Image Classification of Omnivores and Herbivores using the CIFAR-100 Dataset

Authors: Zach Riane I. Machacon, Josh B. Ratificar, Jahna Patricia Poticar

This project aims to classify images into two categories: omnivores and herbivores, utilizing the CIFAR-100 dataset. The CIFAR-100 dataset contains 60,000 32x32 color images in 100 classes, with each class containing 600 images. Here, we focus on distinguishing between animals that primarily consume plants (herbivores) and those that consume both plants and animals (omnivores). Through deep learning techniques and convolutional neural networks (CNNs), we seek to build a model capable of accurate classification among diverse species.

Import modules

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
```

Import the dataset

In [ ]:
```
(X_coarse_train, y_coarse_train), (X_coarse_test, y_coarse_test) = tf.keras.datasets.cifar100.load_data(
    label_mode='coarse'
)

(X_fine_train, y_fine_train), (X_fine_test, y_fine_test) = tf.keras.datasets.cifar100.load_data(
    label_mode='fine'
)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169001437/169001437 [==============================] - 4s 0us/step
```

Data Preprocessing

We will be extracting only the `large omnivores and herbivores` coarse class from the dataset along with their fine classes.

In [ ]:
```
# Extracting images from coarse dataset

coarse_train_index = (y_coarse_train == 11).reshape(X_coarse_train.shape[0]) # Returns the indices where y_coars
X_train = X_coarse_train[coarse_train_index] # Extracts the images from X_train with indices that match the cla

coarse_test_index = (y_coarse_test == 11).reshape(X_coarse_test.shape[0])
X_test = X_coarse_test[coarse_test_index]
```

Since Tensorflow returns CIFAR-100 as a tuple of NumPy arrays, we are able to index the `X_coarse_train` and `X_coarse_test` arrays with a boolean mask. `coarse_train_index` amd `coarse_test_index` creates a boolean mask where `y_coarse_train == 11`. It checks if each element in the array `y_coarse_train` is equal to 11, resulting in a boolean array. `reshape(X_coarse_train.shape[0])` reshapes the boolean array to match the shape of `X_coarse_train`, ensuring compatibility for indexing.

In [ ]:
```
# Extracting index of fine labels

labels_of_herbi_omni = []
label_mapping = {}

str_labels = ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle', 'bicycle', 'bottle',
              'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock', 'cloud', 'cock
              'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster', 'house', 'kangaroo', 'keyb
              'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse', 'mushroom', 'oak_tree', 'orange', 'orchid
              'plate', 'poppy', 'porcupine', 'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose', 's
              'snake', 'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table', 'tank', 'teleph
              'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman', 'worm']
herbi_omni = ['camel', 'cattle', 'chimpanzee', 'elephant', 'kangaroo']
print("Fine labels of Herbivores and Omnivores:")
for i, name in enumerate(str_labels):
  if name in herbi_omni:
    print(f"{i}: {name}")
    label_mapping[i] = herbi_omni.index(name)
```

```
    labels_of_herbi_omni.append(i)
```

```
Fine labels of Herbivores and Omnivores:
15: camel
19: cattle
21: chimpanzee
31: elephant
38: kangaroo
```

From the documentation of CIFAR-100, we were able to gather all possible fine classes of the entire dataset. The index of fine labels we needed were based on their position when all fine labels are arranged in alphabetical order. Here, we just extracted their indices and stored them in a dictionary called `label_mapping`. As for the labels themselves, they are stored in `labels_herbi_omni` for ease of use later in showing the images and their corresponding label.

In [ ]:
```
# Extracting fine labels of images from coarse indices

fine_train_index = np.isin(y_fine_train, labels_of_herbi_omni).reshape(X_fine_train.shape[0])
y_train = y_fine_train[fine_train_index]

fine_test_index = np.isin(y_fine_test, labels_of_herbi_omni).reshape(X_fine_test.shape[0])
y_test = y_fine_test[fine_test_index]
```

Since we're checking for more than 1 value for fine labels unlike coarse labels, we made use of the NumPy function `isin`. This returns a NumPy array of boolean values where it returns `True` if it is found in the `labels_of_herbi_omni` list. This in turn gives us as NumPy array back in `y_test` and `y_train` with our intended fine labels.

In [ ]:
```
# Convert fine labels from their original numbers to arbitrary categorical numbers from 0 - 4

y_train = np.array([label_mapping[label[0]] for label in y_train]).reshape(-1, 1)
y_test = np.array([label_mapping[label[0]] for label in y_test]).reshape(-1, 1)

# Convert training and test labels to categorical variables. This does one-hot encoding

y_train = tf.keras.utils.to_categorical(y_train, num_classes=5)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=5)
```

As shown in an earlier cell, the indices of the fine labels were all over the place and not in order. We converted these to fine labels' indices from 0 - 4 in order to make it easier to identify from the `labels_of_herbi_omni` list. These labels also went through one-hot encoding, which is a technique used in machine learning and data processing to represent categorical variables as binary vectors.

One advantage of one-hot encoding is its facilitation of effective model learning. With each input value represented as a binary vector, containing only one '1' and the rest '0's, the model can readily discern between different inputs. This distinction aids in learning the relationship between input values and target outputs, enhancing the model's effectiveness and ease of learning (Crêteur, 2022).

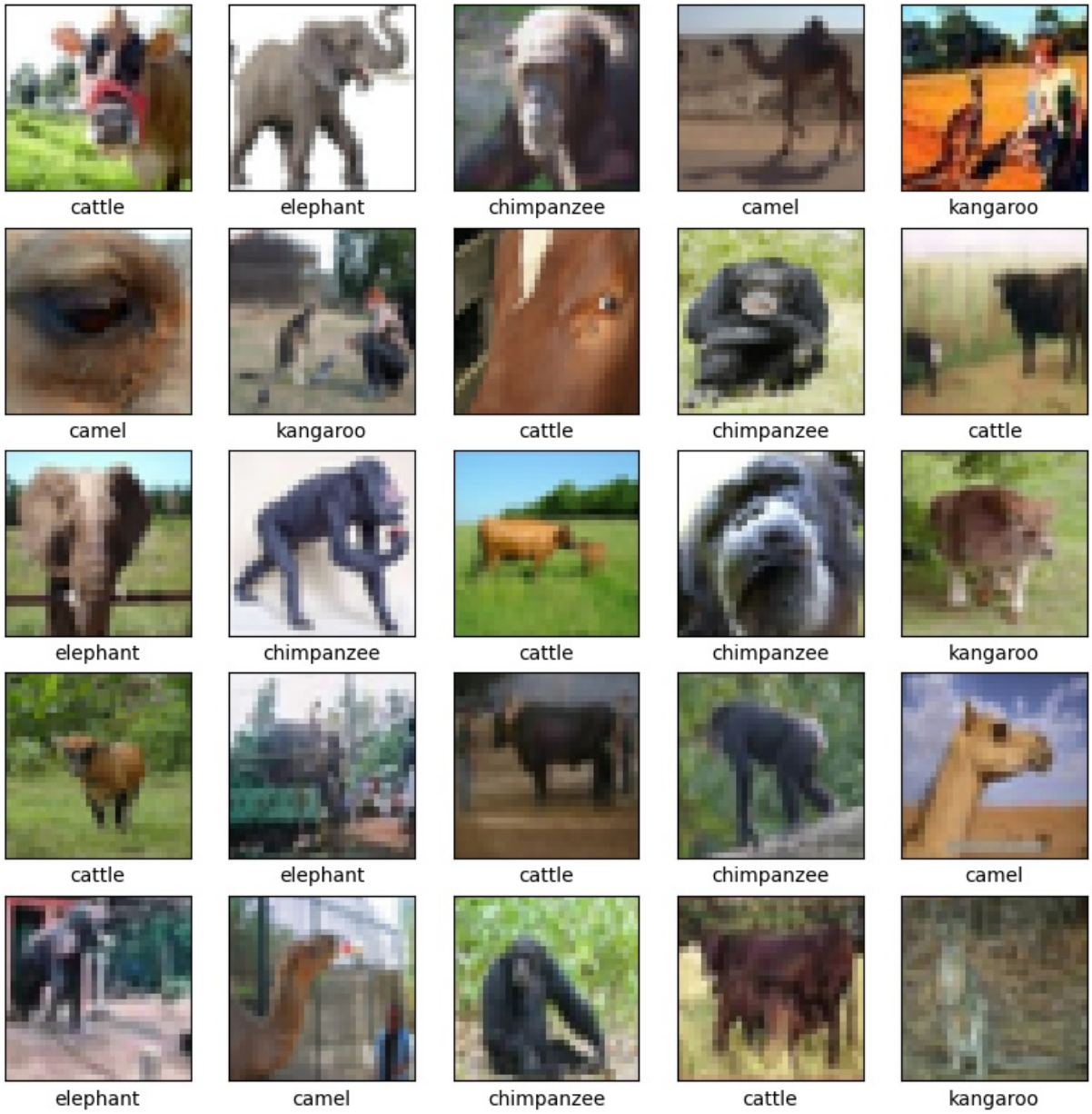
In [ ]:
```
def count_samples():
  # Count samples per class in training set
  train_counts = {label: np.sum(np.argmax(y_train, axis=1) == label) for label in range(y_train.shape[1])}

