```python
# -*- coding: utf-8 -*-
"""Animal_Classification final.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1FqhL_u5f_ivomLqX5TO5VzYwlwmiPU2z
"""

from google.colab import files

uploaded = files.upload()

for filename in uploaded.keys():
  print(f'User uploaded file "{filename}"')

import zipfile

zip_file_name = list(uploaded.keys())[0]

with zipfile.ZipFile(zip_file_name, 'r') as zip_ref:
  zip_ref.extractall('Animal_Classifier')

import os
from collections import Counter

data_dir = '/content/Animal_Classifier/Animal Classification/dataset' # Point to the directory containing class folders
classes = os.listdir(data_dir)
class_counts = {cls: len(os.listdir(os.path.join(data_dir, cls))) for cls in classes}
print(class_counts)

from torchvision import datasets
import torchvision.transforms as transforms
from torchvision.transforms import ToTensor

train_data=datasets.FashionMNIST(root="Animal_Classifier/data/raw",
                                 train=True,
                                 download=True,
                                 transform=transforms.ToTensor(),
                                 target_transform=None)
test_data=datasets.FashionMNIST(
    root="Animal_Classifier/data/raw",
    train=True,
    download=True,
    transform=ToTensor(),
    target_transform=None
)

import os

data_dir_dataset = '/content/Animal_Classifier/Animal Classification/dataset'
print(os.listdir(data_dir_dataset))

from torch.utils.data import random_split

train_size = int(0.7 * len(animal_dataset))
val_size = int(0.15 * len(animal_dataset))
test_size = len(animal_dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(animal_dataset, [train_size, val_size, test_size])

print(f"Training dataset size: {len(train_dataset)}")
print(f"Validation dataset size: {len(val_dataset)}")
print(f"Testing dataset size: {len(test_dataset)}")

from torchvision.datasets import ImageFolder
from torchvision.transforms import Compose, Resize, ToTensor


transform = Compose([
    Resize((64, 64)),
    ToTensor()
])


animal_dataset = ImageFolder(root=data_dir_dataset, transform=transform)

print(f"Total number of images in the dataset: {len(animal_dataset)}")

from torch.utils.data import DataLoader

batch_size = 32

train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

print(f"Number of batches in train_dataloader: {len(train_dataloader)}")
print(f"Number of batches in val_dataloader: {len(val_dataloader)}")
print(f"Number of batches in test_dataloader: {len(test_dataloader)}")

train_images, train_labels = next(iter(train_dataloader))
print(f"Train batch image shape: {train_images.shape}")
print(f"Train batch label shape: {train_labels.shape}")


val_images, val_labels = next(iter(val_dataloader))
print(f"Validation batch image shape: {val_images.shape}")
print(f"Validation batch label shape: {val_labels.shape}")


test_images, test_labels = next(iter(test_dataloader))
print(f"Test batch image shape: {test_images.shape}")
print(f"Test batch label shape: {test_labels.shape}")

import matplotlib.pyplot as plt
import random
import torch

plt.figure(figsize=(5, 5))
random.seed(42)
random_index = random.randint(0, len(animal_dataset) - 1)
image, label = animal_dataset[random_index]

if isinstance(image, torch.Tensor):
    image = image.permute(1, 2, 0).numpy()
plt.imshow(image.squeeze(), cmap='gray')
if animal_dataset.classes and label < len(animal_dataset.classes):
    plt.title(f"Label: {animal_dataset.classes[label]}")
else:
    plt.title(f"Label: {label}")
plt.axis('off')
plt.show()
```

```
import os

file_to_remove = '/content/sample_data/anscombe.json'

if os.path.exists(file_to_remove):
    os.remove(file_to_remove)
    print(f"Removed file: {file_to_remove}")
else:
    print(f"File not found: {file_to_remove}")

import os

file_to_remove = '/content/sample_data/README.md'

if os.path.exists(file_to_remove):
    os.remove(file_to_remove)
    print(f"Removed file: {file_to_remove}")
else:
    print(f"File not found: {file_to_remove}")

from torchvision import datasets
import torchvision.transforms as transforms
from torchvision.transforms import ToTensor

train_data=datasets.FashionMNIST(root="Animal_Classifier/data/raw",
                                 train=True,
                                 download=True,
                                 transform=transforms.ToTensor(),
                                 target_transform=None)
test_data=datasets.FashionMNIST(
    root="Animal_Classifier/data/raw",
    train=True,
    download=True,
    transform=ToTensor(),
    target_transform=None
)

import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    random_index = random.randint(0, len(animal_dataset) - 1)
    image, label = animal_dataset[random_index]

    if isinstance(image, torch.Tensor):
        image = image.permute(1, 2, 0).numpy()

    plt.imshow(image.squeeze(), cmap='gray')
    if animal_dataset.classes and label < len(animal_dataset.classes):
        plt.title(f"Label: {animal_dataset.classes[label]}")
    else:
        plt.title(f"Label: {label}")
    plt.axis('off')
plt.show()

"""### Visualizing Sample Images

This section displays sample images from the dataset along with their corresponding labels. This visual inspection helps to understand the nature of the data and the a
"""

import os

data_dir_dataset = '/content/Animal_Classifier/Animal Classification/dataset'
print(os.listdir(data_dir_dataset))

from torch.utils.data import DataLoader

batch_size = 32

train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

print(f"Number of batches in train_dataloader: {len(train_dataloader)}")
print(f"Number of batches in val_dataloader: {len(val_dataloader)}")
print(f"Number of batches in test_dataloader: {len(test_dataloader)}")

train_images, train_labels = next(iter(train_dataloader))
print(f"Train batch image shape: {train_images.shape}")
print(f"Train batch label shape: {train_labels.shape}")


val_images, val_labels = next(iter(val_dataloader))
print(f"Validation batch image shape: {val_images.shape}")
print(f"Validation batch label shape: {val_labels.shape}")


test_images, test_labels = next(iter(test_dataloader))
print(f"Test batch image shape: {test_images.shape}")
print(f"Test batch label shape: {test_labels.shape}")

import matplotlib.pyplot as plt
import random
import torch

plt.figure(figsize=(5, 5))
random.seed(42)
random_index = random.randint(0, len(animal_dataset) - 1)
image, label = animal_dataset[random_index]

if isinstance(image, torch.Tensor):
    image = image.permute(1, 2, 0).numpy()
plt.imshow(image.squeeze(), cmap='gray')
if animal_dataset.classes and label < len(animal_dataset.classes):
    plt.title(f"Label: {animal_dataset.classes[label]}")
else:
    plt.title(f"Label: {label}")
plt.axis('off')
plt.show()

import os

data_path = '/content/Animal_Classifier/Animal Classification/dataset'
for folder in os.listdir(data_path):
    print(folder)

print("Folders after cleanup:")
print(os.listdir(data_path))

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

