

Professor : Giovanni Finocchio

Arya Khosravirad

Matricola: 534 170



A picture containing text, font, logo, symbol

Description automatically generated

*Image*

*classification*

Dataset: Animal 151

Table of Contents

1.Introduction

2.main strategies

3.code explanation

4.training curves

5.confusion matrix

6.conclusion  
7.refrences

1.Introduction  
  
The objective of this project is to develop an image classification model using the "Animals 151" dataset obtained from Kaggle.

Image classification plays a vital role in computer vision, and this project aims to achieve a test accuracy of over 60% by implementing deep learning techniques.

The dataset comprises images of various animal species, and our model's success in accurately categorizing them will demonstrate its effectiveness in handling complex visual data.

This report presents our approach, including data preprocessing, model selection, training, and evaluation, to attain the desired classification accuracy.

2.main strategies

1. Data Preprocessing: The project initiated with thorough data preprocessing. We organized and structured the "Animals 151" dataset, ensuring uniform image sizes, pixel normalization, and class distribution. This step is pivotal in achieving robust model performance.
2. Model Selection: We adopted a transfer learning approach by leveraging the EfficientNetB0 architecture pre-trained on ImageNet. This decision significantly expedited our training process and increased the model's ability to recognize complex features in animal images.
3. Data Augmentation: Data augmentation was implemented to further enhance the model's generalization capabilities. Techniques such as random flipping were applied to augment the training dataset, allowing the model to learn from a more diverse set of examples.
4. Training and Fine-tuning: The model was trained with a suitable optimizer and loss function. We monitored its performance through training curves, analyzing both training and validation accuracy and loss. Fine-tuning was employed to adjust hyperparameters and improve model convergence.
5. Evaluation and Confusion Matrix: To assess the model's accuracy and class-wise performance, we conducted rigorous evaluation on a separate test dataset. We calculated a confusion matrix and associated metrics, providing insights into the model's strengths and weaknesses for each animal category.

3.code explanation

1. This part of the code is responsible for importing necessary libraries and performing some initial setup.

A screenshot of a computer

Description automatically generated

warnings.filterwarnings('ignore'): Sets a filter to ignore warning messages, ensuring that they won't be displayed during execution.

tf.get\_logger().setLevel('WARNING'): Sets the logging level of TensorFlow to "WARNING," which reduces log output to a minimum.

drive.mount('/content/drive'): Mounts your Google Drive to the '/content/drive' directory in Google Colab, enabling access to files stored on Google Drive within your Colab environment.

1. !mkdir animals/ is used to create a new directory named "animals.

to create a directory dedicated to storing the dataset for the image classification project

A screenshot of a computer program

Description automatically generated

1. BATCH\_SIZE = 64 and IMG\_SIZE = (224, 224) are used to define two important parameters

related to training a machine learning model for image classification.

A screenshot of a computer program

Description automatically generated

* BATCH\_SIZE refers to the number of data samples (in this case, images) that are processed by the machine learning model in each iteration during training.
* A batch is a subset of the entire training dataset that the model uses to update its weights. Instead of processing all training samples at once (which can be computationally expensive and memory-intensive), the dataset is divided into smaller batche
* In this code, the BATCH\_SIZE is set to 64, which means that the model will process 64 images at a time during each training iteration.
* IMG\_SIZE represents the dimensions to which the input images are resized before being fed into the machine learning model.
* In this case, (224, 224) indicates that all input images will be resized to a square shape with a width and height of 224 pixels.

1. org\_dir stores the path to the original directory containing the dataset on Google Drive.

A screenshot of a computer program

Description automatically generated

1. This code reads the class names from the org\_dir and creates corresponding directories in the 'animals' directory if they don't already exist.

A screenshot of a computer program

Description automatically generated

1. This line uses the glob module to generate a list of file paths for all image files with the ".jpg" extension in subdirectories within the "org\_dir" directory.

A screenshot of a computer program

Description automatically generated

"\*/\*.jpg": The pattern \*/\*.jpg indicates that the code should search for files in subdirectories. The first asterisk (*) represents any directory name, and the second asterisk (*) represents any file name with the ".jpg" extension.

1. This loop processes each image file, converts it to RGB format, and saves it to the 'animals' directory, organized by class.

A screenshot of a computer program

Description automatically generated

The outcome of this code is a meticulously organized dataset, where images are grouped by class. This structured dataset is ideally suited for training machine learning models, particularly those designed for image classification. The process ensures that each image is in the correct format and that images of the same class are grouped together, simplifying data loading and preprocessing during model training. As a result, the dataset becomes a valuable resource for developing and evaluating machine learning models, such as those aimed at classifying animals in this instance.

