Butterfly Species Classification using Transfer Learning

A thorough understanding of Butterfly Identification and Classification.

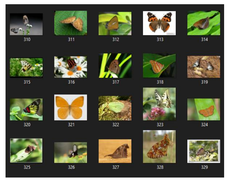
6597644, Faculty of Engineering and Physical Sciences, University Of Surrey

Introduction to the project

Purpose: The basis of the project is to create an artificial intelligence machine capable of analyzing pictures of different species and images of butterflies to determine to which of the 23 different species that certain butterfly belongs to.

Objective of the project:

- Train an Artificial Intelligence model using 10269 different images.
- Create a model able to recognize butterflies
- Make a model that classifies butterflies into one of 23 different species.
- Categorize over 15000 different butterfly testing pictures.
- Reach a model accuracy score of over 75%.



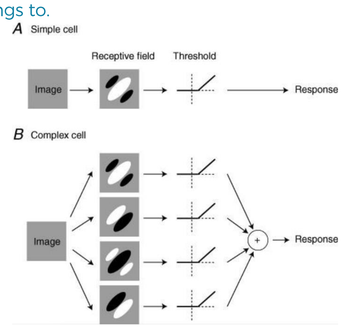


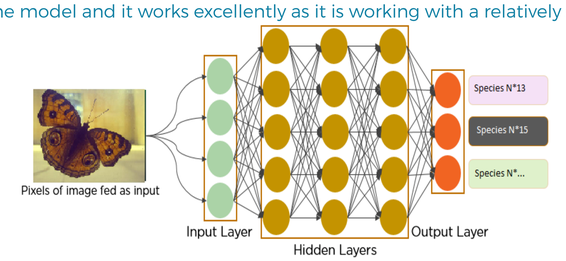
Diagram representing simple and complex cell discovery

Background information: In the year 1959, Torsten Wiesel and David Hubel came across simple and complex cells. In accordance with their study, when it comes to visual pattern recognition, we use two kinds of cells. A simple cell is able to recognize bars and edges of particular orientations at a particular part of the image, while a complex cell can respond to bars and edges of particular orientations as well. Not only that, but complex cells (unlike simple cells) are capable of responding to previously mentioned bars and edges at any position of the image. Complex cells are able to achieve this by sharing and adding data from various simple cells. Throughout the human body, it is possible to perceive both complex and simple cellular structures, which combined embody what is commonly known as the visual system. Inspired by these findings, in the year 1980, Dr. Kunihiko Fukushima designed an artificial neural network that imitates the way of how complex and simple cells work. Whilst C-cells operate like artificial complex cells, S-cells operate like artificial simple cells. These are called artificial since they are not naturally created neurons, but instead, they mimic the algorithmic structure of complex and simple cells. The main idea of Dr. Fukushima's Neocognitron was straightforward: Capture complex patterns utilizing complex cells that gather their data from other lower-level complex cells or simple cells that detect simpler patterns.

Analysis of the project:

For this project I have utilized what is known as a modified depthwise separable convolution by the name of Xception, which is a pointwise convolution accompanied by a depthwise convolution and a depthwise convolutional network as well. A Convolutional Neural Network(CNN) is a deep learning algorithm capable of taking in input images, be able to designate significance(via biases and learnable weights) to several identifiable properties of the image while also being able to tell the images apart from one another. Knowing this, this class of Deep Neural Network is excellent for image classification and recognition. It is worth mentioning I have used Transfer Learning in this project as well. Transfer Learning is a method used in machine learning where a model that is developed for a certain task, or in other words the model uses the output as an input for another model. Using this method we can greatly accelerate the training time of the model and it works excellently as it is working with a relatively small dataset.

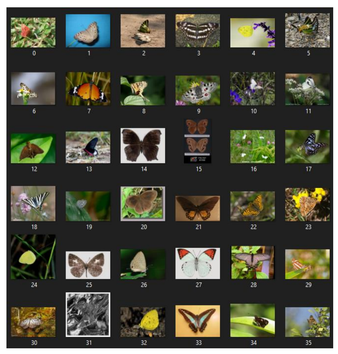
A CNN is based on the connectivity patterns in a Human brain in which individual neurons respond to stimuli in only a restricted area of vision. This means that the network can be trained to understand the sophistication of an image better given the analysis of a combination of these joint restricted areas. The essential role of a CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. A CNN is made up of 7 key layers which combine to create a fully functioning neural network, these layers include; Convolution layers, pooling layers, dense layers, softmax layers... Convolutional layers capture both low-level(colours, gradient orientation, edge detection....) and high-level(low-level features with less parameters.) . Pooling layers serve to reduce the spatial size of the convoluted features. This way the computational power that is required to process the data is reduced. Dense layers are the most commonly found layers in models, and they are deeply connected to the neural network, meaning each of the neurons in the dense layer receives an input from all the neurons of the previous layers.



