DL-Blood Cancer Detection Model

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import VGG16

from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.callbacks import ReduceLROnPlateau

import matplotlib.pyplot as plt

# Dataset path

dataset\_path = r"C:\Users\renee\Downloads\archive\dataset2-master\dataset2-master\images\TRAIN"

# Image parameters

IMG\_SIZE = (224, 224)

BATCH\_SIZE = 32

# Enhanced Data Augmentation

datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=30,   # Increased rotation range

    width\_shift\_range=0.3,

    height\_shift\_range=0.3,

    shear\_range=0.3,

    zoom\_range=0.4,      # Stronger zoom

    brightness\_range=[0.7, 1.3],  # Adjust brightness more

    horizontal\_flip=True,

    validation\_split=0.2  # 20% validation split

)

# Load training data

train\_data = datagen.flow\_from\_directory(

    dataset\_path, target\_size=IMG\_SIZE, batch\_size=BATCH\_SIZE,

    class\_mode='categorical', subset='training'

)

# Load validation data

val\_data = datagen.flow\_from\_directory(

    dataset\_path, target\_size=IMG\_SIZE, batch\_size=BATCH\_SIZE,

    class\_mode='categorical', subset='validation'

)

# Load Pretrained VGG16 model (Transfer Learning)

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Unfreeze last 10 layers for fine-tuning

for layer in base\_model.layers[:-10]:

    layer.trainable = False

for layer in base\_model.layers[-10:]:

    layer.trainable = True

# Custom classifier with Batch Normalization

x = Flatten()(base\_model.output)

x = Dense(512, activation='relu')(x)

x = BatchNormalization()(x)  # Normalize activations for better learning

x = Dropout(0.2)(x)  # Higher dropout to reduce overfitting

x = Dense(len(train\_data.class\_indices), activation='softmax')(x)  # Output layer

# Compile Model with RMSprop

optimizer = RMSprop(learning\_rate=1e-4, decay=1e-6)  # Better optimizer for transfer learning

model = Model(inputs=base\_model.input, outputs=x)

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

# Learning rate scheduler

lr\_scheduler = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=2, verbose=1)

# Train Model for 10 Epochs

history = model.fit(train\_data, validation\_data=val\_data, epochs=10, callbacks=[lr\_scheduler])

# Evaluate Model

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

# Plot accuracy

plt.plot(acc, label="Training Accuracy")

plt.plot(val\_acc, label="Validation Accuracy")

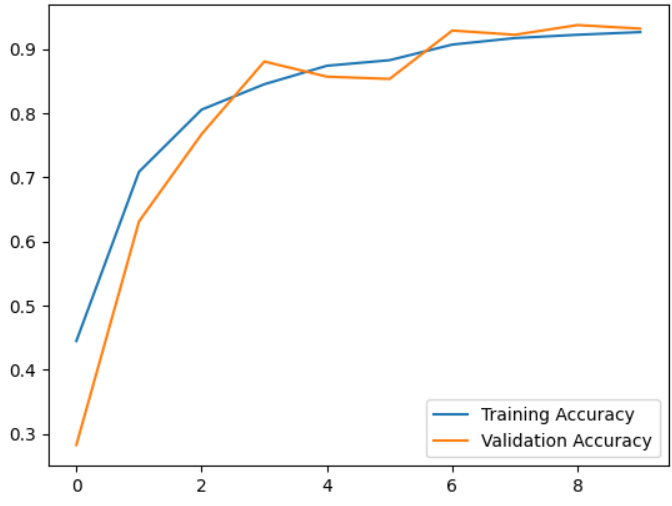
plt.legend()

plt.show()

# Save Model

model.save("blood\_cell\_cnn\_fast.h5")

print("Final Validation Accuracy:", max(val\_acc) \* 100, "%")



# Plot Loss Curve

loss = history.history['loss']

val\_loss = history.history['val\_loss']

plt.figure(figsize=(8, 6))

plt.plot(loss, label="Training Loss")

plt.plot(val\_loss, label="Validation Loss")

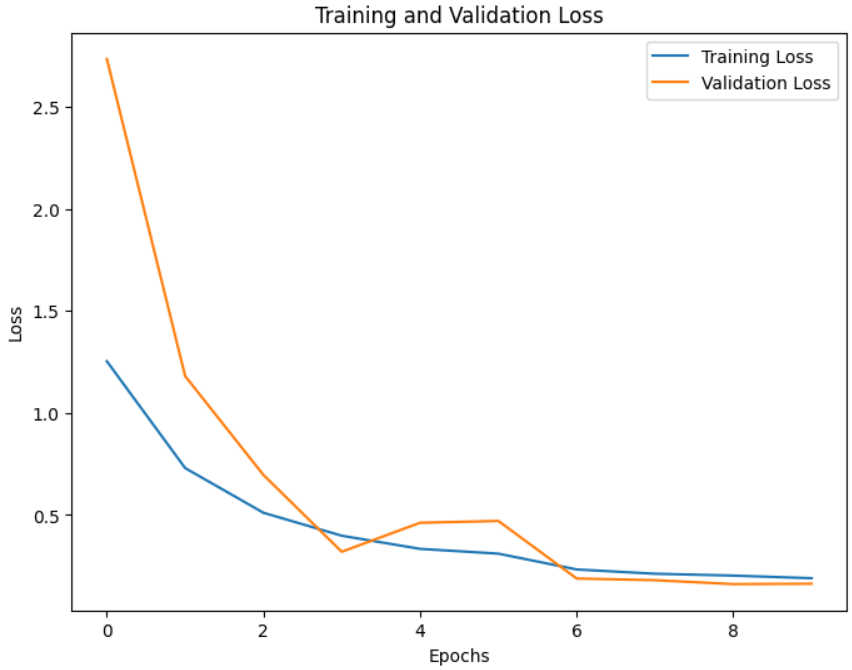
plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.title("Training and Validation Loss")

plt.show()



import numpy as np

import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Predict validation set

y\_pred = model.predict(val\_data)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_true = val\_data.classes