  # Count samples per class in testing set
  test_counts = {label: np.sum(np.argmax(y_test, axis=1) == label) for label in range(y_test.shape[1])}

  print("\nNumber of samples per class in training set: ")
  for label, count in train_counts.items():
      print(f"{herbi_omni[label]} ({label}): {count}")

  print("\nNumber of samples per class in testing set: ")
  for label, count in test_counts.items():
      print(f"{herbi_omni[label]} ({label}): {count}")


  print(f"\nTotal number of training samples: {sum(train_counts.values())}")
  print(f"Total number of test samples: {sum(test_counts.values())}")

count_samples()
```

```
Number of samples per class in training set:
camel (0): 500
cattle (1): 500
chimpanzee (2): 500
elephant (3): 500
kangaroo (4): 500

Number of samples per class in testing set:
camel (0): 100
cattle (1): 100
chimpanzee (2): 100
elephant (3): 100
kangaroo (4): 100

Total number of training samples: 2500
Total number of test samples: 500
```

In order to verify if the samples were extracted correctly, the samples were counted and they matched the 500 samples per class in training and 100 samples per class in test specification of the CIFAR-100 dataset.

In [ ]:
```python
# Normalize the pixels of the images from 0 to 1

X_train = X_train / 255
X_test = X_test / 255
```

In [ ]:
```python
# Display 25 sample images with labels
def view_samples():
  plt.figure(figsize=(10,10))
  for i in range(25):
      plt.subplot(5,5,i+1)
      plt.xticks([])
      plt.yticks([])
      plt.grid(False)
      plt.imshow(X_train[i], cmap=plt.cm.binary)
      plt.xlabel(herbi_omni[np.argmax(y_train[i])])
  plt.show()

view_samples()
```



Data Augmentation

The dataset is too small for accurate image processing. For a dataset to be adequate in size for image classification, around 1000 images per class is needed (Picsellia, 2022).

According to Sajid (2022), image data augmentation involves creating new transformed versions of images from an existing dataset to enhance its diversity. Images, essentially 2-dimensional arrays of pixel values, can be manipulated in various ways to generate augmented versions. These augmented images closely resemble those in the original dataset but provide additional information, aiding the machine learning algorithm's generalization capabilities.

In [ ]:
```
data_augmentation_pipeline = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
  tf.keras.layers.RandomZoom(0.3),
  tf.keras.layers.RandomContrast(0.3)
])

# Define the number of times to augment each image
num_augmentations = 2

# Initialize lists to store augmented images and labels
augmented_training_set = []
augmented_labels = []

# Apply data augmentation to each image multiple times
for image, label in zip(X_train, y_train):
    for _ in range(num_augmentations):
        augmented_image = data_augmentation_pipeline(image)  # Apply data augmentation
        augmented_training_set.append(augmented_image)
        augmented_labels.append(label)

augmented_training_set = np.array(augmented_training_set)
```

```
augmented_labels = np.array(augmented_labels)
```

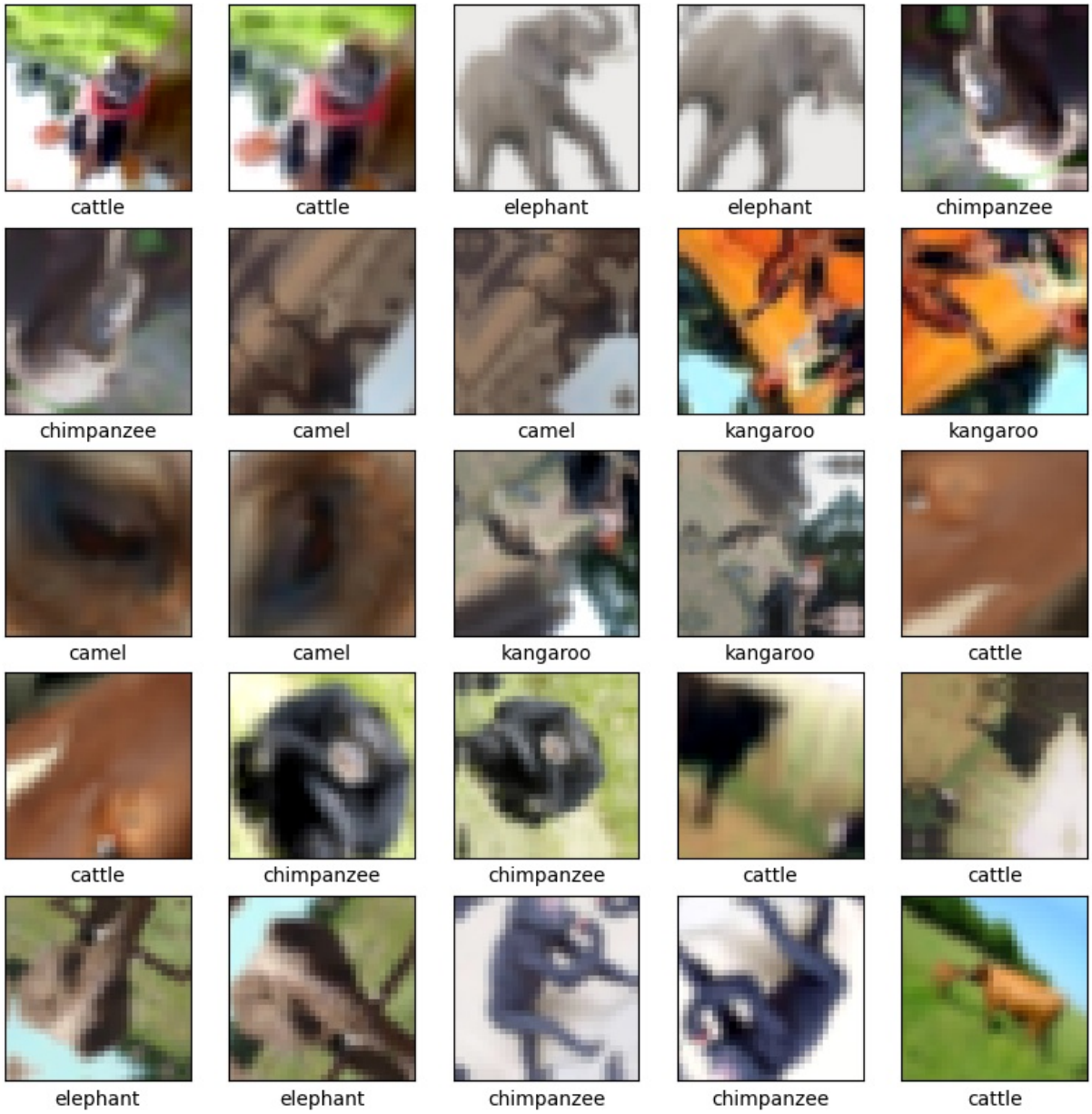
In [ ]:
```
augmented_training_set.shape
```

Out[ ]: (5000, 32, 32, 3)

In order to facilitate data augmentation, we created a pipeline that applied `RandomFlip`, `RandomRotation`, `RandomZoom`, and `RandomContrast` to each image in the training set. For each image, 2 augmentations were also created. This led to the result of 5000 images just from augmentation alone.

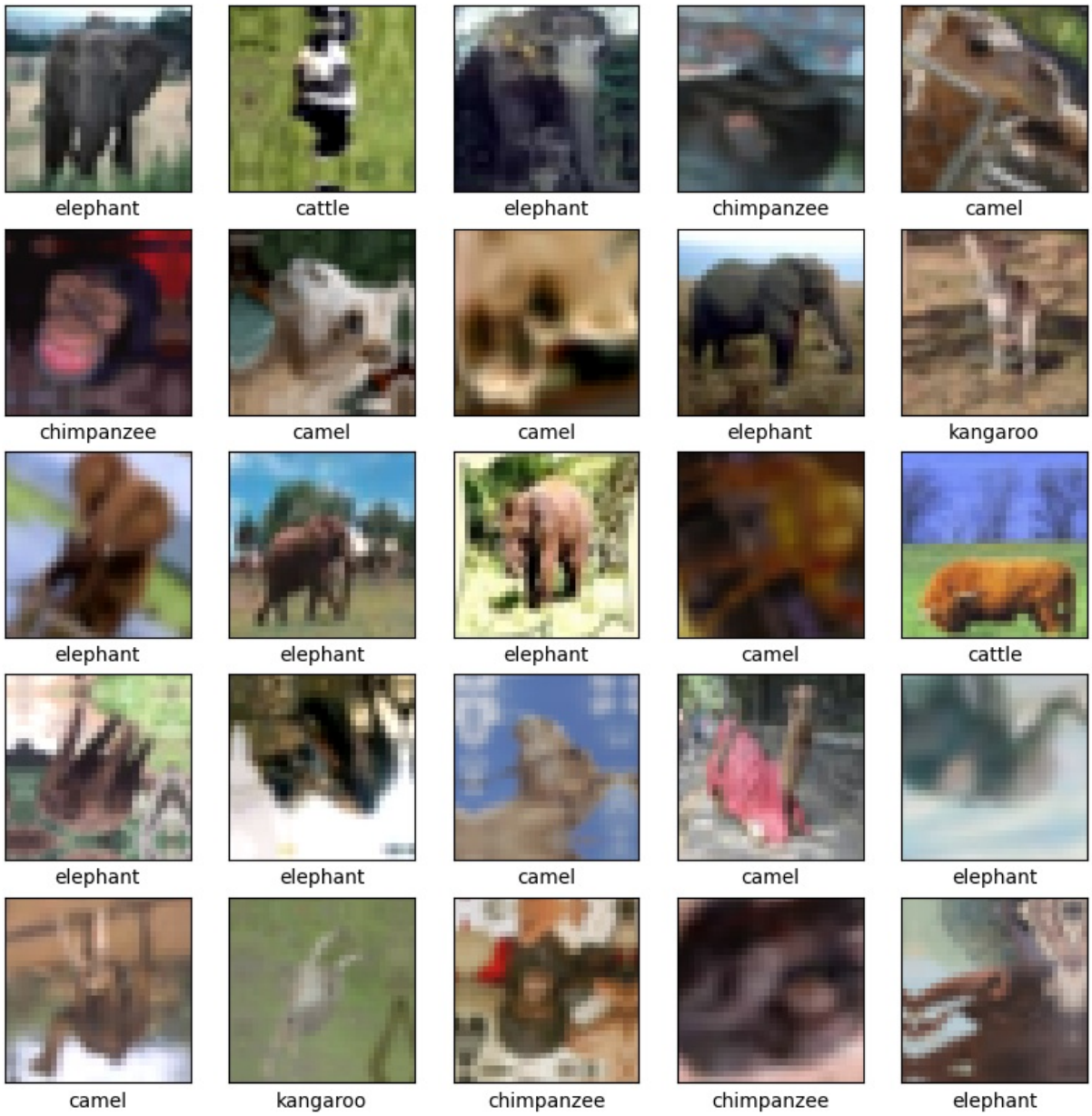
In [ ]:
```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(augmented_training_set[i], cmap=plt.cm.binary)
    plt.xlabel(herbi_omni[np.argmax(augmented_labels[i])])
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
```



In [ ]:
```
# Merge original and augmented dataset, and shuffle to ensure randomness
X_train = np.concatenate((X_train, augmented_training_set), axis=0)
y_train = np.concatenate((y_train, augmented_labels), axis=0)