1. copied\_dir points to the directory containing the organized dataset.

A screenshot of a computer program

Description automatically generated

1. this code prepares the data for training and evaluation, making it suitable for developing and assessing an image classification model using machine learning techniques.

A screenshot of a computer

Description automatically generated

The code creates a training and validation dataset using the keras.utils.image\_dataset\_from\_directory function. It specifies the following parameters:

* copied\_dir: The 'dataset' directory is designated as the source of images.
* validation\_split=0.2: It reserves 20% of the data for validation and uses the remaining 80% for training.
* subset='training': Indicates that this dataset is intended for training.
* batch\_size = batch\_size: Sets the batch size for training data (e.g., 64 images per batch).
* image\_size= image\_dimensions : Defines the image size (e.g., 224x224 pixels).
* shuffle=True: Specifies that the data should be shuffled to ensure randomness.
* seed=common\_seed: Uses the common seed value for shuffling.

dataset consists of 6,270 image files distributed across 151 different classes, and it is split into a training set with 5,016 images and a validation set with 1,254 images. These sets are used for training and evaluating an image classification model. The 80-20 split is a common choice for dividing data into training and validation sets

1. Retrieves the class names from the training dataset and generates a 3x3 grid of images from the training dataset, displaying each image along with its corresponding class label. It specifies the number of rows and columns and then plots a grid of images with their respective class labels.

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

1. In this part we compute the number of batches in the validation dataset. And Splits the validation dataset into a test dataset (25%) and a reduced validation dataset (75%).

A screenshot of a computer

Description automatically generated

1. Prints the number of batches in the training, validation, and test datasets.

A screenshot of a computer

Description automatically generated

1. validation\_batches = tf.data.experimental.cardinality(validation\_dataset):
   * Calculates the total number of batches in the validation dataset using tf.data.experimental.cardinality(validation\_dataset).
2. test\_dataset = validation\_dataset.take(validation\_batches // 4):
   * Creates a testing dataset by taking 25% of the validation data in terms of batches.
3. validation\_dataset = validation\_dataset.skip(validation\_batches // 4):
   * Modifies the validation dataset by excluding the data allocated for testing, retaining 75% for validation.
4. Enables data prefetching for the datasets to improve data loading performance.

A screenshot of a computer program

Description automatically generated

1. AUTOTUNE = tf.data.AUTOTUNE:
   * Sets AUTOTUNE to tf.data.AUTOTUNE to dynamically adjust parallelism for data prefetching, improving data loading efficiency.
2. Training Dataset Optimization:
   * Applies prefetching to the training dataset (train\_dataset) to load data for the next batch while processing the current one, reducing training time.
3. Buffer Size Adjustment:
   * Sets the buffer\_size to AUTOTUNE, enabling TensorFlow to dynamically optimize the prefetch buffer size based on available system resources for optimal performance.
4. The code defines a data augmentation pipeline for image data using TensorFlow. It includes random horizontal flipping. The two commented-out lines that enable random rotation and zoom are often omitted due to potential increased computational load, overfitting risks, or project-specific data characteristics. The decision depends on dataset complexity and desired model generalization.

A screenshot of a computer program

Description automatically generated

tf.keras.layers.RandomFlip(): This layer randomly flips or mirrors the input data horizontally (left to right).

tf.keras.layers.RandomRotation(0.2): This layer, which is currently commented out, would apply random rotations to the input data, with a maximum rotation angle of 0.2 radians

tf.keras.layers.RandomZoom(0.2): This layer, also commented out, would randomly zoom in or out on the input data with a maximum zoom factor of 0.2.

1. these lines configure the base of the neural network model using the EfficientNetB0 architecture. It's set up to accept images with the specified dimensions and color channels, and it's initialized with pre-trained weights from ImageNet.

A screenshot of a computer program

Description automatically generated

Image\_shape = image\_dimensions + (3,):

* image\_dimensions likely represents the desired image dimensions, and (3,) indicates that there are three color channels (Red, Green, and Blue).

1. Setting base\_model.trainable = False in a neural network configuration freezes the weights of the layers in the base\_model, preventing them from being updated during the training process. In other words, it makes the pre-trained layers non-trainable, and they act as fixed feature extractors.

