB0 model was used for the classification of four types of WBCs: Eosinophil, Lymphocyte, Monocyte, and Neutrophil.

1.2 Purpose

The purpose of this project is to develop an efficient and accurate deep learning model that can support pathologists in diagnosing diseases by classifying WBCs in peripheral blood smear images.

2. IDEATION PHASE

2.1 Problem Statement

Manual classification of WBCs is time-consuming, error-prone, and requires expert knowledge. An automated system can significantly speed up and standardize the diagnostic process.

2.2 Empathy Map Canvas

- Says: Pathologists need faster, more consistent diagnostic tools.

- Thinks: Technology can reduce workload and human error.

- Does: Manually reviews blood smear slides.

- Feels: Overwhelmed by repetitive tasks; wants reliable automation.

2.3 Brainstorming

Ideas considered included various CNN architectures, image preprocessing techniques, and data augmentation strategies. EfficientNetB0 was selected for its balance of performance and computational efficiency.

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Users (pathologists/lab technicians) upload WBC images to the system, which then classifies them and provides results for review.

3.2 Solution Requirement

- High classification accuracy

- Low inference time

- Robustness to variations in slide preparation and imaging

3.3 Data Flow Diagram

User -> Image Upload -> Preprocessing -> Model Prediction -> Result Display

3.4 Technology Stack

- Language: Python

- Libraries: TensorFlow, Keras, NumPy

- Hardware: GPU-enabled system (e.g., Kaggle environment)

- Dataset: WBC images organized by class

4. PROJECT DESIGN

4.1 Problem Solution Fit

The proposed system aligns with the problem by automating WBC classification using an EfficientNet-based CNN model.

4.2 Proposed Solution

EfficientNetB0 model trained on augmented image data for classifying WBCs into four classes. Includes callbacks like early stopping, learning rate reduction, and model checkpointing.

4.3 Solution Architecture

1. Image Input

2. Preprocessing and Augmentation

3. EfficientNetB0 Base Model

4. Global Average Pooling + Dense Layers

5. Softmax Output

6. Evaluation and Saving of Best Model

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

- Week 1: Dataset preparation and augmentation

- Week 2: Model architecture and training setup

- Week 3: Training, validation, tuning

- Week 4: Evaluation, reporting, and final submission

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

- Final model accuracy:

  - Train Accuracy: ~0.99

  - Validation Accuracy: ~0.96

  - Test Accuracy: ~0.95

- Loss and accuracy were monitored using callbacks and plotted for analysis.

7. RESULTS

7.1 Output Screenshots

- Model training graph (accuracy/loss)

- Directory structure and dataset overview

- Evaluation metrics printed after model training

8. ADVANTAGES & DISADVANTAGES

Advantages:

- High accuracy

- Automated diagnosis support

- Scalable and adaptable model

Disadvantages:

- Requires good quality images

- Model performance may vary with different staining protocols

9. CONCLUSION

This project demonstrates the successful use of a deep learning model (EfficientNetB0) for classifying WBCs. The system achieved high accuracy and can be used as a diagnostic aid.

10. FUTURE SCOPE

- Extension to more WBC types and disease states

- Integration into clinical decision support systems

- Real-time deployment on mobile or web platforms

11. APPENDIX

- Source Code

```
import os

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import EfficientNetB0

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau


# Configuration

dataset_dir = '/kaggle/input/newwwwwwwwww/dataset2-master/dataset2-
master/images'
```

```
train_dir = os.path.join(dataset_dir, 'TRAIN')
test_dir = os.path.join(dataset_dir, 'TEST')

img_width, img_height = 224, 224  # Increased for better feature extraction
batch_size = 32
epochs = 40
classes = ['EOSINOPHIL', 'LYMPHOCYTE', 'MONOCYTE', 'NEUTROPHIL']
num_classes = len(classes)

# Data Generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=25,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

val_test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
```

```
    class_mode='categorical',
    subset='training',
    shuffle=True
)

val_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Model
def build_efficientnet():
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=(img_width, img_height, 3))
```

```
    base_model.trainable = True  # Allow fine-tuning
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.4)(x)
    output = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer=Adam(1e-4), loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

model = build_efficientnet()

# Callbacks
checkpoint = ModelCheckpoint("hemato_best_model.h5", monitor='val_accuracy',
save_best_only=True, mode='max')
early_stop = EarlyStopping(monitor='val_loss', patience=6, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=3, verbose=1)

# Training
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=val_generator,
    validation_steps=val_generator.samples // batch_size,
    epochs=epochs,
```

```
    callbacks=[checkpoint, early_stop, reduce_lr]
)

# Load Best Weights
model.load_weights("hemato_best_model.h5")

# Final Evaluation
train_loss, train_acc = model.evaluate(train_generator)
val_loss, val_acc = model.evaluate(val_generator)
test_loss, test_acc = model.evaluate(test_generator)

print("\n Final Model Accuracy:")
print(f"Train Accuracy: {train_acc:.4f}")
print(f"Val Accuracy:   {val_acc:.4f}")
print(f"Test Accuracy:  {test_acc:.4f}")

# Save Final Model
model.save("hemato_final_model.h5")
```

- Dataset Link: https://www.kaggle.com/datasets/paultimothymooney/blood-cells

- GitHub & Project Demo Link: https://github.com/Nehapriya1718/Hematovision

https://github.com/Nehapriya1718/Hematovision.git