Università di Catania

Animal Image Classification using Deep Neural Network

Data Science for Management (2022-2023)

Neural Computing

Submitted to: Prof. Sebastiano Battiato

Submitted by: Ashish Singh Bisht (1000039457)

Table of Contents

Introduction

Animals are multicellular, eukaryotic organisms in the biological kingdom Animalia. Over 1.5 million living animal species have been described, of which around 1 million are insects.

Animals range in length from 8.5 micrometers (0.00033 in) to 33.6 meters (110 ft). They have complex interactions with each other and their environments, forming intricate food webs.

Most living animal species are in Bilateria (animals with a bilateral symmetry as an embryo), a clade whose members have a bilaterally symmetric body plan.

The classification of animals based on their images is a challenging task that has many important applications in fields such as wildlife conservation and research. In this study, we aimed to explore the potential of deep neural networks for solving this problem. By training our model on a large dataset of animal images, we aimed to develop a tool that can accurately identify different species of animals in the wild.

This report presents the results of our research on using deep neural networks for animal image classification. Our goal was to develop a machine learning model that can accurately identify different species of animals based on their images. We gathered a diverse collection of animal images and used them to train and evaluate on their image. We gathered a diverse collection of animal images and used them to train and evaluate our model.

The results of our study showed that deep neural networks can effectively classify different species of animals with high accuracy.

The Dataset was taken from Kaggle and can be found by following this link :

https://www.kaggle.com/datasets/iamsouravbanerjee/animal-image-dataset-90-different-animals?select=name+of+the+animals.txt

Dataset Details

This Dataset contains 5400 Images, 60 Images for each of the following 90 different animals,

Antelope , Badger, Bat ,Bear, Bee, Beetle, Bison, Boar, Butterfly, Cat, Caterpillar, Chimpanzee, Cockroach, Cow, Coyote, Crab, Crow, Deer, Dog, Dolphin, Donkey, Dragonfly, Duck, Eagle, Elephant, Flamingo, Fly, Fox, Goat, Goldfish, Goose, Gorilla, Grasshopper, Hamster, Hare, Hedgehog, Hippopotamus, Hornbill, Horse, Hummingbird, Hyena, Jellyfish, Kangaroo, Koala, Ladybugs, Leopard, Lion, Lizard, Lobster, Mosquito, Moth, Mouse, Octopus, Okapi, Orangutan, Otter, Owl, Ox, Oyster, Panda, Parrot, Pelecaniformes, Penguin, Pig, Pigeon, Porcupine, Possum, Raccoon, Rat, Reindeer,

Rhinoceros, Sandpiper, Seahorse, Seal, Shark, Sheep, Snake, Sparrow, Squid, Squirrel, Starfish, Swan, Tiger, Turkey, Turtle, Whale, Wolf, Wombat, Woodpecker, Zebra

The size of the dataset is 698 Mb where there are 60 images of each animal and later, we divide them into test and training datasets.

We will see the examples of the dataset later when we import it in Matlab.

Network Design

In this study, we used deep neural networks to classify animal images. We used a Convolutional Neural Network (CNN) architecture, which is well-suitable for image classification tasks. We trained our model on a dataset of 698 MB animal image and evaluated its performance using **confusion matrix**.

CNNs are Neural Networks which use a function called convolution which consists into the application of a filter sliding over different positions of an input image to learn its characteristics. CNN is a concatenation of three different types of layers: the input the convolution and the output layers. Specifically, the convolution layer puts together different filters with different weights. That all act on the same image.

There are numerous well-known pre-trained models and structures (**AlexNet, ResNet, Inception**, etc.), and we may utilize transfer learning techniques to profit from these models in our projects and create models quickly and efficiently.

ResNet-50, a significantly modified version of the original ResNet-34, will be used in this project.

ResNet-34, a version of the original ResNet design, included 34 weighted layers. By utilizing the idea of shortcut connections, it offered a creative solution for expanding the number of convolutional layers in a CNN without encountering the vanishing gradient issue. A shortcut link turns a conventional network into a residual network by "skipping over" some levels.

The architecture of ResNet-50 is based on the concept shown above, with one significant exception.

The bottleneck building block is used in the 50-layer ResNet. A bottleneck residual block, sometimes referred to as a "bottleneck", employs 1x1 convolutions to cut down on the number of parameters and matrix multiplications. This makes each layer's training significantly faster. Instead of using a stack of two levels, it employs three layers.

