Butterfly Species Classification Model using Transfer Learning

A Comprehensive Description detailing Butterfly Recognition and Identification

6586214, Computer Science Department, University of Surrey

Introduction to the project

Purpose:

This project's purpose is to identify and classify various images of butterflies into different classes, specifically 23 species. As seen in these images, the butterflies have some extreme differences, and so the model will need to be able to classify them accurately.

Background Information:

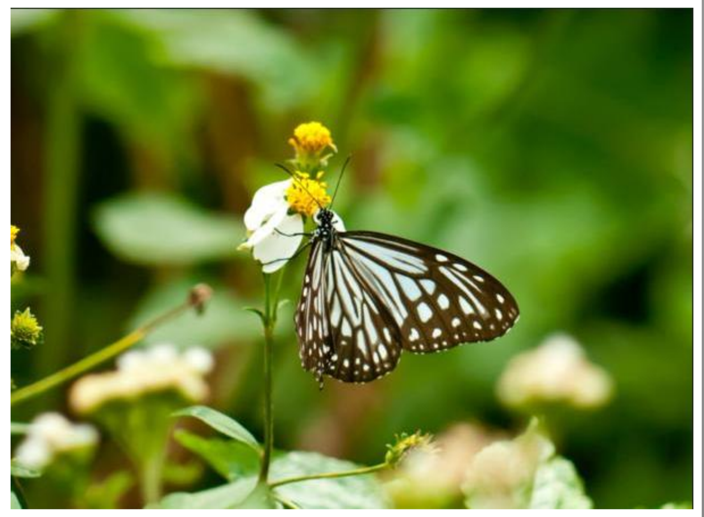
In 1976 Stevo Bozinovski and Ante Fulgosi published a paper explicitly addressing transfer learning in neural networks training. The paper gives a mathematical and geometrical model of transfer learning.

Transfer learning is a method in machine learning that essentially stores the knowledge gained from solving one problem and applying that knowledge to another similar one, in this case: butterflies.

Objectives:

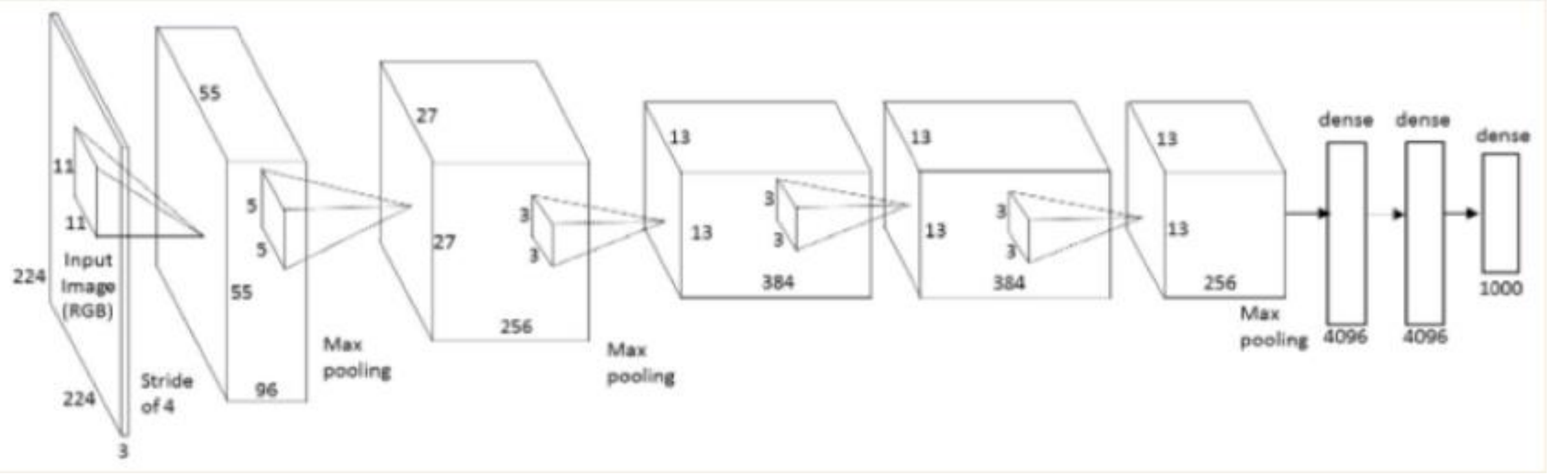
The model must be able to:

- Recognise an image of a butterfly
- Be trained successfully using 10270 images of butterflies
- Classify a butterfly into a specific class out of 23
- Do this with an accuracy above 80%, with 15009 test images.



Analysis of the problems

For the transfer learning model, I will use Xception, which is a Depthwise Convolutional Neutral Network (CNN). A CNN is a class of Deep Neural Network, which excels In object classification and image recognition so it would be optimal for this project. With transfer learning, it will speed up training time significantly and also it will work well with a small amount of training data, this is because it essentially uses an existing neural network that solves a similar issue and then reuse the lower layers of that network. Using this is called feature extraction, where the model uses the output as the input data for a new model [1]. CNNs developed from the study of the brain's visual cortex, and have been used in image recognition for around 40 years, recently they are used to power image search services, self-driving cars and more [2].



