**On Google Colab, using the ResNet50 model:**

**Step 1: Set Up the Environment**

Open Google Colab:

Navigate to Google Colab.

Create a new Python 3 notebook.

Install Required Libraries:

If not already installed, install TensorFlow, Keras, and other necessary libraries by running the following cell:

!pip install tensorflow keras matplotlib pillow

Mount Google Drive :

If your dataset or other files are stored on Google Drive, mount it to your Colab runtime for easy access:

from google.colab import drive

drive.mount('/content/drive')

**Step 2: Prepare the Data**

1. Upload Dataset:

Upload your dataset to Google Drive or directly to Colab if it's not too large.

1. Define Directories:

Define paths to your training and validation datasets (train\_dir and val\_dir).

**Step 3: Build and Train the Mode**l

1. Load Necessary Libraries:

Import TensorFlow and necessary components:

import tensorflow as tf

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

1. Define Paths to Directories:

After uploading or mounting your dataset, define paths (train\_dir and val\_dir) to the directories containing your training and validation data.

train\_dir = '/content/Dataset/train'

val\_dir = '/content/Dataset/test'

**train\_dir:** Path to the directory containing training images or data files.

**val\_dir:** Path to the directory containing validation images or data files.

1. Prepare Data Generators:

Set up data generators for training and validation data with augmentation:

train\_datagen = ImageDataGenerator(

    preprocessing\_function=tf.keras.applications.resnet50.preprocess\_input,

    rotation\_range=40,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest')

val\_datagen = ImageDataGenerator(

    preprocessing\_function=tf.keras.applications.resnet50.preprocess\_input)

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

    train\_dir,

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical')

validation\_generator = val\_datagen.flow\_from\_directory(

    val\_dir,

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical')

1. Build the Model:

Define the ResNet50 base model

base\_model = ResNet50(weights='imagenet',

                      include\_top=False,

                      input\_shape=(224, 224, 3))

x = base\_model.output

x = GlobalAveragePooling2D()(x)

predictions = Dense(len(train\_generator.class\_indices), activation='softmax')(x)

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

1. Compile the Model:

Compile the model with appropriate loss function, optimizer, and metrics:

model.compile(loss='categorical\_crossentropy',

              optimizer=tf.keras.optimizers.Adam(1e-5),

              metrics=['accuracy'])

1. Train the Model:

Train the model on the training data using the prepared data generators:

history = model.fit(

  train\_generator,

  epochs=40,

  validation\_data=validation\_generator)

1. Evaluate the Model:

Evaluate the trained model on the validation data and print the accuracy:

results = model.evaluate(validation\_generator)

accuracy\_percentage = results[1] \* 100

print("Accuracy: {:.2f}%".format(accuracy\_percentage))

**Step 4: Convert to TensorFlow Lite and Download**

1. Convert to TensorFlow Lite:

Convert the trained Keras model to TensorFlow Lite format:

converter = tf.lite.TFLiteConverter.from\_keras\_model(model= model)

model\_tflite = converter.convert()

open("a7modh1el.tflite", "wb").write(model\_tflite

1. Save and Download the TensorFlow Lite Model:

Save the TensorFlow Lite model to a file and download it to your local computer:

# Download the TFLite model to your local computer.

from google.colab import files

files.download('/content/a7modh1el.tflite')

**Step-by-Step Guide to Run and Use the VGG16 Model**

**Step 1: Set Up the Environment**

Open Google Colab:

Navigate to Google Colab.

Create a new Python 3 notebook (File -> New Python 3 notebook).