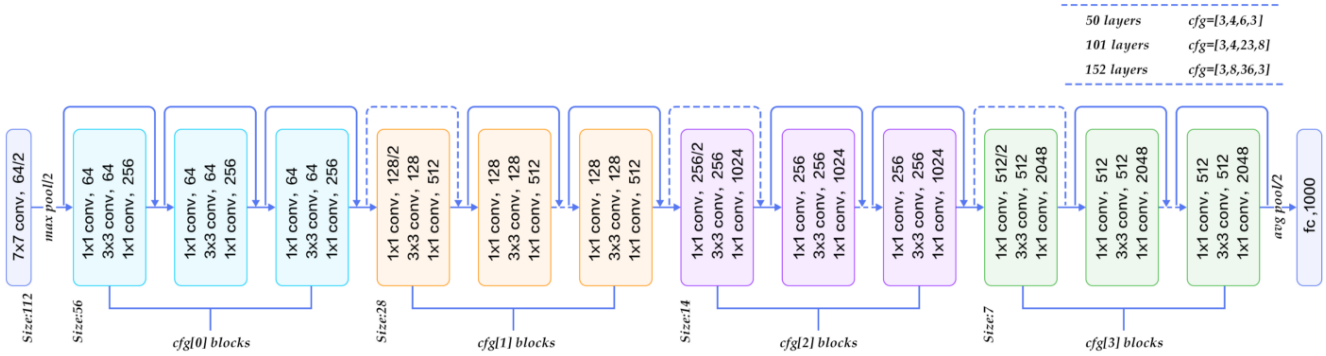
The **ResNet 50** uses an architecture includes the following elements as shown below: -

The **ResNet-50** architecture can be broken into 6 parts

1. Input pre-processing
2. Cfg[0] blocks
3. Cfg[1] blocks
4. Cfg[3] blocks
5. Cfg[3] blocks
6. Fully-connected layer

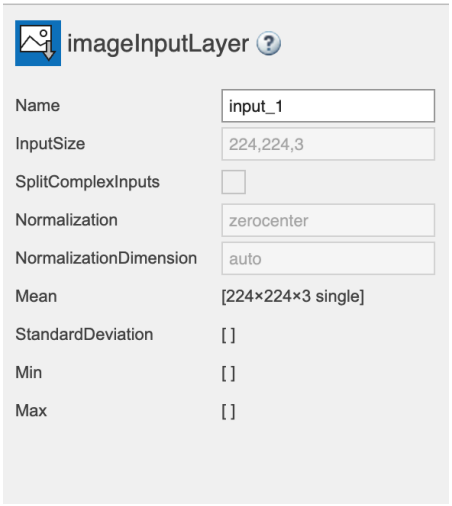
Different versions of the ResNet architecture use a varying number of “Cfg” blocks at different levels, as mentioned in the figure above. A detailed, informative listing can be found below.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 [3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				



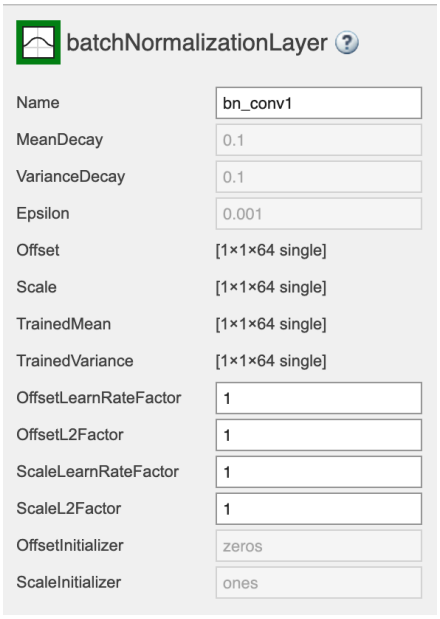
From the beginning in the Matlab architecture of **ResNet 50** we have:

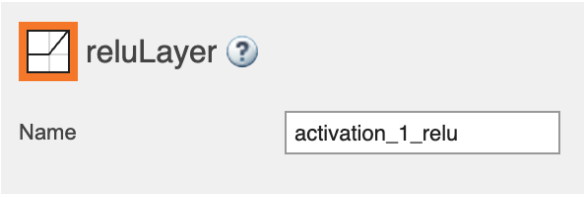
- Image Input Layer that accepts images in the size of 224, 224 with 3 channel colors.