CNN Diagram

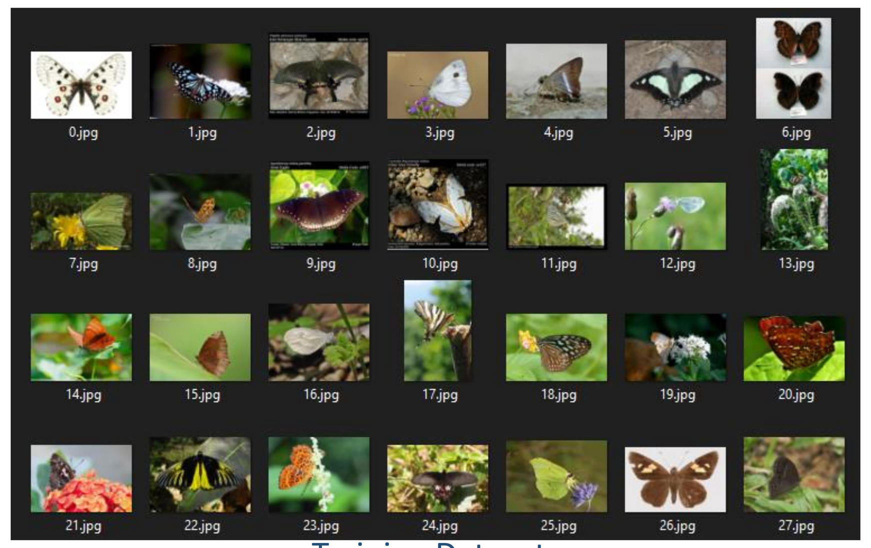
The CNN is made up of many layers in between the input and output. These can be convolutional layers, max pooling layers, dense layers, etc.

The 2D convolution layer creates a convolution kernel that detects low-level characteristics such as colours, This is then convolved with the input layer to produce a tensor of outputs. The dense layer is the typical deep Neural Network layer.

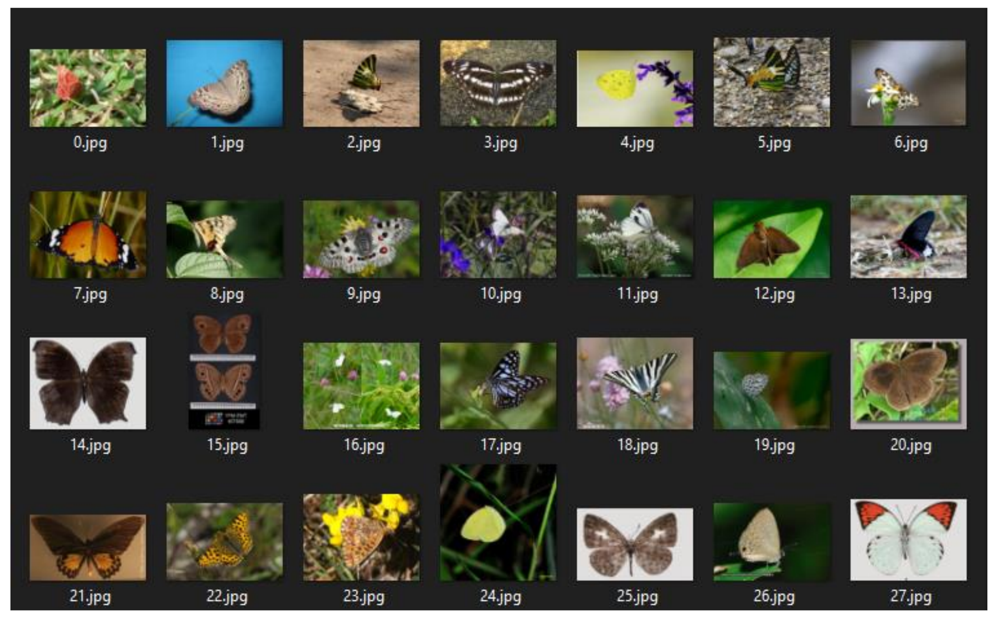
Pooling layers down sample feature maps by progressively reducing the spatial size to reduce computation in the network, allowing for the most useful data.

Datasets that will be used

Training Dataset: This dataset will be used to train the model. It will be split 80% for training, and 20% for validation. This is to have an accuracy for the model and a validation accuracy. This dataset is full of 10270 different images of butterflies from 23 different species. The dataset is provided from Kaggle's UoS COM2028 Coursework competition for this project. This image shows the first 28 images in the dataset, all with different names, these numbers are their IDs, and folders will be created from 0-22.



Training Dataset



Test Dataset

Test Dataset: This dataset will be used to test the model, it is a completely different batch of 15009 images from the training dataset. The dataset is provided from Kaggle's UoS COM2028 Coursework competition for this project. This image also shows the first 28 images in the test dataset, it will follow the same method of classification as the training dataset, however it is a much bigger dataset, and it contains completely different butterfly images, so it is an unbiased result.