Install Required Libraries:

If not already installed, install TensorFlow, Keras, and scikit-learn for evaluation metrics:

!pip install tensorflow keras scikit-learn

Mount Google Drive:

If your dataset or files are stored in Google Drive, mount it to your Colab runtime:

from google.colab import drive

drive.mount('/content/drive')

**Step 2: Load and Prepare the Model**

Load Necessary Libraries:

import os

import numpy as np

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications.vgg16 import VGG16, preprocess\_input

from sklearn.metrics import accuracy\_score, f1\_score, classification\_report

Define Dataset Path and Parameters:

dataset\_path = 'Animals\_Dataset/dataset'

img\_size = (224, 224)

batch\_size = 32

Set Up Image Data Generator:

Initialize the ImageDataGenerator with preprocessing function for VGG16:

data\_generator = ImageDataGenerator(preprocessing\_function=preprocess\_input)

**Step 3: Build and Compile the Model**

Load Pre-trained VGG16 Model:

Load VGG16 model with ImageNet weights (excluding the top layers):

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

Build the Classification Model:

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

model = Sequential()

model.add(base\_model)

model.add(GlobalAveragePooling2D())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(11, activation='softmax'))

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

**Step 4: Train the Model**

Set Up Data Generators:

Create generators for training and validation data:

generator = data\_generator.flow\_from\_directory(

    dataset\_path,

    target\_size=img\_size,

    batch\_size=batch\_size,

    class\_mode='categorical',

    shuffle=True

)

Train the Model:

Train the model on the dataset:

steps\_per\_epoch = np.ceil(generator.samples / batch\_size)

history = model.fit(

    generator,

    steps\_per\_epoch=steps\_per\_epoch,

    epochs=10,

    verbose=1

)

Evaluate the Model:

Evaluate the trained model on the validation set (optional):

generator.reset()

y\_true = generator.classes

y\_pred\_probs = model.predict(generator, steps=steps\_per\_epoch, verbose=1)

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

accuracy = accuracy\_score(y\_true, y\_pred)

f1 = f1\_score(y\_true, y\_pred, average='weighted')

print(f'Accuracy: {accuracy}')

print(f'F1 Score: {f1}')

print('\nClassification Report:')

print(classification\_report(y\_true, y\_pred))

**Step 5: Save and Use the Trained Model**

Save the Model:

Save the trained model for future use:

model.save('animal\_classifier\_model.h5')

**The Data Augmentation Code's Usage Procedure**

1. **Set Up Your Project Environment**

Create a Project Directory:

Create a new directory for your project. This is where you'll organize your scripts and datasets.

Install Required Packages:

You'll need to install the necessary packages:

pip install albumentations pillow tqdm

**albumentations:** Library for image augmentations.

**PIL:** Python Imaging Library for handling images.

**tqdm:** Library for showing progress bars during augmentation.

1. **Write the Augmentation Script**

Create a Python script (augmentation\_script.py) in your project directory.

import os

import random

from shutil import copy2

from tqdm import tqdm

from PIL import Image  # To handle image operations

import albumentations as A  # Library for augmentations

# Function to perform augmentations on a given image

def augment\_image(image\_path, output\_path, augmentations, new\_image\_name):

    """

    Applies a series of augmentations to an image and saves the result.

    Args:

        image\_path (str): Path to the original image.

        output\_path (str): Path to save the augmented image.

        augmentations (A.Compose): Albumentations Compose object defining the augmentations.

        new\_image\_name (str): Name for the augmented image.

    """

    img = Image.open(image\_path)

    img = img.convert("RGB")

    # Apply augmentations

    img\_aug = augmentations(image=np.array(img))["image"]

    # Save the augmented image

    output\_file\_path = os.path.join(output\_path, new\_image\_name)

    Image.fromarray(img\_aug).save(output\_file\_path)

    return output\_file\_path

# Example usage of augmentation function

def main():

    input\_image\_path = "path/to/your/input/image.jpg"

    output\_folder = "path/to/your/output/folder"

    output\_image\_name = "augmented\_image.jpg"

    augmentations = A.Compose([

        A.HorizontalFlip(p=0.5),

        A.RandomRotate90(p=0.5),

        A.RandomBrightnessContrast(p=0.5),

        A.GaussianBlur(p=0.5),

        A.HueSaturationValue(p=0.5),

    ])

    augmented\_image\_path = augment\_image(input\_image\_path, output\_folder, augmentations, output\_image\_name)

    print(f"Augmented image saved at: {augmented\_image\_path}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**3. Modify Paths and Inputs**

Modify Paths:

Replace "path/to/your/input/image.jpg" with the path to an image in your local filesystem that you want to augment. Update "path/to/your/output/folder" to the directory where you want to save augmented images.