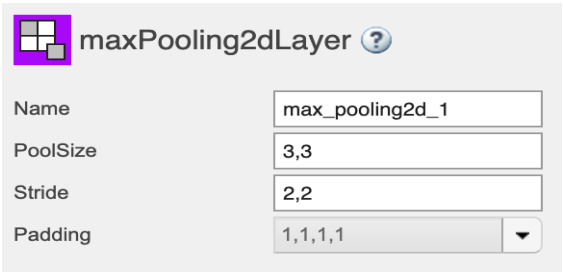
- 7x7 kernal convolution with 64 Filters and a size 2 sized stride

- Batch normalization layer followed by a Relu Activation Layer

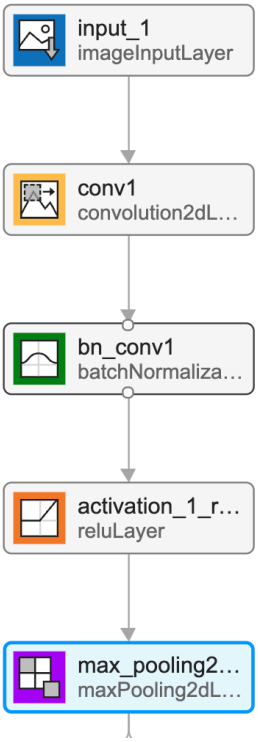




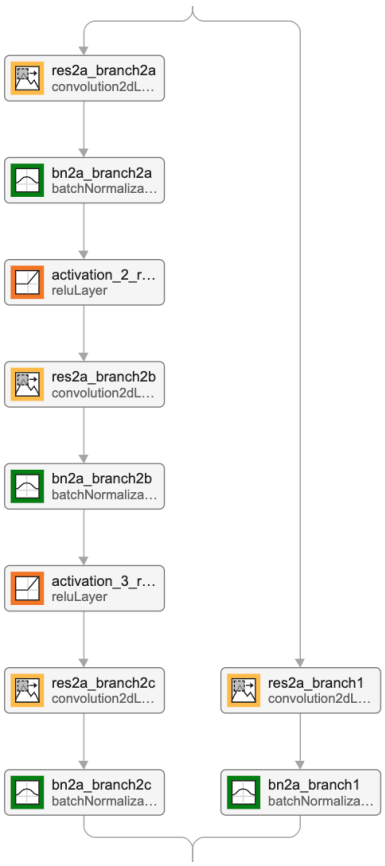
- Max pooling layer with a 2-sized stride.



All the above layers connected look like this:



- 9 more layers – 3x3, 64 convolution, another with 1x1, 64 kernals, and a third with 1x1256 kernals . These 3 layers are repeated 3 times.

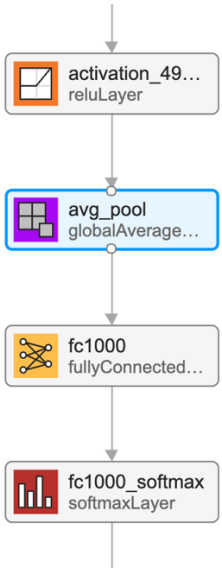


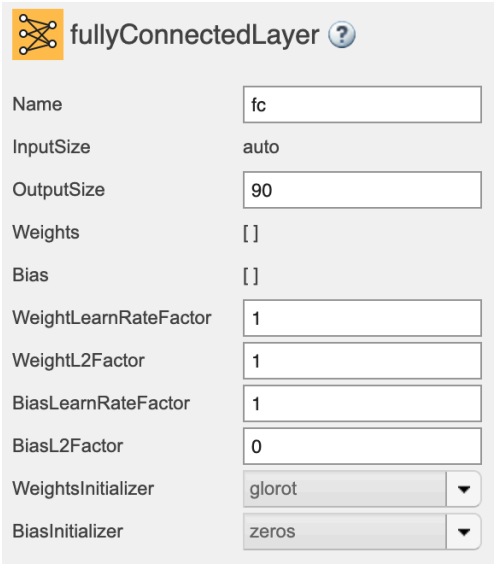
- 12 more layers with 1x1,128 kernels, 3x3, 128 kernals and 1x1,512 kernals, iterated 4 times.

