

# ISY5002 Continuous Assessment 2

Deep Learning for  
Classification of  
Bees, Butterflies and Moths

---

---

## Contents

Introduction and Problem Statement	3
Literature Review	3
Methodology and Results	5
Building dataset	5
Data Exploration	6
Data Preparation	6
Image Scaling and Aspect Ratio	6
Noise Removal	7
Structuring Image Set	7
RGB-normalisation	7
Data augmentation on training data	9
Deep Learning Modelling	9
Loading of image set	9
Baseline with popular deep learning architectures	9
Design of Custom Deep Learning models	10
Fine Tuning of Models	11
Model Selection and Performance	11
Challenges	13
Computational Limitations	13
Discussions and Conclusions	14
References	15
Appendix	16
Steps to reproduce Conda environment	16
Steps to run training and testing scripts	16

## Introduction and Problem Statement

Identifying insects correctly is critically important in entomological research, pest control, and insect-resource utilization. However, to identify an unknown species using traditional methods is very time consuming due to the vast number of insect species that must be identified and limited taxonomist resources. It is a problem that has been recognised for quite a while and there have been attempts for computer-based recognition systems to identify insect species automatically [1, 2].

In this project, we have chosen to focus on three insects that are vital for nature's pollination – Butterflies, bees and moths. This is a challenging problem due to the large intra-class variations (e.g., pose and appearance changes), as well as small inter-class variations (i.e., there are only subtle differences in the overall appearance between classes).

A sample of 1300 images of butterflies, moths and bees each, was split into training, test and validation sets. With this dataset, we employed deep convolutional neural networks (CNN) and Residual Network (ResNets) to extract the features of the 3 classes, and thereby reliably classify the species. We compared and evaluated several variations of the architecture with different depth of layers configurations for the CNN and ResNet frameworks and tuned the hyper-parameters to achieve the highest possible accuracy.

In addition, to set a baseline to compare against, we also used pre-trained models for popular image architectures such as VGG16, ResNet50 and Inception V3, left out the last fully connected layer, froze all existing layers and added our custom classification layer to our number of classes.

## Literature Review

There are over 175,000 species of moths and butterflies (Lepidoptera, “lep” for short) and around 20,000 species of bees worldwide. Hymenoptera is a large order of insects, comprising the sawflies, wasps, bees, and ants. Although some species are easy to recognize in photos, the majority can be quite challenging to identify. Some leps look nearly identical to sibling species, while others can be identified only by examining specific anatomical features. No single paper-bound field guide can include all known Lepidoptera, even when focusing on one geographic region.

Within this class of animals, both butterfly and moths are in the order Lepidoptera. The butterflies and moths are highlighted in blue as shown in Figure 1. [3]

Butterflies and bees both have two pairs of wings. Both are insects which feed on nectar from flowering plants. The main difference is that both butterflies and bees belong to the sub-phylum Insecta. They have six legs and they have an exoskeleton - their skeleton is on the outside of their bodies

Many day-flying moths also mimic bees for protection. These moths can share almost all of the superficial features of a bee: four wings, thickened antennae that can appear elbowed, and a furry, black-and-yellow coloration. The trick for seeing through these mimics is looking at the legs, and, if you can see it, the mouth. Moth legs are very slender, and clearly lack a pollen basket. The mouthparts of a moth consist of a long, slender tube (the proboscis), often equal to or greater than the length of the moth itself. This is held coiled under the head, but will elongate when the moth feeds.

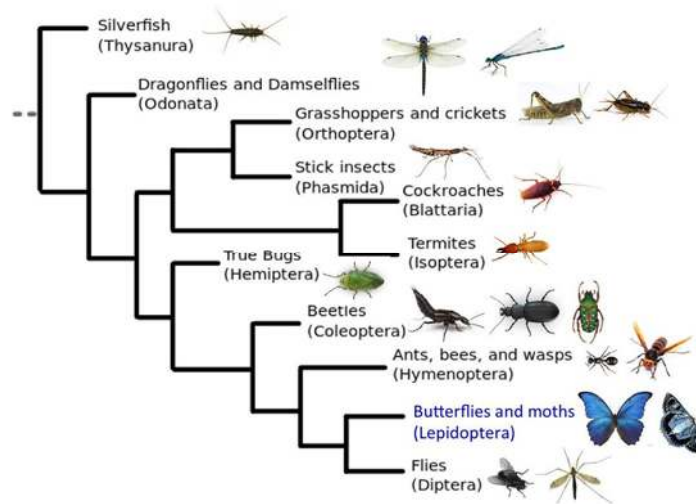




Figure 1 : Common insect orders (groups) within the overarching class of insects [Error! Reference source not found.]

Butterflies usually rest with their wings closed, while moths rest with their wings open. Butterflies have long, thin antenna, while moths have shorter feathery antennae. Butterflies generally gather food during the day while moths are seen more at nighttime.