# Shuffle the merged dataset to maintain randomness
shuffle_indices = np.random.permutation(len(X_train))
X_train = X_train[shuffle_indices]
y_train = y_train[shuffle_indices]

view_samples()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
0..255] for integers).
```



In this step, the augmented training set was merged to the original training set, and were also shuffled. Shuffling data is a crucial preprocessing method used to enhance model learning. It helps overcome problems associated with sequential patterns in training samples, reducing the risk of overfitting. Additionally, it addresses imbalances or differences in the distribution of objects or images between training and validation sets (Dupont, 2023).

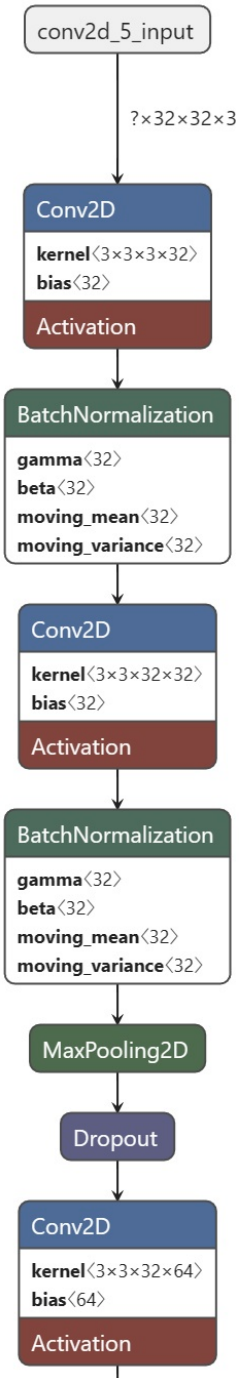
In [ ]:
```
# Count samples again to see how much the dataset increased by
count_samples()
```

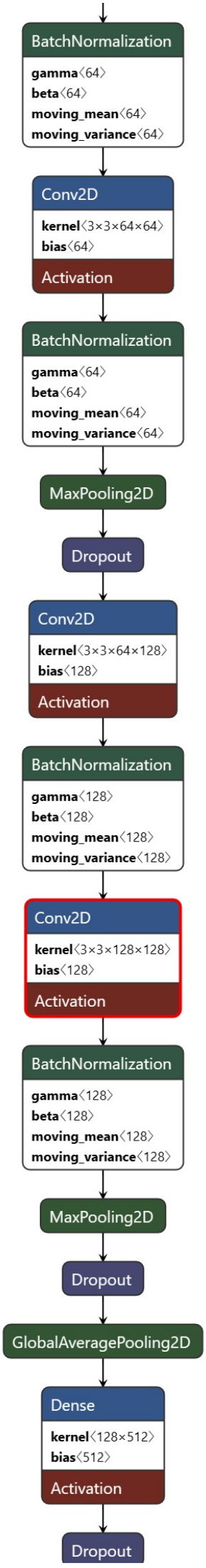
```
Number of samples per class in training set:
camel (0): 1500
cattle (1): 1500
chimpanzee (2): 1500
elephant (3): 1500
kangaroo (4): 1500

Number of samples per class in testing set:
camel (0): 100
cattle (1): 100
chimpanzee (2): 100
elephant (3): 100
kangaroo (4): 100

Total number of training samples: 7500
Total number of test samples: 500
```

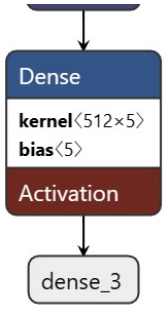
We have now satisfied the minimum 1000 as cited earlier. The dataset has now been expanded to 1500 images per class on training with a total of 7500 images for training alone. However, the test set was untouched. The test set should not be augmented because its purpose is to evaluate the model's performance on unseen data that reflects real-world scenarios. Augmenting the test set would introduce artificial variations that the model has not been trained to handle, thereby invalidating the assessment of its generalization ability (Barreto, 2022).

Model Training

Model Architecture




BatchNormalization
gamma〈64〉
beta〈64〉
moving_mean〈64〉
moving_variance〈64〉
Conv2D
kernel〈3×3×64×64〉
bias〈64〉
Activation
BatchNormalization
gamma〈64〉
beta〈64〉
moving_mean〈64〉
moving_variance〈64〉
MaxPooling2D
Dropout
Conv2D
kernel〈3×3×64×128〉
bias〈128〉
Activation
BatchNormalization
gamma〈128〉
beta〈128〉
moving_mean〈128〉
moving_variance〈128〉
Conv2D
kernel〈3×3×128×128〉
bias〈128〉
Activation
BatchNormalization
gamma〈128〉
beta〈128〉
moving_mean〈128〉
moving_variance〈128〉
MaxPooling2D
Dropout
GlobalAveragePooling2D
Dense
kernel〈128×512〉
bias〈512〉
Activation
Dropout



The neural network to be created is a Convolutional Neural Network (CNN). The model architecture comprises multiple convolutional layers with rectified linear unit (ReLU) activation functions, which extract hierarchical features from input images. These convolutional layers are followed by batch normalization layers that normalize the inputs to each layer, improving the stability and convergence of the model during training. Max-pooling layers downsample the feature maps, reducing spatial dimensions and extracting dominant features. Dropout layers are incorporated to prevent overfitting by randomly deactivating neurons during training. The model employs global average pooling to reduce spatial dimensions and compute feature representations. Fully connected layers with ReLU activation functions further process the extracted features before the final softmax layer, which produces class probabilities for multi-class classification. The Adam optimizer with gradient clipping helps stabilize the training process by limiting the magnitude of gradients.

In [ ]:
```
# Define the file path to save the model
filepath = 'highest_performing_model.h5'