- 18 more layers with 1x1,256 cores, and 2 cores 3x3, 256 and 1x1,1024, iterated 6 times.

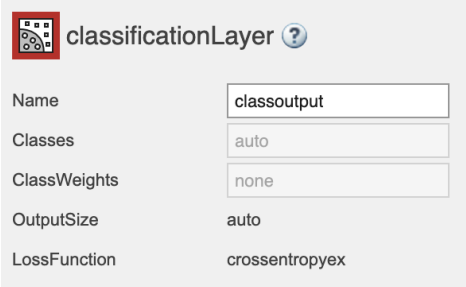
- 9 more layers with 1x1,512 cores, 3x3,512 cores , and 1x1,2048 cores iterated 3 times

- Average pooling is a fully connected layer with 36 nodes, using the softmax function.
-



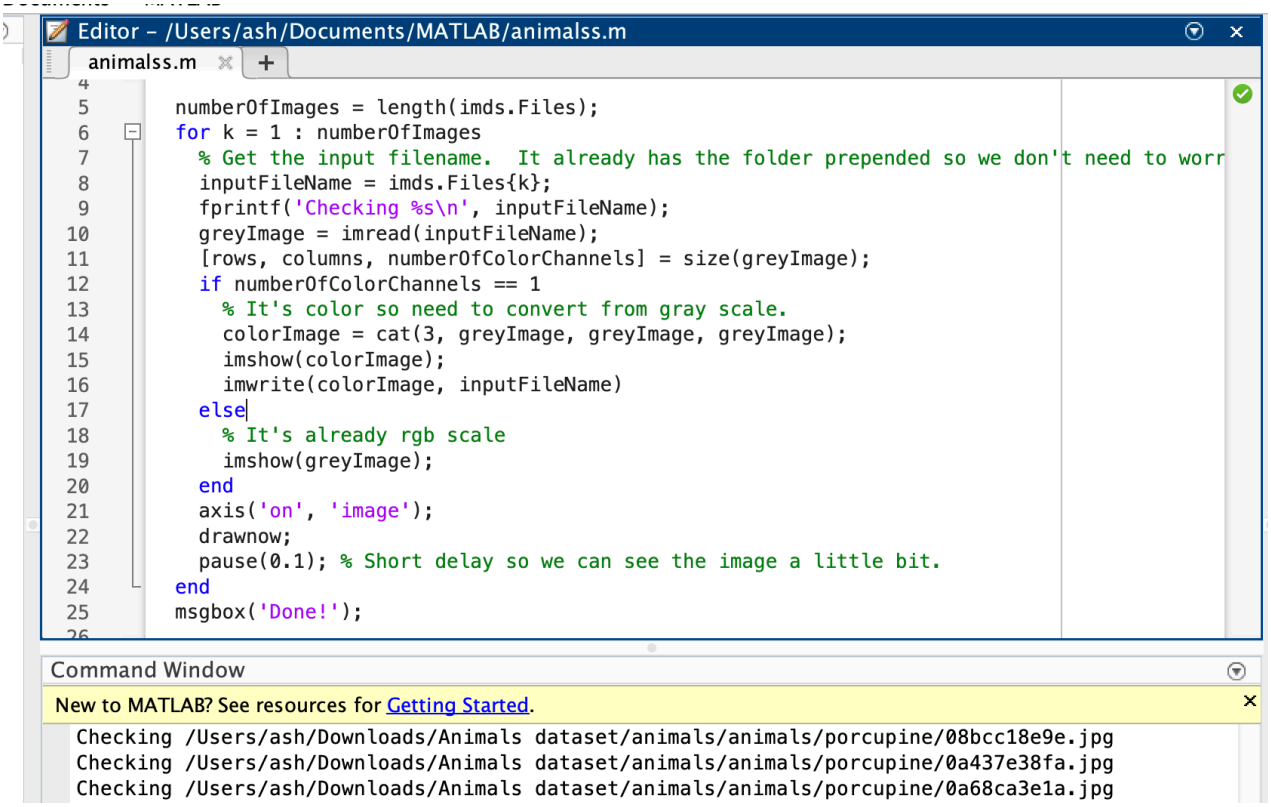


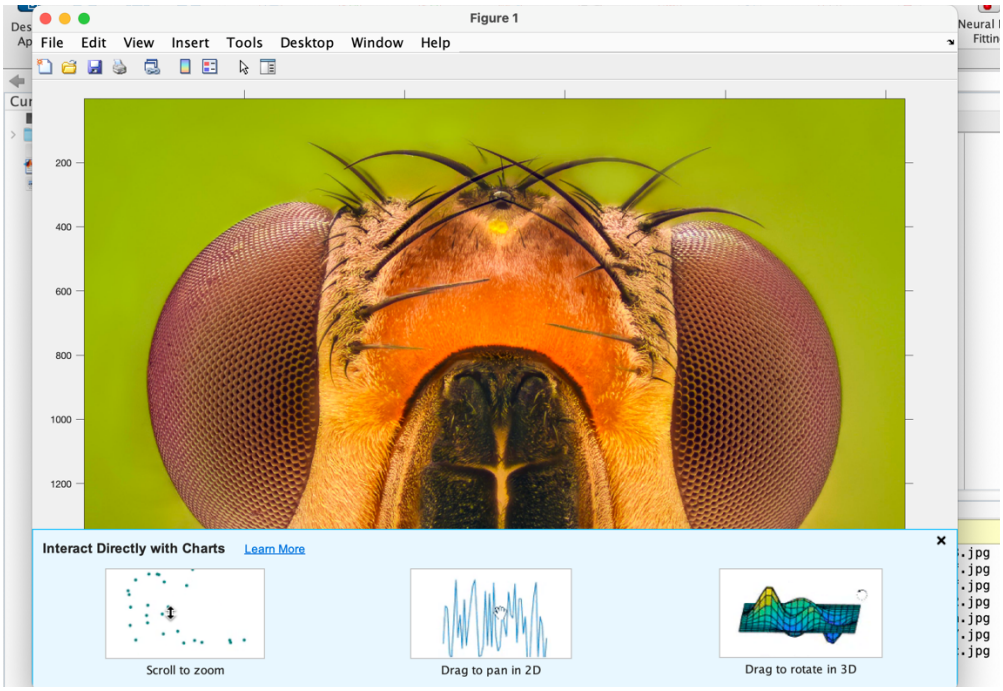
- Lastly we add the classification layer



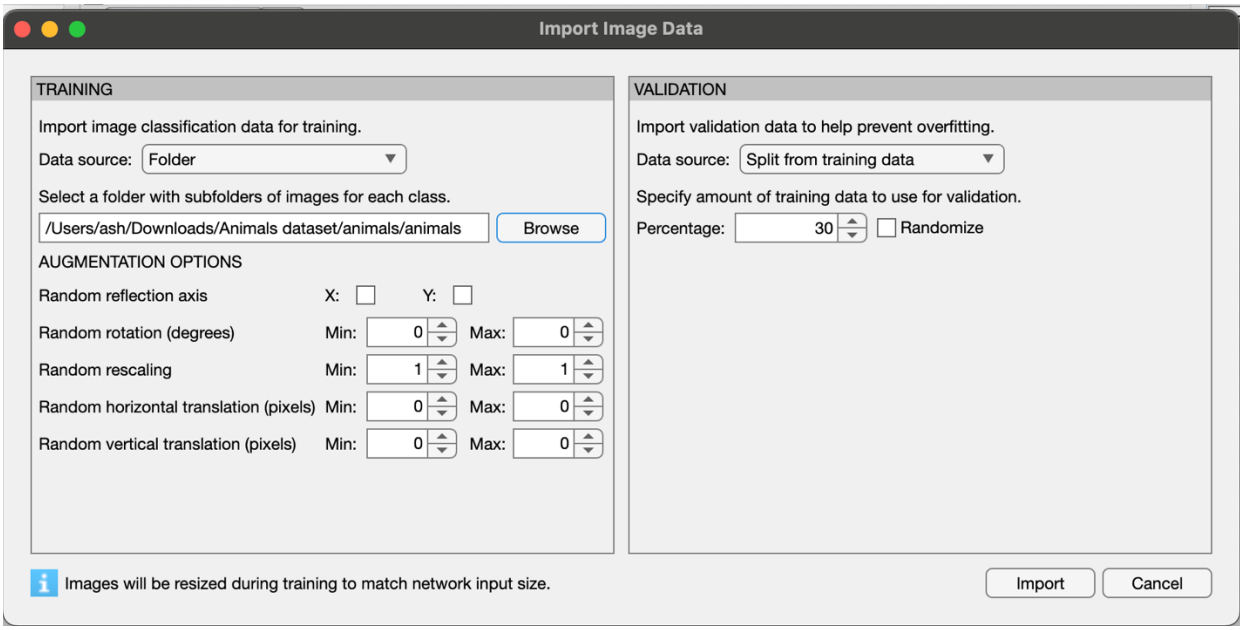
Importing Data

Now we start by importing our data into MATLAB so that we can start working on it. We use some commands to import our data and then to change the images into greyscale.

