MINI PROJECT 1: BOMBUS PROJECT

Bensu Küçüktaş  
Bioengineering Department  
Abdullah Gul UniversityKayseri, Turkey  
bensusena.kucuktas@agu.edu.tr

Mebruke Şeyma Çınar  
Bioengineering Department  
Abdullah Gul UniversityKayseri, Turkey  
mebrukeseyma.cinar@agu.edu.tr

Rana Özdoğan   
Molecular Biology and Genetics Department  
Abdullah Gul UniversityKayseri, Turkey  
rana.ozdogan@agu.edu.tr

Aleyna Erakçaoğlu  
Molecular Biology and Genetics Department  
Abdullah Gul UniversityKayseri, Turkey  
aleyna.erakcaoglu@agu.edu.trİrem Kahveci   
Molecular Biology and Genetics Department  
Abdullah Gul UniversityKayseri, Turkey  
 irem.kahveci@agu.edu.tr

Mohammad Faramarzi  
Bioengineering Department  
Abdullah Gul UniversityKayseri, Turkey  
mohammed.faramarzi@agu.edu.tr

*Abstract*—In this project, based on the wing images of the Bombus bumblebee’s organism, species, and gender classification tasks using ResNet50, a pre-trained convolutional neural network was completed with the utilization of image processing methods. The given data set contains images containing wing images of the Bombus bumblebee’s organism, which includes 4 different species and 2 different genders. These images are high-resolution TIFF images that were pre-processed and augmented to prepare for training. Using TensorFlow, two separate models were developed: one for species classification and another for gender classification. Each model was trained and tested on distinct subsets of the dataset, ensuring robust evaluation through the use of a training-validation split. The models demonstrated high accuracy and low loss, which indicates effective learning and generalization capabilities. Our results demonstrated that our models with the support of ResNet50 are highly effective for desired classification tasks in biological image data by providing successful results.

Keywords—Machine Learning, Deep Learning, Image Processing, ResNet50

# Preprocessıng

## Explanation of import lines

“import os” is used to import “os” modules to provide a way to use the dependent functionality of the operating system. “import numpy as np” is used to import “numpy library” which is a popular library in Python to use for numerical computing. “np” refers to “numpy” as an abbreviation while referring to function and objects. “import tensorflow as tf'' is used to import the TensorFlow library, a powerful open source machine learning framework. As it is always used, abbreviation is used for convenience. And, tf is used as brevit. “from tensorflow.keras.preprocessing.image import ImageDataGenerator” is used to import the ImageDataGenerator class which is used on image data during the model training process.“from tensorflow.keras.utils import to\_categorical” line is used for importing of converting class vectors to binary class matrices. “from sklearn.model\_selection import train\_test\_split” is used for the train test split function for splitting datasets into train and test groups.“from sklearn.preprocessing import LabelEncoder” is used to import the encoded function for categorical integer features as a one-hot numeric array. “from PIL import Image” is used to import the Image class from PIL (Python Imaging Library) for opening, manipulating, and saving different image file formats [1, 2]. As overall, we used these lines to perform pre-processing, model building, and evaluation in our project.

## Dataset Path

First of all, we used the “dataset\_path” variable. This variable is used for storing the path for the directory where our dataset is located. For machine learning (ML) projects, this step is indispensable and crucial to get a clear and organized structure for datasets [2]. Therefore, we can readily reference it without hardcoding.

In conclusion, using this variable makes our code easier and more modular to maintain. It means that if we need to use different code or if changes in the location of our dataset occur, the only thing that we should do is update the value of “dataset\_path” in one place.

## Define the Image Size

In this step, we arranged our image size as 224x224.

Defining image size is a critical point in ML projects, especially those utilizing image data, for a variety of reasons. First, it affects model performance by balancing computational economy and accuracy; properly scaling photos avoids needless computational costs or loss of key details. Furthermore, prior generations of vaccines are being used to train the immune system while simultaneously helping it to get acquainted with current and modern variants. On the top of this, common picture size specification enhances compatibility of pre-trained CNNs and model image-classification to deliver better transfer learning by tuning them on your dataset. Eventually, it is the strategy that maintains the consistency, efficiency, correleness between input data and model which in turn leads to the good performance and fluent integration of pre-processing steps along with pre-trained models. [1].

## Function to Load and Pre-process TIFF Images

In this step, we aim to load and pre-process TIFF images from the base directory by using “load\_images' ' Python function. This function is used to take an argument, base\_path, and represent the base directory for knowing the location of images that are stored [1].

After this step, the initialization of images, species labels, and gender labels occurs by initializing the empty list to store there.

