CODE :

import kagglehub

import os

import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

import cv2

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50

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

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

# Download the dataset

path = kagglehub.dataset\_download("shuvokumarbasak2030/medical imaging-fetal-colorized-new-dataset-umrict")

print("Path to dataset files:", path)

# Set the dataset path

dataset\_path = "/root/.cache/kagglehub/datasets/shuvokumarbasak2030/medical-imaging fetal-colorized-new-dataset-umrict/versions/1/2d fetal ultrasound images/2d fetal ultrasound images/data"

# Check if dataset path exists

if not os.path.exists(dataset\_path):

raise FileNotFoundError(f"Dataset path not found: {dataset\_path}")

# Set image size and batch size

IMG\_SIZE = (224, 224)

BATCH\_SIZE = 32

# Set up the image data generator for preprocessing

datagen = ImageDataGenerator(

rescale=1.0/255.0,

validation\_split=0.2

)

# Training and validation data generators

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

dataset\_path,

target\_size=IMG\_SIZE,

batch\_size=BATCH\_SIZE,

class\_mode="categorical",

subset="training"

)

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

dataset\_path,

target\_size=IMG\_SIZE,

batch\_size=BATCH\_SIZE,

class\_mode="categorical",

subset="validation"

)

# Print number of samples in each set

print(f"Train dataset: {train\_generator.samples} images")

print(f"Validation dataset: {val\_generator.samples} images")

# Load ResNet50 model without the top layer

IMG\_SIZE\_INT = IMG\_SIZE[0]

base\_model = ResNet50(weights="imagenet", include\_top=False, input\_shape=(IMG\_SIZE\_INT, IMG\_SIZE\_INT, 3))

base\_model.trainable = False # Freeze the base model layers

# Add custom top layers

x = GlobalAveragePooling2D()(base\_model.output)

x = Dense(256, activation="relu")(x)

x = Dropout(0.4)(x)

x = Dense(128, activation="relu")(x)

x = Dropout(0.3)(x)

output = Dense(train\_generator.num\_classes, activation="softmax")(x)

# Compile the final model

model = Model(inputs=base\_model.input, outputs=output) model.compile(optimizer=Adam(learning\_rate=0.0001), loss="categorical\_crossentropy", metrics=["accuracy"])

# Display model summary

model.summary()

# Train the model

history = model.fit(

train\_generator,

validation\_data=val\_generator,

epochs=10

)

# Plot accuracy graphs

plt.plot(history.history["accuracy"], label="Train Accuracy")

plt.plot(history.history["val\_accuracy"], label="Validation Accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.title("Training & Validation Accuracy")

plt.legend() plt.show()

# Function to predict image

def predict\_image(image\_path):

img = cv2.imread(image\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) 10

img = cv2.resize(img, (IMG\_SIZE\_INT, IMG\_SIZE\_INT))

img = np.expand\_dims(img / 255.0, axis=0).astype(np.float32)

prediction = model.predict(img)

class\_idx = np.argmax(prediction)

confidence = np.max(prediction) \* 100

return f"Predicted Class: {train\_generator.class\_indices[class\_idx]} (Confidence: {confidence:.2f}%)"

# Test prediction with a sample image

print(predict\_image("/content/Patient00168\_Plane6\_1\_of\_4\_10.jpg"))