A screenshot of a computer program

Description automatically generated

1. Here I added two layers to your neural network model

A screenshot of a computer program

Description automatically generated

GlobalAveragePooling2D:Global average pooling computes the average value for each feature map in the input. This layer reduces spatial dimensions by averaging feature maps, often used for dimensionality reduction before classification or regression layers.

Dense(len(class\_names)): This line creates a fully connected layer (Dense layer) with a number of units equal to the number of classes in your dataset. This layer is used for making predictions. The number of units matches the output classes to produce class probabilities.

1. This code sets up a model for image classification, where the base\_model provides feature extraction, data augmentation enhances the data, and the final layers are responsible for making predictions. The model is ready to be compiled and trained for the specific classification task.

A screenshot of a computer program

Description automatically generated

* inputs = tf.keras.Input(shape=image\_shape): Defines the input layer.
* x = image\_augmentation(inputs): Applies data augmentation.
* x = base\_model(x, training=False): Passes data through pre-trained layers.
* x = global\_avg\_pooling(x): Applies global average pooling.
* x = tf.keras.layers.Dropout(0.2)(x): Adds a dropout layer.
* outputs = prediction(x): Defines the final prediction layer.
* model = tf.keras.Model(inputs, outputs): Creates the complete model architecture.

1. code is used to compile the model, specifying its optimizer, loss function, and evaluation metrics.

After compiling the model, it's ready to be trained using the specified optimizer, loss function, and evaluation metric. The learning rate (base\_learning\_rate) can be adjusted to control the rate of weight updates during training.

A screenshot of a computer

Description automatically generated

* base\_learning\_rate = 0.001: Here, we define a base learning rate of 0.001. The learning rate is a hyperparameter that controls the step size at which the model's weights are updated during training. A smaller learning rate generally leads to slower but more stable training, while a larger learning rate can lead to faster but potentially unstable training.
* loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),: Here, you specify the loss function to measure the difference between the predicted values and the true labels during training. An optimizer is responsible for updating the model's weights based on the gradients of the loss function.
* metrics=['accuracy']: This line specifies the evaluation metric to monitor during training. In this case, we're tracking the accuracy of your model, which measures how well it predicts the correct classes.

1. The model is training a neural network model using 5 epochs using the training dataset and validated on the validation dataset. Training history is saved in the history variable

A screenshot of a computer program

Description automatically generated

* history = model.fit(...): This line trains the model, capturing training and validation performance history.
* train\_dataset: The training dataset used for model training with input images and labels.
* epochs=5: Specifies the number of training iterations (epochs), set to 5.
* validation\_data=validation\_dataset: Evaluates the model on unseen validation data after each epoch.

1. Here the model evaluates the model's performance on a testing dataset and displays the accuracy.

A screenshot of a computer program

Description automatically generated

* evaluation\_result = model.evaluate(test\_dataset): This line evaluates the model's performance using the testing dataset. It computes various metrics, including accuracy, and stores the results in the evaluation\_result variable.
* print("Accuracy: {:.3f}%".format(evaluation\_result[1] \* 100)): This line prints the accuracy on the testing dataset. It formats and displays the accuracy as a percentage with three decimal places.

4.training curves

This section of the code is responsible for visualizing training and validation metrics, such as accuracy and loss, using Matplotlib.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

5.confusion matrix

We retrieve the class labels from the original dataset directory and prints them to use it for confusion matrix.