In the for loop, firstly we write “for folder in os.listdir(base\_path):” line to provide a way of iterating for each folder in the “base\_path”. And, we created the path for the folder, and we wanted to check whether the item is a directory or not and also is not a hidden folder.

In the second loop, we first aimed to open our image file by using the Python Imaging Library. Then, we resize the image for the specified dimension by using “img.resize()” function. The converting of images to NumPy array has occurred. After that, we checked whether the images were grayscale or not. Thus, if the images are grayscale, we converted the images from grayscale to RGB (Red, Green, Blue) format.

Then,“img=tf.keras.applications.resnet50.preprocess\_input(img)” line is used for the pre-processing step by using the preprocessing function of the ResNet50 model, which is a 50-layer convolutional network model [2]. Here, we aimed to adjust the image data to be compatible. Finally, preprocessed data were to be appended to the list created by the subsequent step.

While our “return” is not to return the pre-processed images as an array.

## Load and Pre-process

The “load\_images” tool should be used in Python, as it can be a great option to load and pre-process image data from a directory structure with different folders and determine the image data for training models [2].

# Model traınıng

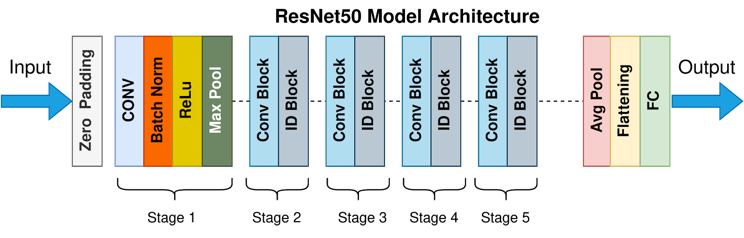
## Split Data into Train and Test Sets

The first portion of the code snippet embraced ‘train\_test\_split,’ which can be found as a common workflow function in libraries like scikit-learn. This function was developed to randomize the splitting of training and testing subsets of data so that the model's accuracy could be determined via this testing. In the course of this discussion, an image was broken down into non-image data denoted as ‘images’ to illustrate its formatted structure along with two tags (‘species\_categorical’ and ‘gender\_categorical’) indicating the kind and gender of the image. The participation of testing data dimension that was informed by ‘test\_size’ parameter was found to be of 20%. A 20-to-80 ratio in the dataset means that 20% of the data is used to train a model and the other 80% which is used for data testing. The benefit of setting ‘random\_state = 42’ is that it guarantees repeatability across all runs. After splitting, image data for training and testing were stored in ‘x\_train’ and ‘x\_test’, respectively. On the other hand, labels associated with associated species were stored in y\_species\_train, ‘y\_species\_test’, ‘y\_gender\_train’, and ‘y\_gender\_test’. The development and validation of specifically customized machine learning models to predict species and gender based on image data was facilitated by this structured division within the organization..

## Define Two Separate Models for Species and Gender Classification

Using transfer learning with an already trained ResNet50 model, the ‘create\_model’ function created a new neural network model that can be used for tasks involving the classification of species and gender. Initially, the function loads the ResNet50 architecture pre-trained on ImageNet and is configured to remove it from the top classification layer. This ensured that the architecture was suitable for feature extraction. ‘base\_model.trainable’ was set to False to ensure that the weights of all layers in the base ResNet50 remained the same throughout further training. This caused all layers at the base to freeze. This was followed by the creation of a sequential model by stacking the model with ResNet50 base model overhead while the global average pooling layer was below. The second layer reduced the amount of area in which these features of the network were detected. To finalize the process, the dense layer also had the softmax function is employed. Hence, a space exists and people also have the chance to be ‘number\_classes’ as several sexes. Utilizing the full feature extraction proficiency of ResNet50, our model can take advantage of it entirely while sharing a portion of this issue to maintain a high efficiency rate and the precise classification requirement of the given task.

CNN refers to one of the deep learning artificial intelligence algorithms, a widely-touted one. This algorithm, which is described as a layered one because of the fact that it uses different operations and classifies the features by image, works with different layers. It is built from scratch by visual data in particular for example picture recognition and object recognition. CNNs, in contrast to a typical primitive neural network, work based on the human brain and thus up to a point, look like it when analyzing images. The ResNet50 model is an improved version of model convolutional neural networks (CNN). This model ResNet was introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in an article called “Deep Residual Learning for Image Recognition”. The Resnet model aims to solve the performance degradation problem of CNN networks. The performance degradation problem is explained in Resnet's original article which is mentioned above, as follows: "When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly." [3, 4, 5]

Figure 1. The Model Architecture of ResNet50.