# Define the ModelCheckpoint callback
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath, monitor='val_accuracy', save_best_only=True, mode='ma

# Define the model architecture
model = tf.keras.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3), padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.35),

    layers.GlobalAveragePooling2D(),  # Applying Global Average Pooling instead of Flatten
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(5, activation='softmax')
])

# Applying Gradient Clipping
optimizer = tf.keras.optimizers.Adam(clipvalue=0.5)

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

model.summary()

base_history = model.fit(X_train, y_train, epochs=300, validation_split=0.1, validation_data=(X_test[:1500], y_
model.load_weights(filepath)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Bat  (None, 32, 32, 32)        128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 global_average_pooling2d (  (None, 128)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 512)               66048

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 5)                 2565

=================================================================
Total params: 357413 (1.36 MB)
Trainable params: 356517 (1.36 MB)
Non-trainable params: 896 (3.50 KB)
_________________________________________________________________
Epoch 1/300
235/235 [==============================] - ETA: 0s - loss: 1.4595 - accuracy: 0.3864
Epoch 1: val_accuracy improved from -inf to 0.20000, saving model to highest_performing_model.h5
235/235 [==============================] - 13s 15ms/step - loss: 1.4595 - accuracy: 0.3864 - val_loss: 2.6172 -
val_accuracy: 0.2000
Epoch 2/300
  8/235 [>.............................] - ETA: 1s - loss: 1.3252 - accuracy: 0.4570
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mode
l as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
234/235 [============================>.] - ETA: 0s - loss: 1.3190 - accuracy: 0.4606
Epoch 2: val_accuracy improved from 0.20000 to 0.37000, saving model to highest_performing_model.h5
235/235 [==============================] - 2s 9ms/step - loss: 1.3186 - accuracy: 0.4609 - val_loss: 1.5012 - va
l_accuracy: 0.3700
Epoch 3/300
234/235 [============================>.] - ETA: 0s - loss: 1.2678 - accuracy: 0.4880
Epoch 3: val_accuracy improved from 0.37000 to 0.50200, saving model to highest_performing_model.h5
235/235 [==============================] - 2s 9ms/step - loss: 1.2687 - accuracy: 0.4877 - val_loss: 1.2389 - va
l_accuracy: 0.5020
```

```
234/235 [============================>.] - ETA: 0s - loss: 0.0710 - accuracy: 0.9753
Epoch 286: val_accuracy did not improve from 0.75400
235/235 [==============================] - 3s 13ms/step - loss: 0.0711 - accuracy: 0.9753 - val_loss: 1.7838 - v
al_accuracy: 0.7320
Epoch 287/300
232/235 [============================>.] - ETA: 0s - loss: 0.0673 - accuracy: 0.9791
Epoch 287: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0677 - accuracy: 0.9791 - val_loss: 1.6735 - v
al_accuracy: 0.7200
Epoch 288/300
235/235 [==============================] - ETA: 0s - loss: 0.0727 - accuracy: 0.9760
Epoch 288: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0727 - accuracy: 0.9760 - val_loss: 1.6293 - v
al_accuracy: 0.7220
Epoch 289/300
233/235 [============================>.] - ETA: 0s - loss: 0.0636 - accuracy: 0.9776
Epoch 289: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 9ms/step - loss: 0.0634 - accuracy: 0.9777 - val_loss: 1.9413 - va
l_accuracy: 0.7200
Epoch 290/300
230/235 [============================>.] - ETA: 0s - loss: 0.0710 - accuracy: 0.9758
Epoch 290: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 9ms/step - loss: 0.0702 - accuracy: 0.9761 - val_loss: 1.6506 - va
l_accuracy: 0.7260
Epoch 291/300
234/235 [============================>.] - ETA: 0s - loss: 0.0587 - accuracy: 0.9793
Epoch 291: val_accuracy did not improve from 0.75400
235/235 [==============================] - 3s 12ms/step - loss: 0.0588 - accuracy: 0.9791 - val_loss: 1.4891 - v
al_accuracy: 0.7300
Epoch 292/300
235/235 [==============================] - ETA: 0s - loss: 0.0615 - accuracy: 0.9793
Epoch 292: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0615 - accuracy: 0.9793 - val_loss: 1.5659 - v
al_accuracy: 0.7100
Epoch 293/300
234/235 [============================>.] - ETA: 0s - loss: 0.0669 - accuracy: 0.9768
Epoch 293: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0669 - accuracy: 0.9768 - val_loss: 1.6676 - v
al_accuracy: 0.7060
Epoch 294/300
233/235 [============================>.] - ETA: 0s - loss: 0.0649 - accuracy: 0.9776
Epoch 294: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 9ms/step - loss: 0.0650 - accuracy: 0.9775 - val_loss: 1.7756 - va
l_accuracy: 0.7040
Epoch 295/300
232/235 [============================>.] - ETA: 0s - loss: 0.0718 - accuracy: 0.9760
Epoch 295: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0719 - accuracy: 0.9760 - val_loss: 2.4102 - v
al_accuracy: 0.6660
Epoch 296/300
231/235 [============================>.] - ETA: 0s - loss: 0.0622 - accuracy: 0.9771
Epoch 296: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 11ms/step - loss: 0.0622 - accuracy: 0.9772 - val_loss: 1.9838 - v
al_accuracy: 0.7120
Epoch 297/300
233/235 [============================>.] - ETA: 0s - loss: 0.0651 - accuracy: 0.9784
Epoch 297: val_accuracy did not improve from 0.75400
235/235 [==============================] - 3s 11ms/step - loss: 0.0649 - accuracy: 0.9784 - val_loss: 1.6503 - v
al_accuracy: 0.7140
Epoch 298/300
235/235 [==============================] - ETA: 0s - loss: 0.0672 - accuracy: 0.9784
Epoch 298: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0672 - accuracy: 0.9784 - val_loss: 1.7662 - v
al_accuracy: 0.7040
Epoch 299/300
235/235 [==============================] - ETA: 0s - loss: 0.0695 - accuracy: 0.9759
Epoch 299: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0695 - accuracy: 0.9759 - val_loss: 2.1344 - v
al_accuracy: 0.6940
Epoch 300/300
235/235 [==============================] - ETA: 0s - loss: 0.0677 - accuracy: 0.9775
Epoch 300: val_accuracy did not improve from 0.75400
235/235 [==============================] - 2s 10ms/step - loss: 0.0677 - accuracy: 0.9775 - val_loss: 2.0664 - v
al_accuracy: 0.6880
```

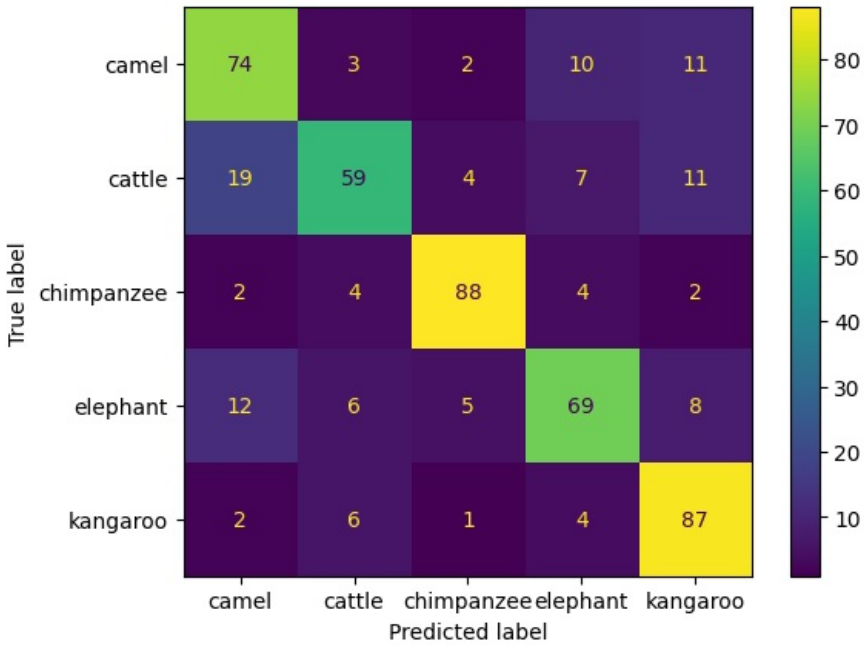
Metrics of Base Model

In [ ]:

```
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
model = tf.keras.models.load_model(filepath)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
y_pred_classes = [np.argmax(element) for element in y_pred]
y_test_classes = [np.argmax(element) for element in y_test]

print("Classification Report:\n", classification_report(y_test_classes, y_pred_classes))

conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)

cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=herbi_omni)

cm_display.plot()
plt.show()
```

```
16/16 [==============================] - 0s 3ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.74      0.71       100
           1       0.76      0.59      0.66       100
           2       0.88      0.88      0.88       100
           3       0.73      0.69      0.71       100
           4       0.73      0.87      0.79       100

    accuracy                           0.75       500
   macro avg       0.76      0.75      0.75       500
weighted avg       0.76      0.75      0.75       500
```



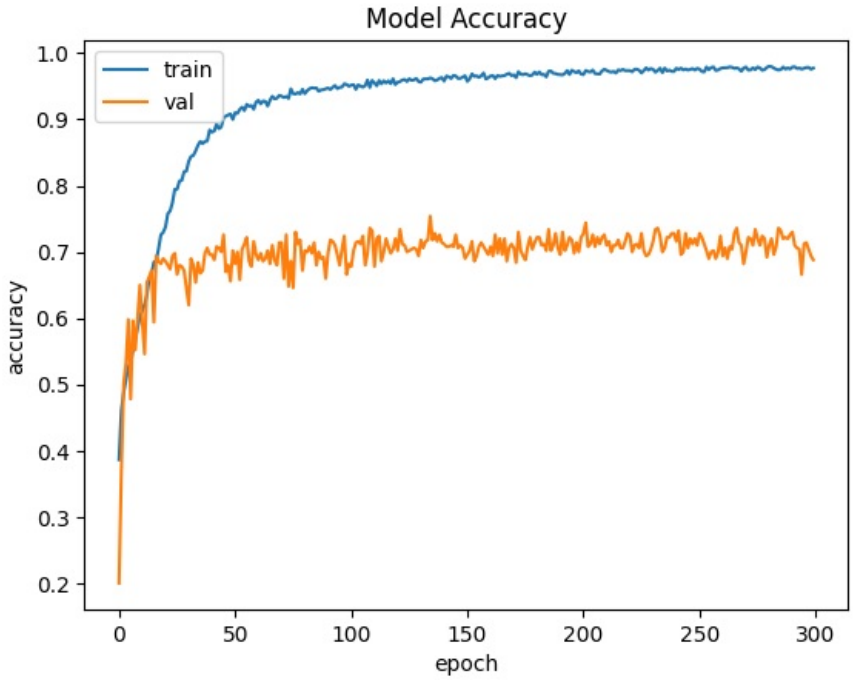
After 300 epochs of training, the model managed to hit a peak of 75.4% validation accuracy. It achieves a balance between precision and recall across different classes, with some classes exhibiting higher precision while others showcase superior recall rates. The `chimpanzee` class stands out with high precision, recall, and F1-score, indicating robust performance in classification. Despite slight variations in performance across classes, the model maintains a weighted average precision, recall, and F1-score of 0.76, reflecting its consistency across the dataset.

In [ ]:
```
# Evaluate loss on the training set
train_loss = model.evaluate(X_train, y_train, verbose=0)

print("Training Loss:", train_loss[0])
print("Training Accuracy:", train_loss[1])

# Evaluate loss on the testing set
test_loss = model.evaluate(X_test, y_test, verbose=0)

print("Testing Loss:", test_loss[0])
print("Testing Accuracy:", test_loss[1])
```

```
Training Loss: 0.005873758811503649
Training Accuracy: 0.9983999729156494
Testing Loss: 1.4739247560501099
Testing Accuracy: 0.7540000081062317
```

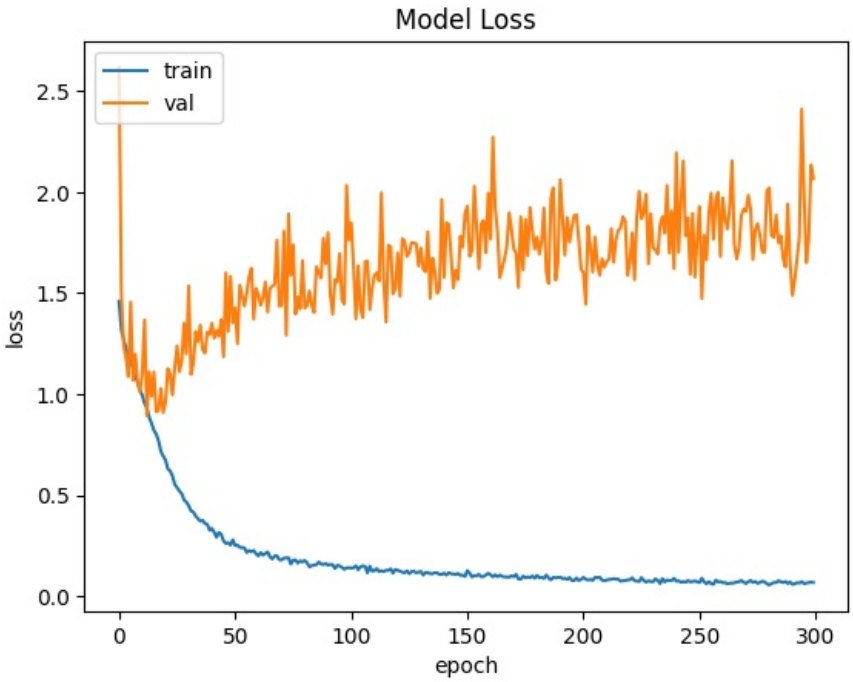
In [ ]:
```
plt.plot(base_history.history['accuracy'])
plt.plot(base_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

```
plt.show()
```



In terms of model accuracy, training accuracy converged or did not improve any further around the 150th epoch. For validation accuracy, it peak at around the 150th epoch also. Additional training did not improve both metrics any more.

In [ ]:
```
plt.plot(base_history.history['loss'])
plt.plot(base_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



As for model loss, training loss converged at around the 150th epoch also. However for validation loss, the values are volatile and fluctuates heavily. The lack of convergence for validation loss could indicate overfitting on the training set and that there is simply not enough data to learn from.

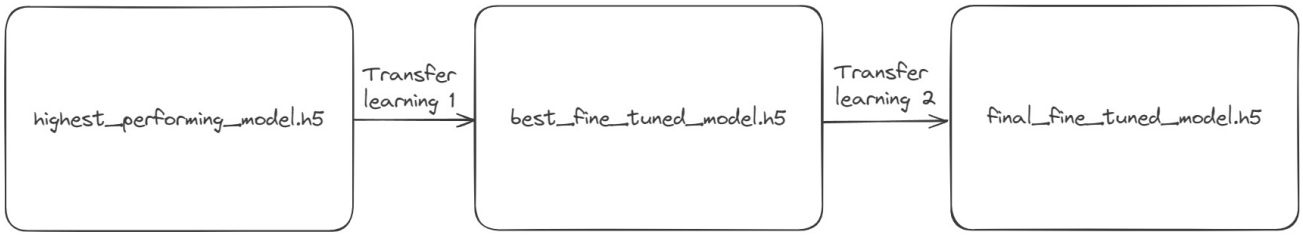
In pursuit of better metrics, a new technique was implemented which is called transfer learning.

Transfer Learning

Transfer learning is a pivotal method in Deep Learning, enabling the reuse of pre-existing models and their insights on new tasks. This approach revolutionizes training deep neural networks, particularly in scenarios with scarce data. Transfer learning is particularly relevant in data science, where access to labeled data can be limited. In our case, we are using our previous model

`highest_performing_model.h5` to transfer learn a new model, hoping that validation accuracy would improve.

The Process



Transfer learning was done 2 times in order to achieve the highest accuracy we possibly could. Both training phases used the same model as the original training however, with a lower learning rate for fine-tuning. A reduced learning rate is commonly favored as it promotes greater stability during the fine-tuning process. This helps maintain the model's previously acquired features with minimal disruption or significant modifications (Buhl, 2023).

In [ ]:
```python
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint

# Load the pre-trained model
model = tf.keras.models.load_model('highest_performing_model.h5')

# Unfreeze some layers
for layer in model.layers:
    layer.trainable = True

# lower learning rate for fine-tuning
optimizer = tf.keras.optimizers.Adam(lr=1e-5)  # Example: lr=1e-5

# Compile the model with the new optimizer and a lower learning rate
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
filepath = 'best_model.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)
history = model.fit(X_train, y_train, epochs=300, validation_data=(X_test[:1500], y_test[:1500]), callbacks=[ch
best_model = tf.keras.models.load_model(filepath)