Graph representing CNN in the project

Datasets to be used:

The datasets used in this project is a training dataset to teach the model and another dataset to test the effectiveness of the model. The training dataset contains 10269 different pictures of 23 butterfly species whereas the testing dataset contains 15009 images. Both datasets are provided by University Of Surrey and can be found on Kaggle under the name "UoS COM2028 coursework". We split the training dataset into a validation dataset that is 20% the size of the original training dataset, containing 2054 images. This new dataset will be used to evaluate the training dataset when the model is learning. These pictures have different shapes, sizes, colours, shading, orientation... This is the reason I have used ImageGenerator, to help the model deal with these inconsistencies and be trained in a more effective way. Once the main model is trained the AI will categorize the images in the testing dataset following a similar method of classification to that of the training dataset and write the results of the predictions in a csv file using the Id of the images and the labels guessed by the AI representing the category the model puts each butterfly in(23 different categories). It is worth mentioning that the images in the testing dataset are different and completely new to those of the training dataset and that There is roughly 500 more images to categorize in the testing dataset.



First 35 images of testing folder

References:

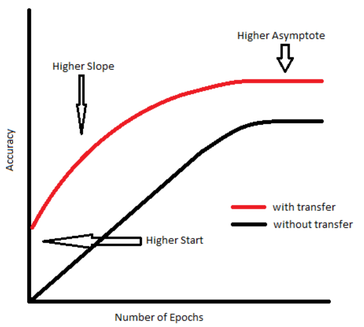
https://machinelearningmastery.com/transfer-learning-for-deep-learning/ introduction to transfer learning for deep learning
https://towardsdatascience.com/the-brief-history-of-convolutional-neural-networks-45afa1046f7f The brief history of Convolutional Neural Networks

Implementation of the methods:

Generate Training and Validation folders	Produce Image generators	Build the models	Compile and fit the models.	Write predictions
Created two for loop that each produce a folder, a training one and a validation one. This was accomplished by reading the train.txt file and creating subfolders where respective images with the proper label will go into. Validation folder is created with 20% of the size of the original training folder.	Created image data generators that greatly help the model. This method is useful since it transforms images and changes them so they become more like one another, by changing properties such as size, rotation, brightness, etc.	Two models have been built for this project. A base model and a secondary model. The base model is trained using already trained weights, and the output of the bottom layers are transferred into the second model. The next model takes the input from the base one, making it have a higher starting accuracy	Next starts the fitting of the models for them to be trained. Using 32 batch size the model fits fast. Callbacks have been used with a patience of 5, this means that if the loss value does not decrease in 5 epochs it will take the lowest loss value in 5 epochs. This helps maximize the accuracy value.	Lastly predictions are made using the last model fit. With the help of the model, individual predictions of test images are done, and those predictions are then written in csv file as the label of that specific image one by one, increasing with id. Once completed all predictions are written in the csv and prepared to be submitted.

Evaluation of the methods:

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. This is the main method used in the project. As it can be seen in the graph to the right, using transfer learning allows the model to start with a higher accuracy rate since the beginning, as well as a higher slope in the model and a higher final asymptote signifying how accurate the model gets. Using transfer learning leads to a significant Increase in the accuracies result in contrast to not using transfer learning at all. The result turns out to be rather satisfying. Although each model by itself gets decent accuracy(base model for example gets around 60%) the combination of both models results in scores of up to around 85%. Fitting the model is quite fast as well, with each epoch lasting about one and half minutes to finish, the fitting of the entire model takes about 45 minutes to complete. Using callbacks I assure that the accuracy value the model ends up with is the highest out of any epoch. This is the most effective method to getting the most accurate working model I could find, with a result of 86%. Although there is ways to further improve the model I am personally content with the resulting model and score, so I will make no further changes to the code.



Graph representing difference between using transfer learning and not using it.