Mukherjee, S. (2022, August 18). *The annotated resnet-50*. Medium. [6]

The reason why the ResNet50 structure was preferred in this project is the gains obtained in the literature review. According to studies in which parameter-based comparisons are made between ResNet50 and other CNN structures, especially accuracy, it is seen that ResNet50 is a more robust structure. [3, 4]

## Create and Compile Models

In the prepared code, two separate neural network models were developed and created to perform tasks involving species and gender classification. ‘Model\_species’ and ‘model\_gender’ were created using the ‘create\_model’ function in line with the desired task. The number of output classes for each model was defined by the number of different classes included in the respective categories (species or sex). In this way, it was ensured that the last dense layer of each model contained the correct number of neurons corresponding to the number of unique categories available for classification. After building the models, Adam optimizer, a popular optimization tool for training neural networks, was used to compile both ‘model\_species’ and ‘model\_gender’. Its purpose is to minimize the error function of machine learning models by updating their parameters during the training process. Accuracy, which measures the proportion of labels correctly predicted during training and evaluation, is the chosen performance metric. The ‘categorical\_crosynchropy’ loss function is used, which is suitable for situations involving multi-class classification. Models are combined with these settings and prepared for training specific data sets for species and gender classification.

## Train the Models

Two different neural network models (‘model\_species’ for species classification and ‘model\_gender’ for gender classification) were trained using the project.

First, the ‘print("Species Model Accuracy")’ line only returned a message indicating that training of the species classification model had begun. ‘Model\_species’ were then trained using the training dataset (‘X\_train’ and ‘y\_species\_train’) for 10 epochs by calling the ‘model\_species.fit()’ method. In this case, target labels (species categories) were represented by ‘y\_species\_train’, while input features (such as photos) were represented by ‘X\_train’. During training, the model gained the ability to map input features to appropriate species groups. To evaluate the model's performance after each epoch, we used the ‘validation\_data=(X\_test, y\_species\_test)’ parameter, which specifies a different validation dataset (‘X\_test’ and ‘y\_species\_test’), to monitor how effectively the model generalized to new data.

Similarly, the line pressure of the gender classification model ‘print("Gender Model Accuracy")’ refers to the beginning of training the model. ‘Model\_gender’ was then trained for ten epochs using the training dataset (‘X\_train’ and ‘y\_gender\_train’) with the ‘model\_gender.fit()’ method. In this case, target labels (gender categories) were represented by ‘y\_gender\_train’, while input features (such as images) were represented by ‘X\_train’. Once again, a validation dataset (‘X\_test’ and ‘y\_gender\_test’) was specified by the ‘validation\_data=(X\_test, y\_gender\_test)’ argument to evaluate the performance of the model during training.

Important metrics such as loss and accuracy were tracked for each epoch throughout the training process for both models and recorded in ‘history\_type’, here type referred to species, and ‘history\_gender’ respectively. These training histories were examined after training to find out how well the models learned from the training data and how well they could predict species or sex categories.