```python
train_generator = datagen.flow_from_directory(
    data_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = datagen.flow_from_directory(
    data_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

"""### Setting up TensorFlow Data Generators

This code uses TensorFlow's `ImageDataGenerator` to create data generators for training and validation. It includes data augmentation techniques to artificially increa
"""

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest',
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    data_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    data_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

import matplotlib.pyplot as plt
import numpy as np


images, labels = next(train_generator)


plt.figure(figsize=(10,10))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i])
    plt.axis("off")
plt.tight_layout()
plt.show()

"""### Visualizing Augmented Images

This section displays a batch of images from the training data generator after applying the data augmentation transformations. This helps visualize the effects of the
"""

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10
)

val_loss, val_acc = model.evaluate(val_generator)
print(f"Validation Loss: {val_loss}")
print(f"Validation Accuracy: {val_acc}")

"""### Evaluating the Model

This section evaluates the trained model's performance on the validation dataset and prints the validation loss and accuracy.
"""

import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense


model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
```

```python
    Dense(train_generator.num_classes, activation='softmax')
])


model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam

base_model = MobileNetV2(input_shape=(224, 224, 3),
                         include_top=False,
                         weights='imagenet')


base_model.trainable = False


x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

"""This code defines and compiles a deep learning model for image classification using transfer learning with the pre-trained MobileNetV2 model. The pre-trained model'

model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

base_model.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


fine_tune_history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=5
)

"""### Fine-tuning the Model

This section fine-tunes the entire model (including the pre-trained base) with a lower learning rate. This allows the model to learn more specific features from the ar
"""

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    brightness_range=[0.6, 1.4],
    fill_mode='nearest',
    validation_split=0.2
)

import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'] + fine_tune_history.history['accuracy'])
plt.plot(history.history['val_accuracy'] + fine_tune_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

plt.subplot(1,2,2)
plt.plot(history.history['loss'] + fine_tune_history.history['loss'])
plt.plot(history.history['val_loss'] + fine_tune_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.tight_layout()
plt.show()

model.save('animal_classifier_model.h5')

from google.colab import drive
drive.mount('/content/drive')

import os

drive_path = '/content/drive/My Drive/'
dataset_folder_name = 'Animal_Classifier_Dataset'
new_dataset_dir = os.path.join(drive_path, dataset_folder_name)

os.makedirs(new_dataset_dir, exist_ok=True)

print(f"Created dataset directory in Google Drive: {new_dataset_dir}")

import os


drive_dataset_path = '/content/drive/My Drive/Animal_Classifier_Dataset'


model_filename = 'animal_classifier_model.keras'


drive_model_path = os.path.join(drive_dataset_path, model_filename)

try:
    model.save(drive_model_path)
    print(f"Model successfully saved to Google Drive at: {drive_model_path}")
except Exception as e:
```

```
    print(f"Error saving the model to Google Drive: {e}")

"""### Saving the Trained Model to Google Drive

This code saves the trained model to a specified directory in Google Drive. This allows for persistent storage of the model and its future use without retraining.
"""

import os


drive_dataset_path = '/content/drive/My Drive/Animal_Classifier_Dataset'
model_filename = 'animal_classifier_model.keras'
drive_model_path = os.path.join(drive_dataset_path, model_filename)


if os.path.exists(drive_model_path):
    print(f"Model file found in Google Drive: {drive_model_path}")
else:
    print(f"Model file not found in Google Drive: {drive_model_path}")

"""## Conclusion

The model was trained for a total of 15 epochs (10 initial epochs and 5 fine-tuning epochs). The training and validation accuracy and loss were plotted to visualize th

After training, the model achieved a validation accuracy of approximately 90.6%. The training process involved using data augmentation to improve the model's generaliz
"""
```