# Evaluate the best model on the test set
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
best_model.save('best_fine_tuned_model.h5')
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.
,tf.keras.optimizers.legacy.Adam.
Epoch 1/300
232/235 [============================>.] - ETA: 0s - loss: 0.9040 - accuracy: 0.7435
Epoch 1: val_accuracy improved from -inf to 0.72800, saving model to best_model.h5
235/235 [==============================] - 7s 11ms/step - loss: 0.9014 - accuracy: 0.7440 - val_loss: 0.8516 - v
al_accuracy: 0.7280
Epoch 2/300
233/235 [============================>.] - ETA: 0s - loss: 0.6609 - accuracy: 0.7657
Epoch 2: val_accuracy improved from 0.72800 to 0.77000, saving model to best_model.h5
235/235 [==============================] - 2s 10ms/step - loss: 0.6610 - accuracy: 0.7661 - val_loss: 0.8019 - v
al_accuracy: 0.7700
Epoch 3/300
230/235 [============================>.] - ETA: 0s - loss: 0.6067 - accuracy: 0.7807
Epoch 3: val_accuracy did not improve from 0.77000
235/235 [==============================] - 3s 11ms/step - loss: 0.6046 - accuracy: 0.7816 - val_loss: 0.8540 - v
al_accuracy: 0.7620
Epoch 4/300
230/235 [============================>.] - ETA: 0s - loss: 0.5783 - accuracy: 0.7898
Epoch 4: val_accuracy did not improve from 0.77000
235/235 [==============================] - 2s 9ms/step - loss: 0.5765 - accuracy: 0.7899 - val_loss: 0.8477 - va
l_accuracy: 0.7360
Epoch 5/300
233/235 [============================>.] - ETA: 0s - loss: 0.5326 - accuracy: 0.8054
Epoch 5: val_accuracy did not improve from 0.77000
235/235 [==============================] - 2s 9ms/step - loss: 0.5323 - accuracy: 0.8056 - val_loss: 0.9081 - va
l_accuracy: 0.7700
Epoch 6/300
232/235 [============================>.] - ETA: 0s - loss: 0.5107 - accuracy: 0.8090
Epoch 6: val_accuracy did not improve from 0.77000
235/235 [==============================] - 2s 9ms/step - loss: 0.5094 - accuracy: 0.8095 - val_loss: 0.9436 - va
l_accuracy: 0.7360
Epoch 7/300
```

```
Epoch 289: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0726 - accuracy: 0.9769 - val_loss: 1.7932 - va
l_accuracy: 0.7520
Epoch 290/300
233/235 [============================>.] - ETA: 0s - loss: 0.0571 - accuracy: 0.9819
Epoch 290: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0568 - accuracy: 0.9820 - val_loss: 1.6199 - va
l_accuracy: 0.7620
Epoch 291/300
232/235 [============================>.] - ETA: 0s - loss: 0.0492 - accuracy: 0.9822
Epoch 291: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0490 - accuracy: 0.9823 - val_loss: 1.6108 - va
l_accuracy: 0.7740
Epoch 292/300
233/235 [============================>.] - ETA: 0s - loss: 0.0531 - accuracy: 0.9816
Epoch 292: val_accuracy did not improve from 0.78800
235/235 [==============================] - 3s 12ms/step - loss: 0.0530 - accuracy: 0.9817 - val_loss: 1.7145 - v
al_accuracy: 0.7380
Epoch 293/300
233/235 [============================>.] - ETA: 0s - loss: 0.0484 - accuracy: 0.9846
Epoch 293: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0490 - accuracy: 0.9845 - val_loss: 1.5297 - va
l_accuracy: 0.7580
Epoch 294/300
235/235 [==============================] - ETA: 0s - loss: 0.0527 - accuracy: 0.9821
Epoch 294: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0527 - accuracy: 0.9821 - val_loss: 1.5804 - va
l_accuracy: 0.7700
Epoch 295/300
235/235 [==============================] - ETA: 0s - loss: 0.0568 - accuracy: 0.9803
Epoch 295: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0568 - accuracy: 0.9803 - val_loss: 1.5357 - va
l_accuracy: 0.7580
Epoch 296/300
232/235 [============================>.] - ETA: 0s - loss: 0.0641 - accuracy: 0.9789
Epoch 296: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0641 - accuracy: 0.9788 - val_loss: 1.5397 - va
l_accuracy: 0.7620
Epoch 297/300
230/235 [============================>.] - ETA: 0s - loss: 0.0577 - accuracy: 0.9817
Epoch 297: val_accuracy did not improve from 0.78800
235/235 [==============================] - 3s 11ms/step - loss: 0.0575 - accuracy: 0.9815 - val_loss: 1.5185 - v
al_accuracy: 0.7440
Epoch 298/300
231/235 [============================>.] - ETA: 0s - loss: 0.0510 - accuracy: 0.9844
Epoch 298: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 10ms/step - loss: 0.0508 - accuracy: 0.9845 - val_loss: 1.7837 - v
al_accuracy: 0.7340
Epoch 299/300
233/235 [============================>.] - ETA: 0s - loss: 0.0533 - accuracy: 0.9818
Epoch 299: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0537 - accuracy: 0.9816 - val_loss: 1.5307 - va
l_accuracy: 0.7600
Epoch 300/300
230/235 [============================>.] - ETA: 0s - loss: 0.0518 - accuracy: 0.9808
Epoch 300: val_accuracy did not improve from 0.78800
235/235 [==============================] - 2s 9ms/step - loss: 0.0514 - accuracy: 0.9811 - val_loss: 1.5091 - va
l_accuracy: 0.7420
16/16 [==============================] - 0s 4ms/step - loss: 1.3621 - accuracy: 0.7880
Test Loss: 1.3621013164520264
Test Accuracy: 0.7879999876022339
```

In [ ]:

```python
# Final Transfer Learning

# Load the previous pre-trained model
model = tf.keras.models.load_model('best_fine_tuned_model.h5')

# Unfreeze some layers
for layer in model.layers:
    layer.trainable = True

# lower learning rate for fine-tuning
optimizer = tf.keras.optimizers.Adam(lr=1e-5)  # Example: lr=1e-5

# Compile the model with the new optimizer and a lower learning rate
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
filepath = 'best_model.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)
transfer_history = model.fit(X_train, y_train, epochs=300, validation_data=(X_test[:1500], y_test[:1500]), call
best_model = tf.keras.models.load_model(filepath)
```