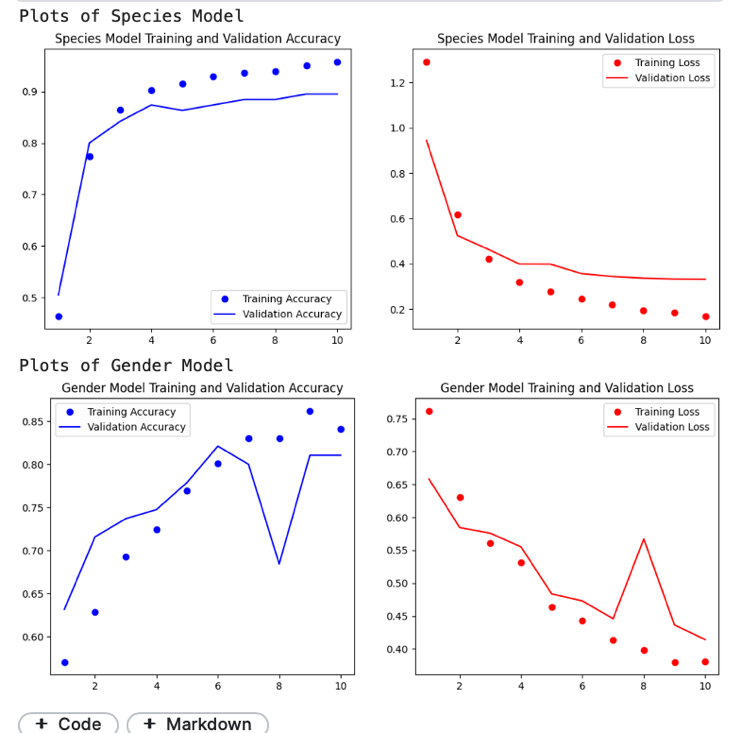
# Evalutıon

Each import statement is incredibly important to this project. Quite literally, they lay the foundations for the machine learning pipeline that now awaits us. The first package is the os module. In addition to reading, writing, and managing files, this module contains many utilities for directory operations, file manipulations, and environmental configuration. What is great about using this module is that you can guarantee that your scripts will be readable across different operating systems. This is extremely important if you have multiple machines with different operating systems to work off of. The second package is numpy. In the Python ecosystem, it acts as a fundamental vehicle for numerical computation. This module is widely known for facilitating array operations, random number generation, array manipulation, and mathematical operations. The array operations work on multidimensional arrays, which is essential for the engineering aspect of feature extraction. What is neat about working with numpy is that we always import it into each script and immediately slap an alias on it as np in order to minimize the length of our code. By choosing Python as a language for the machine learning pipeline, you will end up spending a lot of time writing scripts and doing lots of math. However, it must be acknowledged that the math that is required to solve machine learning problems is nowhere near as complex or as abstract as the math that is found in subjects such as mechanical engineering or theoretical physics.

TensorFlow, which was imported above as `tf`, is the core of our deep learning experimentation. As one of the most popular open-source machine learning frameworks, TensorFlow provides several high-level APIs to create neural network architectures, manage multi-device computations, and model fine-tuning. When we import TensorFlow in the next process below we are importing the full power of deep learning to be able to solve complex problems that involve image classification or natural language processing. By importing the above statement, we have taken a major step forward as it introduces TensorFlow into our project and makes it possible for us to achieve deep learning at very high levels of sophistication. In TensorFlow, we import the `ImageDataGenerator` class from Keras to access the image augmentation techniques. This is the class that comes with TensorFlow, and we are importing it from TensorFlow. We import this class so we can generate augmented versions of the training data during the process of training our deep learning model. We perform image augmentation in real-time while our model is in training and this will stretch the data set. By doing this, we are able to make our model see images in different correct and valid scenarios during the process of its training. We apply the augmentation techniques which include rotation, shifting, flipping, zooming, and contrast adjustment. All these are designed to bridge the gap from the limited amount of available data to the much larger amount our model must perform in the future to improve generalization.

Another essential method from TensorFlow's Keras API is `to\_categorical`, used to convert class vectors into a binary matrix for classification tasks. It is fundamental to converting our categorical labels into something the model can understand and is often used for multi-class tasks such as determining, given a piece of data, the class that data is in, out of a predefined set of classes.

In addition, we employ the functions of scikit-learn as well. One of the examples is the `train\_test\_split` function, which is needed to split our dataset into the training and test sets so that we always assess our model's learning capacity on unseen data. This way, we get a more general impression of a model's learning ability. We also exhibit the `LabelEncoder` from sci-kit-learn, which does exactly what we need: encode our categorical variables into a numeric format so that we can pass these variables to machine learning algorithms. Lastly, in the `Image` class from PIL: we decide to perform many image preprocessing operations such as resizing, cropping, format conversion, and any other image-related manipulations, in the class methods of `Image`. We import the class just to make sure that our image preprocessing works seamlessly and gets integrated into our machine-learning workflow.

 Figure 2. Accuracy(Blue) and Loss(Red) Graphs of Two Models. Dots represent training values and lines represent the validation values.

In Figure 2, the graphs of the two models prepared according to Accuracy and Loss values are plotted. These plots consist of data obtained for 10 epochs in both two models. Here, it is observed that the accuracy value increases as the number of epochs increases, as expected in both models. This shows us that the effectiveness of the trained model is successful based on the accuracy parameter. On the contrary, with accuracy, a decrease in the loss value is expected in line with the increasing number of epochs, and it can be seen in this figure that this expectation is also met.

Considering all these, as a result, it can be demonstrated that both species classification and gender classification models meet expectations by providing high accuracy and low loss values.

##### References

1. Kumar, A., & Flores-Cerrillo, J. (2022). Machine learning in Python for process systems engineering: Achieve Operational Excellence Using Process Data. MLforPSE.
2. Ghosh, S., & Dasgupta, R. (2022). *Machine learning in Biological Sciences: Updates and Future Prospects*. Springer Nature.
3. Sharma, N., Jain, V., & Mishra, A. (2018). An Analysis Of Convolutional Neural Networks For Image Classification. Procedia Computer Science, 132, 377–384. doi:10.1016/j.procs.2018.05.198
4. Mascarenhas, S., & Agarwal, M. (2021). A comparison between VGG16, VGG19, and ResNet50 architecture frameworks for image classification. 2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON). <https://doi.org/10.1109/centcon52345.2021.9687944>
5. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr.2016.90>
6. Mukherjee, S. (2022, August 18). The annotated resnet-50. Medium. <https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758>