Modify Augmentation Parameters:

Adjust the augmentations list in A.Compose([...]) to include or modify the augmentation techniques as per your requirements.

**4. Run the Script**

**5. Verify Output**

Check Output:

After running the script, verify that the augmented image was saved correctly in the specified output folder (output\_folder). Check if the augmentations applied match your expectations.

**Access and Download the Dataset**

**Dataset Access:**

Visit the Google Drive link <https://drive.google.com/drive/folders/1e6FeP0E7vN7TXirRpvDaoCXIBrpSdxos>

**Download Dataset:**

Download the dataset folder (Animals in this case) to your local machine. This folder should contain subdirectories like images and labels corresponding to images and their annotations respectively**.**

**YOLOv8 Model Training Setup and Execution on Windows**

**Step 1: Install Visual Studio Code (VS Code):**

1. Download Visual Studio Code:

* Visit the official Visual Studio Code download page.
* Download the installer appropriate for your operating system.

1. Install Visual Studio Code:

Run the downloaded installer and follow the on-screen instructions to complete the installation.

**Step 2: Install Python and Pip**

Download Python:

Go to the Python download page.

Download the latest Python installer for your operating system.

Install Python:

Windows:

Run the installer and ensure the "Add Python to PATH" option is checked.

macOS and Linux:

Follow the installer prompts. On macOS.

Follow the on-screen instructions to complete the installation.

**Step 3: Install Required Libraries**

Install PyTorch:

Visit the PyTorch website for installation instructions specific to your system.

Install Ultralytics YOLOv8:

Run the command:

pip install ultralytics

Install Additional Dependencies:

Run the command:

pip install ultralytics

**Step 4: Prepare Your Dataset**

Ensure Dataset Structure:

Organize your dataset as specified in “dataset.yaml”

Verify dataset.yaml:

Ensure the paths in your dataset.yaml file correctly point to the dataset directories.

train: D:\Animails\split\_dataset\train

val: D:\Animails\split\_dataset\val

test: D:\Animails\split\_dataset\test

nc: 10

names: [Horse', 'Monkey', 'Cattle', 'Deer', 'Rabbit', 'Goat', 'Spider', 'Mule', 'Hamster', 'Koala']

**Step 5: Create and Execute the Training Script**

Create a Python Script:

In VS Code, create a new file named train\_yolov8.py.

Copy and paste the following code into train\_yolov8.py:

from ultralytics import YOLO

import torch

if \_\_name\_\_ == '\_\_main\_\_':

    # Ensure GPU is available

    device = "cuda" if torch.cuda.is\_available() else "cpu"

    # Define training parameters

    epochs = 100  # Reduced number of epochs for faster training

    img\_size = 416  # Reduced image size to lower memory usage

    batch\_size = 4  # Further reduced batch size for lower memory usage

    workers = 2  # Reduced number of data loader workers

    # YOLO model setup

    yolo\_model = YOLO("yolov8n.yaml")  # YOLOv8n (smallest variant) for speed

    # Additional training configurations

    train\_params = {

        "data": "D:\\Animails\\dataset.yaml",

        "epochs": epochs,

        "imgsz": img\_size,

        "batch": batch\_size,

        "device": device,

        "workers": workers,

        "patience": 5,  # Early stopping

        "val": True,  # Enable validation

        "name": "yolov8n\_experiment",  # Experiment name for easier tracking

        "verbose": True,  # Output more detailed information during training

    }

    print(train\_params)

    # Mixed precision training for speed

    torch.backends.cudnn.benchmark = True  # Enable cuDNN autotuning

    # Start training

    results = yolo\_model.train(\*\*train\_params)

    # Print results

print(results)

Run the Training Script