```python
# Evaluate the best model on the test set
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
best_model.save('final_fine_tuned_model.h5')
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.
,tf.keras.optimizers.legacy.Adam.
Epoch 1/300
234/235 [============================>.] - ETA: 0s - loss: 0.8400 - accuracy: 0.7724
Epoch 1: val_accuracy improved from -inf to 0.75200, saving model to best_model.h5
235/235 [==============================] - 7s 13ms/step - loss: 0.8396 - accuracy: 0.7725 - val_loss: 0.8143 - v
al_accuracy: 0.7520
Epoch 2/300
  8/235 [>.............................] - ETA: 1s - loss: 0.6071 - accuracy: 0.7656
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mode
l as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
229/235 [============================>.] - ETA: 0s - loss: 0.6135 - accuracy: 0.7858
Epoch 2: val_accuracy improved from 0.75200 to 0.75800, saving model to best_model.h5
235/235 [==============================] - 2s 8ms/step - loss: 0.6169 - accuracy: 0.7849 - val_loss: 0.7809 - va
l_accuracy: 0.7580
Epoch 3/300
231/235 [============================>.] - ETA: 0s - loss: 0.5663 - accuracy: 0.8019
Epoch 3: val_accuracy improved from 0.75800 to 0.76600, saving model to best_model.h5
235/235 [==============================] - 2s 9ms/step - loss: 0.5698 - accuracy: 0.8011 - val_loss: 0.8037 - va
l_accuracy: 0.7660
Epoch 4/300
235/235 [==============================] - ETA: 0s - loss: 0.5266 - accuracy: 0.8092
Epoch 4: val_accuracy improved from 0.76600 to 0.78200, saving model to best_model.h5
235/235 [==============================] - 2s 9ms/step - loss: 0.5266 - accuracy: 0.8092 - val_loss: 0.7971 - va
l_accuracy: 0.7820
Epoch 5/300
233/235 [============================>.] - ETA: 0s - loss: 0.5102 - accuracy: 0.8128
Epoch 5: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 8ms/step - loss: 0.5105 - accuracy: 0.8127 - val_loss: 0.8282 - va
l_accuracy: 0.7640
Epoch 6/300
235/235 [==============================] - ETA: 0s - loss: 0.4788 - accuracy: 0.8248
Epoch 6: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 10ms/step - loss: 0.4788 - accuracy: 0.8248 - val_loss: 0.8028 - v
al_accuracy: 0.7660
Epoch 7/300
229/235 [============================>.] - ETA: 0s - loss: 0.4511 - accuracy: 0.8354
Epoch 7: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 10ms/step - loss: 0.4485 - accuracy: 0.8367 - val_loss: 0.8480 - v
al_accuracy: 0.7640
Epoch 8/300
231/235 [============================>.] - ETA: 0s - loss: 0.4259 - accuracy: 0.8462
Epoch 8: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 8ms/step - loss: 0.4254 - accuracy: 0.8469 - val_loss: 1.0047 - va
l_accuracy: 0.7460
Epoch 9/300
234/235 [============================>.] - ETA: 0s - loss: 0.4008 - accuracy: 0.8523
Epoch 9: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 9ms/step - loss: 0.4015 - accuracy: 0.8521 - val_loss: 0.9670 - va
l_accuracy: 0.7520
Epoch 10/300
231/235 [============================>.] - ETA: 0s - loss: 0.3922 - accuracy: 0.8539
Epoch 10: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 8ms/step - loss: 0.3924 - accuracy: 0.8537 - val_loss: 1.1465 - va
l_accuracy: 0.7380
Epoch 11/300
231/235 [============================>.] - ETA: 0s - loss: 0.3899 - accuracy: 0.8596
Epoch 11: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 9ms/step - loss: 0.3937 - accuracy: 0.8588 - val_loss: 0.9070 - va
l_accuracy: 0.7580
Epoch 12/300
234/235 [============================>.] - ETA: 0s - loss: 0.3618 - accuracy: 0.8670
Epoch 12: val_accuracy did not improve from 0.78200
235/235 [==============================] - 3s 11ms/step - loss: 0.3621 - accuracy: 0.8665 - val_loss: 0.9210 - v
al_accuracy: 0.7680
Epoch 13/300
232/235 [============================>.] - ETA: 0s - loss: 0.3686 - accuracy: 0.8697
Epoch 13: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 9ms/step - loss: 0.3682 - accuracy: 0.8697 - val_loss: 1.0419 - va
l_accuracy: 0.7600
Epoch 14/300
229/235 [============================>.] - ETA: 0s - loss: 0.3286 - accuracy: 0.8818
Epoch 14: val_accuracy did not improve from 0.78200
235/235 [==============================] - 2s 8ms/step - loss: 0.3294 - accuracy: 0.8813 - val_loss: 1.0178 - va
l_accuracy: 0.7620
```

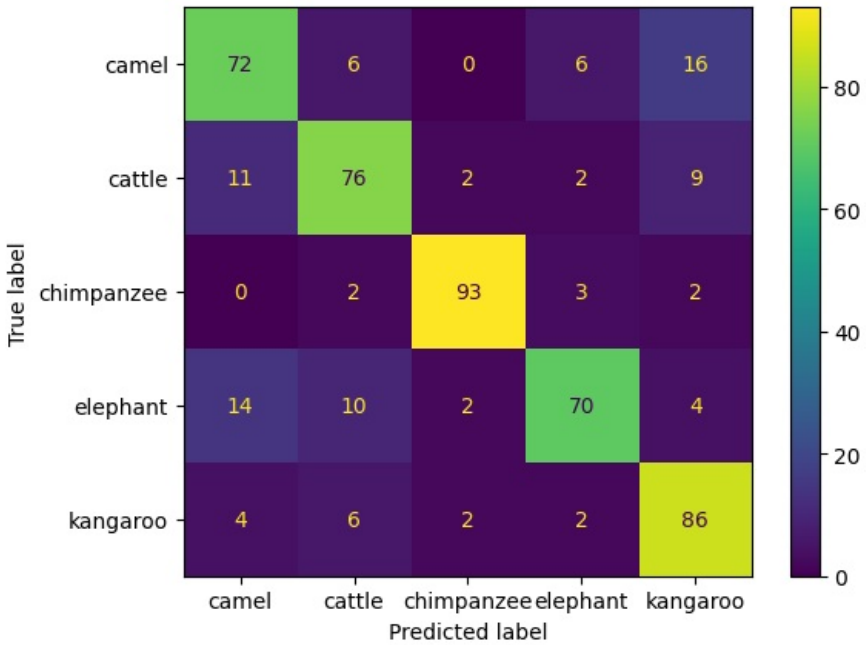
```
233/235 [============================>.] - ETA: 0s - loss: 0.0467 - accuracy: 0.9855
Epoch 297: val_accuracy did not improve from 0.79400
235/235 [==============================] - 2s 9ms/step - loss: 0.0465 - accuracy: 0.9856 - val_loss: 1.5183 - va
l_accuracy: 0.7760
Epoch 298/300
234/235 [============================>.] - ETA: 0s - loss: 0.0484 - accuracy: 0.9852
Epoch 298: val_accuracy did not improve from 0.79400
235/235 [==============================] - 2s 9ms/step - loss: 0.0484 - accuracy: 0.9852 - val_loss: 1.5018 - va
l_accuracy: 0.7620
Epoch 299/300
231/235 [============================>.] - ETA: 0s - loss: 0.0566 - accuracy: 0.9815
Epoch 299: val_accuracy did not improve from 0.79400
235/235 [==============================] - 2s 9ms/step - loss: 0.0563 - accuracy: 0.9816 - val_loss: 1.5062 - va
l_accuracy: 0.7540
Epoch 300/300
234/235 [============================>.] - ETA: 0s - loss: 0.0578 - accuracy: 0.9802
Epoch 300: val_accuracy did not improve from 0.79400
235/235 [==============================] - 3s 11ms/step - loss: 0.0578 - accuracy: 0.9801 - val_loss: 1.4687 - v
al_accuracy: 0.7880
16/16 [==============================] - 0s 5ms/step - loss: 1.5348 - accuracy: 0.7940
Test Loss: 1.5348057746887207
Test Accuracy: 0.7940000295639038
```

Metrics of Transfer Learned Model

In [ ]:

```
model = tf.keras.models.load_model('final_fine_tuned_model.h5')

# Make predictions on the test set
y_pred = model.predict(X_test)

y_pred_classes = [np.argmax(element) for element in y_pred]
y_test_classes = [np.argmax(element) for element in y_test]

print("Classification Report:\n", classification_report(y_test_classes, y_pred_classes))

conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)

cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=herbi_omni)

cm_display.plot()
plt.show()
```

```
16/16 [==============================] - 0s 3ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.71      0.72      0.72       100
           1       0.76      0.76      0.76       100
           2       0.94      0.93      0.93       100
           3       0.84      0.70      0.77       100
           4       0.74      0.86      0.79       100