These images are of all different sizes in terms of pixels, and some of them aren't oriented correctly, so an Image Generator could be used to correct these properties to assist the models.

Implementation

Creation Of Training and Validation Folders →

- Read Traint.txt into a Pandas Dataframe, create a split of 20% from that dataframe (validation folder size) and put images from the Train dataset into their corresponding folders based from Train.txt (Dataframe).
- Now exactly 80% of the images from the Train dataset should be in a "Training" folder, and 20% in a "Validation" folder, all sorted into their proper folders, e.g. "0", "22", etc.

Create Image Generators →

- Make Image Data Generators to "generalise" images, some images have different sizes, or a horizontally flipped differently from each other, this ensures they have the same properties, such as brightness, size, etc.

Make Base Model →

- The base model will be using the Xception Keras architecture, which uses pre-trained weights. It will be frozen and then compiled and trained.
- The bottom layers will be "transferred" to the next model for higher accuracy.

Build The Next Model →

- The new model will take layers from the base model, so it will have a higher starting accuracy, it will also have significantly lower learning rate.
- Before compiling and fitting this model, the base model will be unfrozen.

Model.fit →

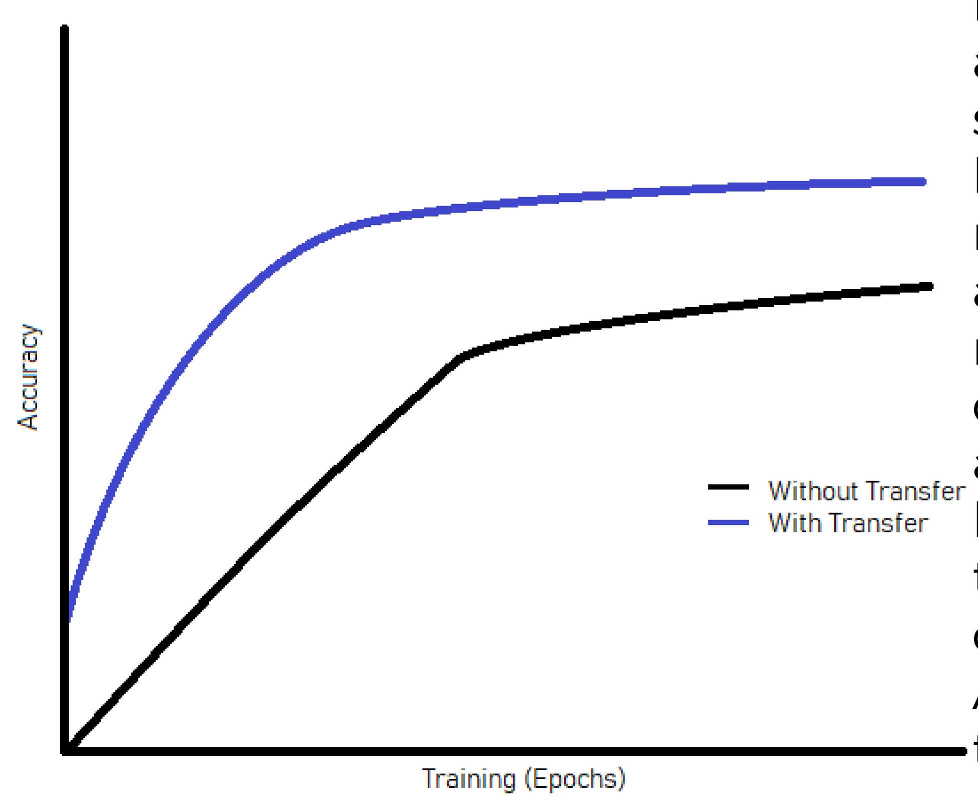
- The training data will be processed by the Image Data Generator, with fairly small dimensions and 50 epochs.
- 32 batch size allows for a good amount of steps per epoch, a callback will be set to stop the model based on the val_loss.

Model.predict

- The predictions will be made using the new model. The test dataset will also be passed through the Image Data Generator, and the steps will be the length of the test dataset divided by the batch size (32). Then this will be passed into a csv file.

Evaluation

Transfer learning involves the usage of another model's layers to get higher accuracy and better results in similar tasks the original model solves. In this case, butterflies.



By doing this, the model starts out with a higher accuracy, as seen in the graph. There is also a higher slope compared to a model without transfer learning, and it overall got a significantly higher result. The base model ends at around 60% accuracy, however when using it's layers with the new model, it ends at around 90% accuracy, this is due to feature extraction, small optimisations such as a high learning rate (0.1) for the base model, and a low learning rate (1e-5) for the new model also help the model greatly. Also, using epsilon in the optimiser helped achieve and even greater accuracy. A Global Average Pooling 2D layer was also used on top of the base model, on average it took around 2 minutes (121 seconds) for each epoch to process. This is a fairly large amount of time, but considering transfer learning increased my accuracy from a model without, by around 30%, it significantly helped creating a good model. There are many different optimisations such as fine-tuning that can be done to perhaps further improve this model as well.

References

https://keras.io/guides/transfer_learning/ [1] | Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems Paperback – Illustrated, 14 Oct. 2019 [2]