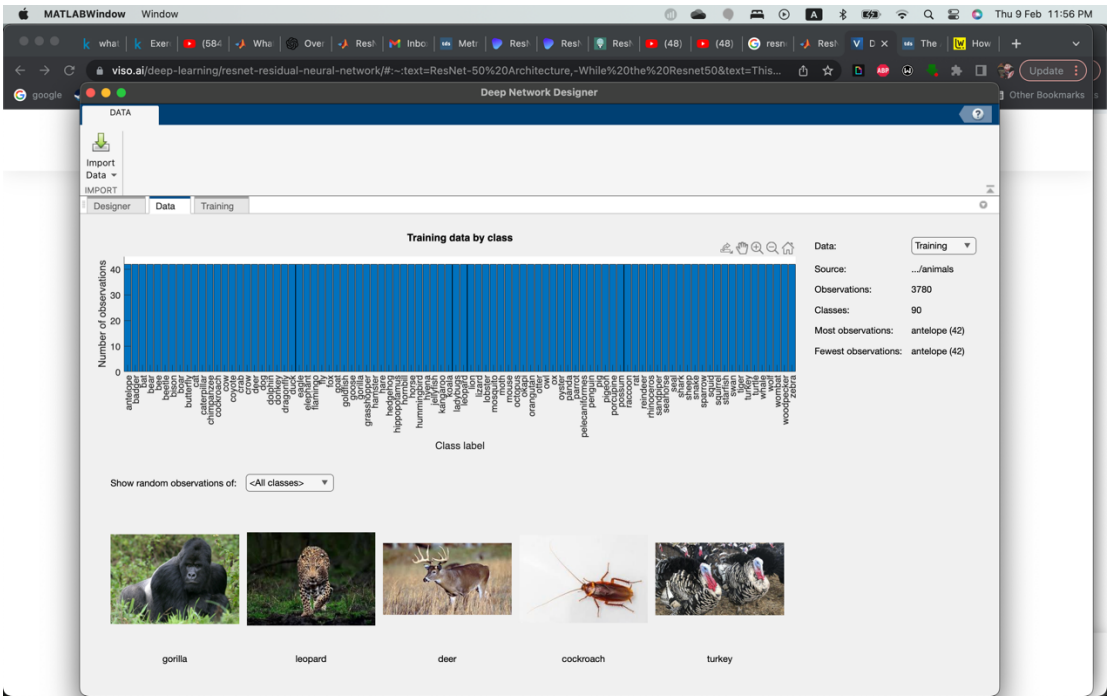




Here after the images are converted for better processing now we can proceed further.



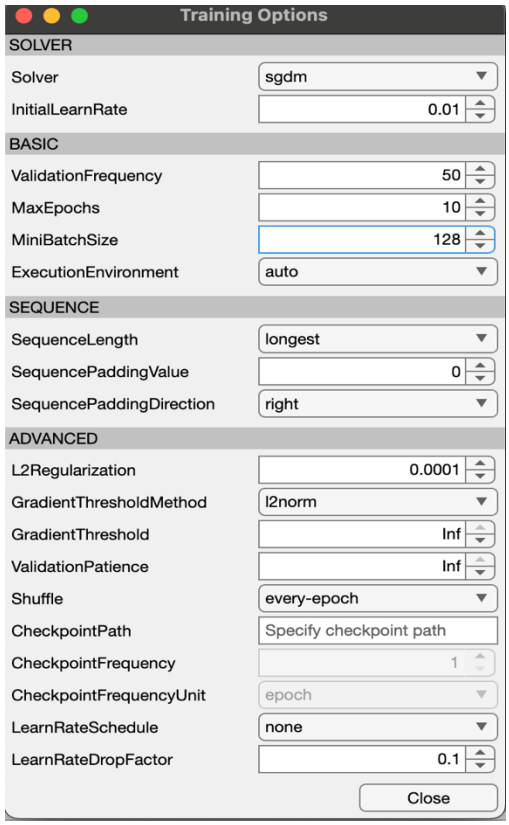
Here we have divided the data into 70 -30% into training and validation data. It will look like this before we start the Training.