    accuracy                           0.79       500
   macro avg       0.80      0.79      0.79       500
weighted avg       0.80      0.79      0.79       500
```

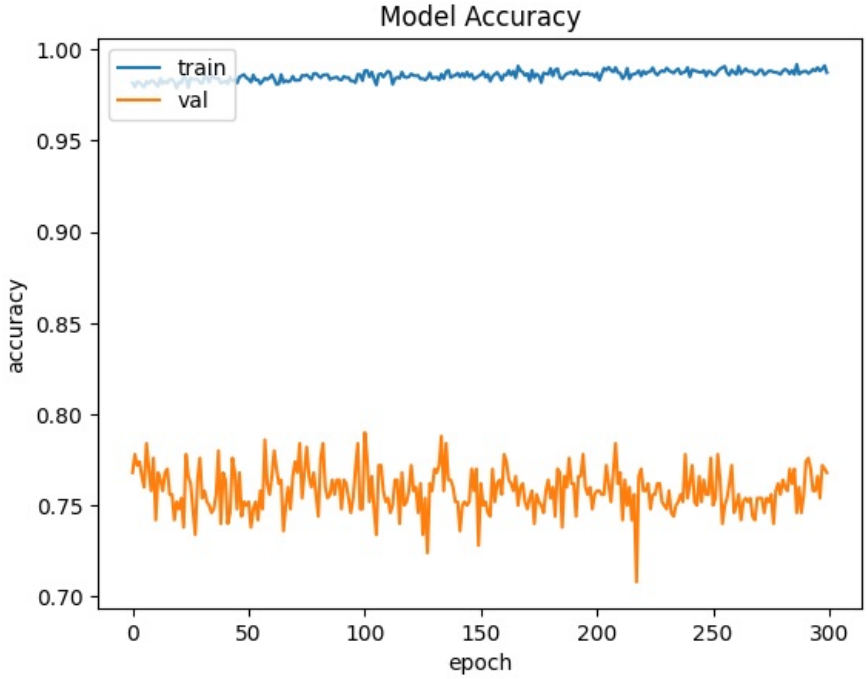


Significantly, classes `camel`, `cattle`, `elephant`, and `kangaroo` demonstrate improved precision, recall, and F1-scores, suggesting enhanced performance in classification. Class `chimpanzee` still retains its notably high precision, recall, and F1-score, implying superior accuracy and dependability in predictions. Additionally, the model's overall accuracy has risen to 0.79, indicating improved predictive abilities.

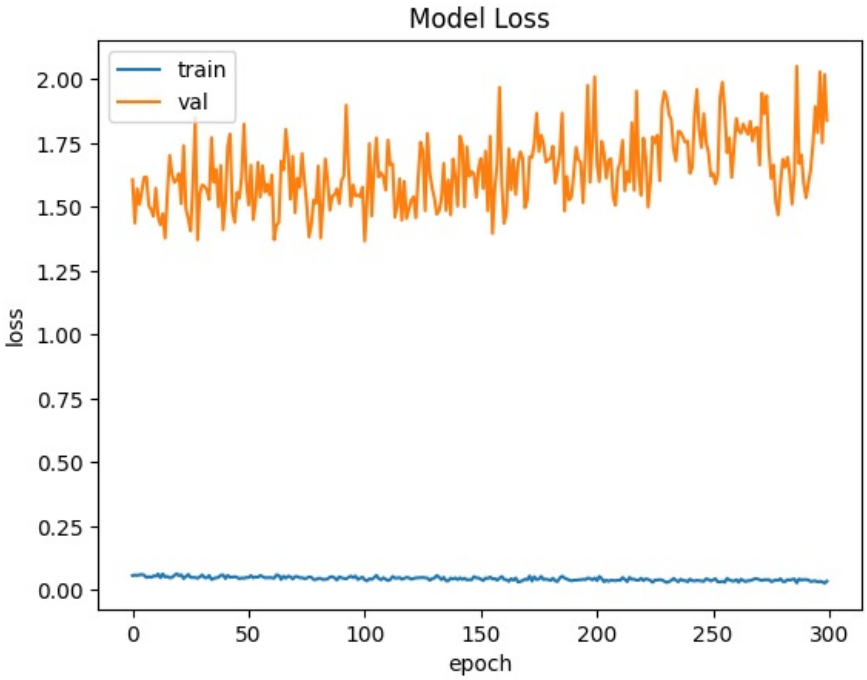
In [ ]:
```
# Evaluate loss on the training set
train_loss = model.evaluate(X_train, y_train, verbose=0)

print("Training Loss:", train_loss[0])
print("Training Accuracy:", train_loss[1])

# Evaluate loss on the testing set
test_loss = model.evaluate(X_test, y_test, verbose=0)

print("Testing Loss:", test_loss[0])
print("Testing Accuracy:", test_loss[1])
```

```
Training Loss: 0.00021732655295636505
Training Accuracy: 1.0
Testing Loss: 1.5348057746887207
Testing Accuracy: 0.7940000295639038
```

In [ ]:
```
plt.plot(transfer_history.history['accuracy'])
plt.plot(transfer_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [ ]:
```
plt.plot(transfer_history.history['loss'])
plt.plot(transfer_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In general, model validation loss and accuracy still struggles to achieve convergence which is an issue of overfit and lack of data.

Exporting the model

In [ ]:
```
import datetime
file_name = input("Enter a file name: ")
current_date = datetime.datetime.now().strftime("%Y-%m-%d")
model_path = f'{file_name}_{current_date}.h5'
model.save(model_path)
model.save_weights(f'{file_name}_{current_date}_weights.h5')
```

```
Enter a file name: 79VAC_FINALMODEL_
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mode
l as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Challenges Encountered

The final result took around 2 weeks of trial and error with testing multiple parameters and architectures across multiple machines. Some challenges the authors encountered were:

1. **Lack of Reproducibility**: Due to the random nature of augmentation, and shuffling, recreating previous results proved to be difficult for the authors. To mitigate this, we created a repository on GitHub which kept track of the best models so far.
2. **Model Architecture**: As the authors were new to deep learning, we had a difficulty formulating and planning the architecture of the neural network considering the time constraints of the project. As a way to lessen our load on architecture without prior knowledge, we tried to model the CNN after state-of-the-art neural network models used for Image Classification.
3. **Image Quality**: The images from CIFAR-100 are only 32x32. According to Dodge and Karam (2016), neural networks are highly susceptible to quality distortions such as blur and noise. In order to make the model adapt to these types of images, we implemented random augmentations which did help training the model to 79% validation accuracy.
4. **Training Time**: As we were training on 300 epochs, training on cloud services such as Colab took too long with an average epoch running time of around a minute each on our current architecture. This pushed us to use Windows Subsystem for Linux or `WSL` to take advantage of our GPUs on our local machines. Using Ubuntu and the Conda environment allowed us to leverage training with CUDA which lessened epoch time from 1 minute to around 1 - 3 seconds each on average.

Recommendations

The model performed satisfactorily on testing however, overfitting still remains prevalent after transfer learning and data augmentation. These are some points that were taken note by the authors in order to improve the model:

1. **More Data Augmentation**: Expanding the dataset with more samples and varying techniques could reduce overfit and improve the generalization ability of the model.

2. **Regularization Techniques**: Implement regularization techniques like L2 regularization to prevent overfitting during training. These methods can help the model generalize better to unseen data by reducing reliance on specific features.

3. **Hyperparameter Tuning**: Perform systematic hyperparameter tuning to optimize the model's architecture and training parameters. This includes adjusting learning rates, batch sizes, and optimizer algorithms to find the combination that yields the best performance on the testing set.

4. **Model Architecture Exploration**: Experiment with different neural network architectures, such as increasing the depth or width of the network, adding more layers, or trying different activation functions. A more complex model may capture intricate patterns in the data more effectively.

5. **Transfer Learning Refinement**: Fine-tune the transfer learning process by adjusting the layers frozen during pre-training and the learning rate schedule. This allows the model to adapt more specifically to the target task while still benefiting from pre-existing knowledge.

6. **Ensemble Methods**: If allowed to use pre-trained models like `VGG-16` or `YOLOv8`, these models could be ensembled through `bagging` or `boosting` in order to derive a more accurate result.

Conclusions and Key Takeaways

After 300 epochs of initial training and transfer learning, the model yielded promising results. With a training loss of `0.0002` and a training accuracy of `1.0`, it demonstrated a high level of proficiency in learning the training data. The testing phase revealed a slightly lower but still respectable testing accuracy of `0.79`, accompanied by a testing loss of `1.53`. These metrics suggest that the model was able to generalize reasonably well to unseen data, indicating the effectiveness of transfer learning in this context.

The exercise underscores the potential of AI in various sectors. In business and industry, AI-driven models can enhance decision-making processes, optimize operations, and personalize user experiences. By leveraging pre-existing knowledge and adapting it to new tasks, AI can improve efficiency and accuracy in tasks such as image recognition, sentiment analysis, and customer behavior prediction. Such advancements not only benefit businesses by increasing productivity and reducing costs but also enhance consumer experiences through tailored recommendations, improved search results, and better customer service interactions. Thus, even from this simple exercise, the far-reaching implications of AI in both business and consumer realms become apparent, signaling a transformative potential for numerous industries.

References

Barreto, S. (2022). Data augmentation. Baeldung. [Online]. Available: https://www.baeldung.com/cs/ml-data-augmentation

Buhl, N. (2023). Training vs. fine-tuning: What is the difference? Encord. [Online]. Available: https://encord.com/blog/training-vs-fine-tuning/#:~:text=A%20lower%20learning%20rate%20is

Crêteur, S. (2022, December 16). Machine learning: one-hot encoding vs integer encoding. Geek Culture. [Online]. Available: https://medium.com/geekculture/machine-learning-one-hot-encoding-vs-integer-encoding-f180eb831cf1#:~:text=One%20advantage%20of%20one%2Dhot

Dodge, S., & Karam, L. (2016). Understanding how image quality affects deep neural networks. Paper presented at the 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX). doi:10.1109/qomex.2016.7498955

Dupont, L. (2023). Data shuffling. Deci. [Online]. Available: https://deci.ai/course/data-shuffling/

Picsellia. (2022). How to ensure image dataset quality for image classification? Picsellia. [Online]. Available: https://www.picsellia.com/post/image-data-quality-for-image-classification

Sajid, H. (2022). What is image data augmentation? Picsellia. [Online]. Available: https://www.picsellia.com/post/image-data-augmentation