A computer screen shot of a computer screen

Description automatically generated

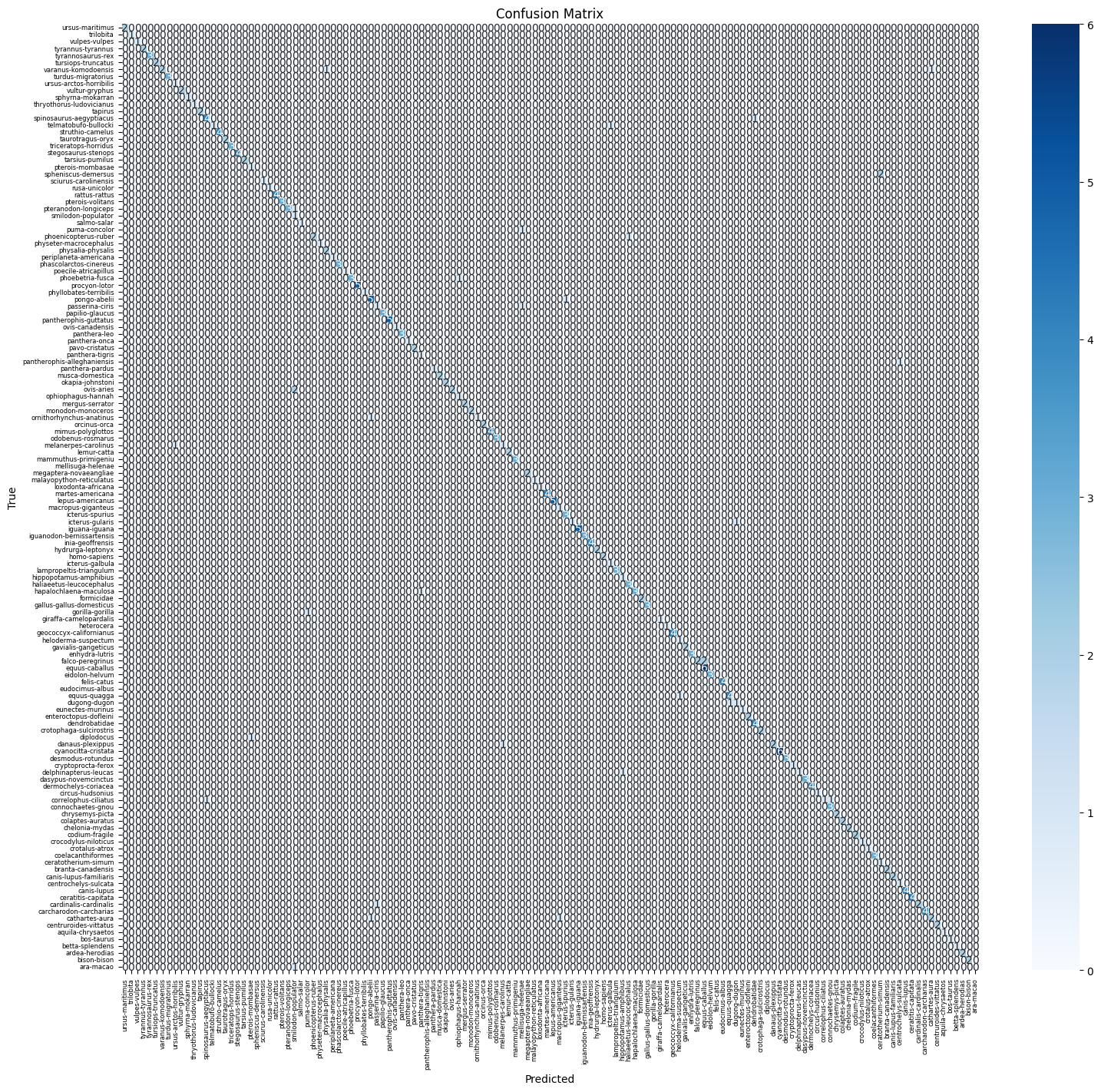
This part of the code is responsible for creating and visualizing the confusion matrix, which is a crucial tool for evaluating the performance of a classification model.

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

* true\_labels and pred\_labels are lists initialized to store true and predicted class labels, respectively.
* A loop iterates through the test dataset to load image batches and their true labels.
* True class labels are extracted and added to true\_labels.
* The model makes predictions on image batches, generating predicted class probabilities.
* Predicted class labels are obtained by selecting the class with the highest predicted probability for each image.
* The predicted class labels are added to pred\_labels.
* class\_names is a list that defines class names based on your dataset.
* The confusion\_matrix function computes the confusion matrix by comparing true and predicted class labels.
* A heatmap is created using Seaborn's heatmap function, visualizing the confusion matrix.
* Labels are set for the x-axis (predicted) and y-axis (true) of the heatmap.
* Font sizes for tick labels are adjusted for better readability.
* The title "Confusion Matrix" is added to the heatmap.
* The confusion matrix visualization is displayed.



6.conclusion

In conclusion, our project successfully achieved a test accuracy of over 60% in the image classification task using the "Animals 151" dataset. We employed data preprocessing, data augmentation, and a transfer learning approach with the EfficientNetB0 model, demonstrating the effectiveness of these techniques. The visualization of the confusion matrix offered insights into the model's performance across different classes. Overall, this project highlights the potential of deep learning and transfer learning for image classification and serves as a valuable foundation for further research and applications in this field.

7.refrences

[1.Kaggle Dataset](http://www.kaggle.com/)

[2.TensorFlow Documentation](https://www.tensorflow.org/)

[3.EfficientNet Paper](https://arxiv.org/)

[4.Seaborn Documentation](https://seaborn.pydata.org/)