Training the model

Before starting the training, the model, we must set the hyperparameters.

The “sgdm” (Stochastic Gradient Descent with momentum) will be used to update the gradient descendent. This strategy is the most stable and the most generalized one.



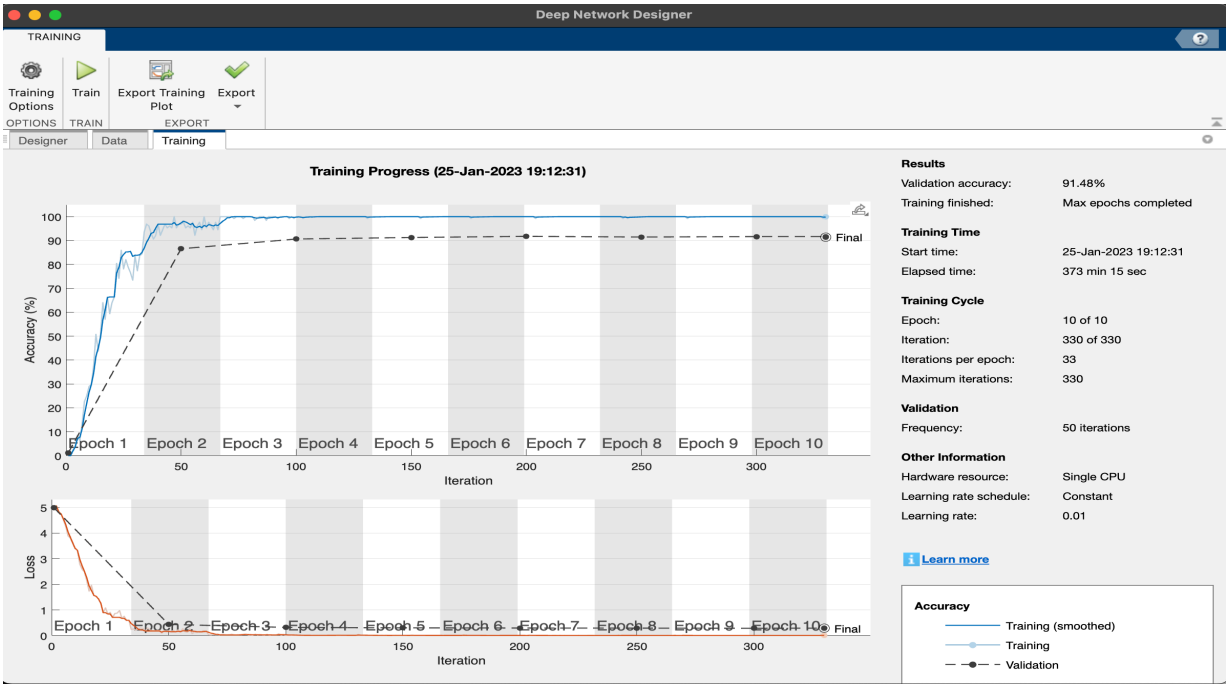
The Learning rate was set to 0.01 which achieved good results.

Then we have the Validation Frequency set to 50 and the MaxEpochs as 10. Each epoch is made by 33 iterations. After each 50 iterations we will be able to observe the validation accuracy of the corresponding value.

The L2Regulation is set to 0.0001 (by default) this type of regularization penalizes the large weights.

The other settings were left on default.

We start training our model now.



As observed by the training graph we can see that it after only 2 Epochs we reached an accuracy and validation rate of higher than 80%. The model finished all the iterations in under 380 minutes and resulted with the validation accuracy of 91.48%.

Testing the model

We have trained our model and achieved good results for validation accuracy so now we can head on to the test our data using the following code:

```
>>[imdsTrain, imdsValidation, imdsTest] = splitEachLabel(imds,0.6,0.2,0.2);
```

- We divide the tables into 3 parts and then we test our accuracy with:

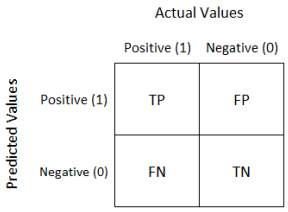
```
>>auimds_test = augmentedImageDatastore([224 224], imdsTest)
    YTest = imdsTest.Labels
    YTest_pred = classify(trainedNetwork_1, auimds_test)
    accuracy_test = mean(YTest == YTest_pred)

accuracy_test =

    0.9148
```

The accuracy of the model against the testing dataset is 91.48% which is the same as the validation frequency and we can state that the model performed well.

Finally we can move on to analyzing the confusion matrix.

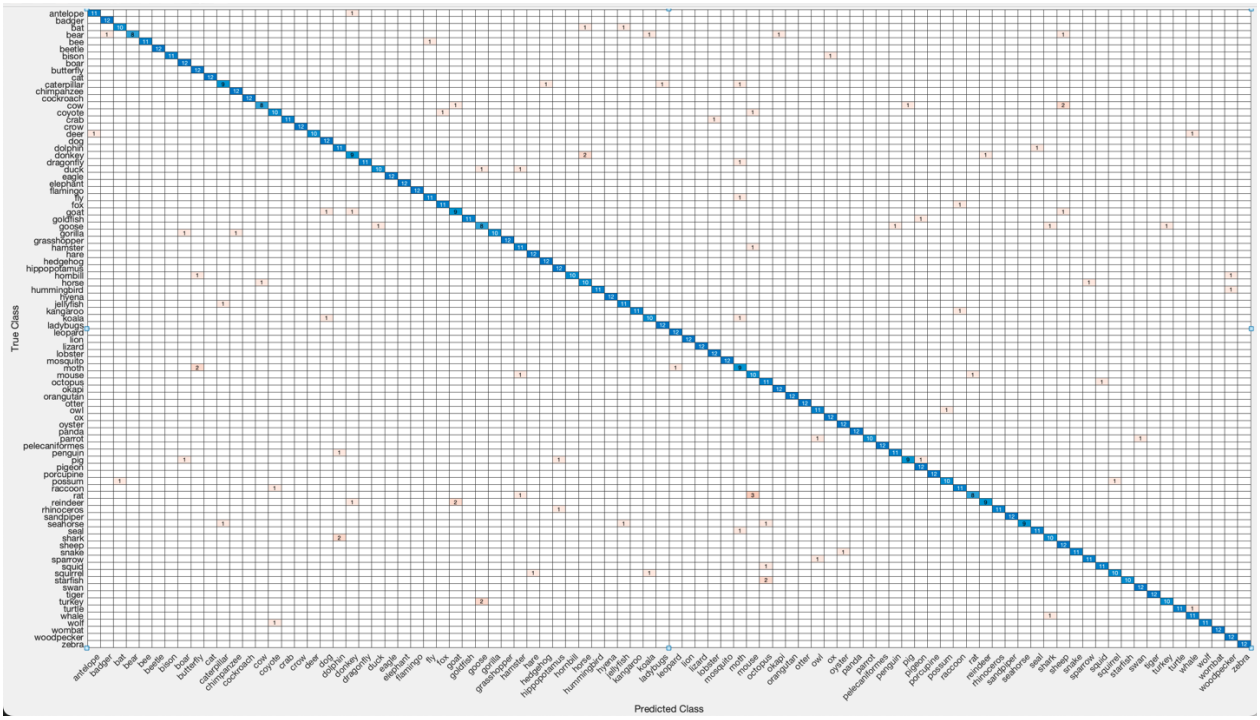
Confusion Matrix

A confusion matrix or error matrix is a N x N matrix where N is the number of target classes used for measuring the performance of a classification model where the output can be 2 or more classes. It is a table with 4 different combinations of predicted and actual values.



We run the following code :

>>confusionchart(test_labels,predicted_labels);



On this plot, the rows correspond to the predicted class that (Output class) and the columns correspond to the true class (Target class). The diagonal cells correspond to the observations that are correctly classified. The other cells correspond to the misclassified observations, where for example, in the first row we can observe that the model correctly predicted 11 Antelopes but misinterpreted one and predicted it as a Deer.

Conclusion

After training on our dataset many times, we have a model that can successfully classify the different animals with the accuracy of 91.48 that was built on ResNet-50 as its base infrastructure on the deep convolutional neural network application in MATLAB.