One of the easiest ways to tell the difference between a butterfly and a moth is to look at the antennae. A butterfly's antennae are club-shaped with a long shaft and a bulb at the end. A moth's antennae are feathery or saw-edged. The difference between butterfly and a moth is as shown in Figure 2

Moth	Butterfly
	
Active at night (nocturnal)	Active during the day (diurnal)
Dull colors	Bright colors
Wings rest at their sides	Wings rest together and upright
Feathered or pointed antennae	Straight and clubbed antennae
Thick body	Thin body

\* There are some exceptions

Figure 2 : Butterfly-Moth Differentiation

## Methodology and Results

Our approach to addressing the problem at hand was comprised of five distinct components: Building dataset, Data Exploration, Data Preparation, Deep Learning Modelling, Model Selection. Each is described below:

### Building dataset

Initially, we constructed a dataset containing 1000 images each on butterflies, moths and bees. After initial training, results show that we may require more training data, hence, the final dataset contains 1300 images each for the butterfly, moth and bees. Most of the images were natural images from different diversified sources as follow:

- <https://leps.fieldguide.ai/figures>
- <https://unsplash.com/search/photos/>
- <https://pixabay.com/images>

We assembled the corresponding images via Bing Image Search API as well:

- <https://api.cognitive.microsoft.com/bing/v7.0/images/search>

The steps taken to build the datasets via Bing Image Search are as follows:

- Register for [Bing Image Search API](#), by clicking “Get API key”
- After finishing registration, note down the API key
- Create Python script to download images from Bing, making use of the API key earlier

During the pre-processing of the images, some corrupted images were encountered and were removed. Also, manual visual inspections of the images were required to ensure the quality and correctness of the respective insect images. correctness of the respective insect images as some images contain more than one objects which is not in the scope of this image recognition scope

## Data Exploration

The dataset consisted of 1300 images for bees, 1300 images for butterflies and 1300 images for moths. The images were of the following image formats – jpeg, jpg and png, with a dimension minimum size of 190 by 340 and maximum size of 6000 by 4002 and file minimum size of 6KB to a maximum of 10MB. In general, the images were clear and of good quality.

## Data Preparation

### Image Scaling and Aspect Ratio

In our deep neural network classification model, we would be assuming a square shape input image of a specific size. Therefore, the first thing to do was to crop and resize each image in our dataset to the same size, so that they could be fed into the deep net with the same input parameters (height, width). We did a centre crop as most of the images had the subject (butterfly, bee or moth) around the centre, and then did a resize to a square image (*See Figure 3*).

A higher resolution might potentially give higher accuracy but would mean more parameters and computational power. An excessively low resolution might need poor accuracy as there would be insufficient information in the images. We chose to resize to (224,224) as it is a popular size by multiple state-of-the-art models. and based on available computing power, it seemed likely that (224,224) would be our maximum image input size for our models. In model training, we might resize to a smaller input size if necessary.

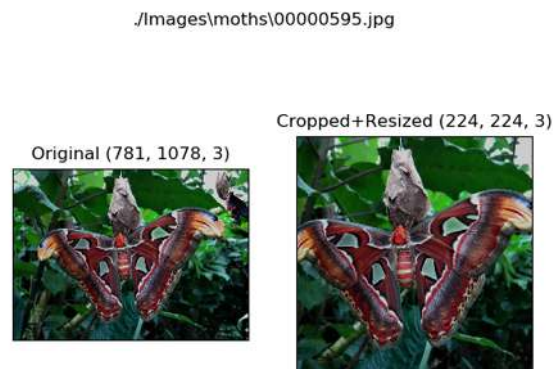


Figure 3. Example Original and Cropped+Resized Image

### Noise Removal

Since these were nature photos, many of which were professionally taken, photos were well-lit and had very little noise. Therefore, we decided not to do any noise removal.

### Structuring Image Set

After basic pre-processing was done, images were saved in a manner that would facilitate loading, training and evaluation of our deep learning models later. A script was created to structure our image set automatically in the following hierarchy, according to specified test-train-validation split (See Figure 4).

Further pre-processing would be done when loading the images for training using ImageDataGenerator with a custom preprocessing\_function. This allowed us to experiment with different secondary preprocessing methods easily.

```
Resized/
test/
  bees/
  butterflies/
  moths/
train/
  bees/
  butterflies/
  moths/
validation/
  bees/
  butterflies/
  moths/
```

Figure 4. Folder structure for image sets

### RGB-normalisation

Small weight values are preferred for neural networks, and inputs with large integer values can disrupt or slow down the learning process. As such it is good practice to normalize the pixel values so that each pixel value has a value between 0 and 1, in this case, by dividing all RGB values by 255.

We also wanted to ensure that each input parameter (RGB values) had a similar data distribution to make convergence faster when training the network later [3]. After normalisation, we calculated a per-channel means and standard deviation for the entire training dataset, and transformed the distribution of pixel values to be a standard Gaussian with per-channel mean of 0 and standard deviation of 1 (See

Table 1). This would help us to centre the data around zero mean for each channel (RGB) so that gradients act uniformly for each channel.

Table 1. Per-channel means and standard deviation for our training dataset

Channel	R	G	B
Mean	0.496	0.475	0.343
Standard Deviation	0.280	0.258	0.282

Using the same neural network architecture model, we did a comparison of pre-processing techniques on our training dataset for the following:

- 1) Normalising:  $x/255$
- 2) Normalising then standardising with per-channel means and standard deviation:  $(x/255 - \text{mean})/\text{std}$  for each channel

From Figure 5, we could see that standardising indeed gave better convergence for reduction of training loss and improvement of training accuracy. Therefore, we used pre-channel standardisation for all our models.

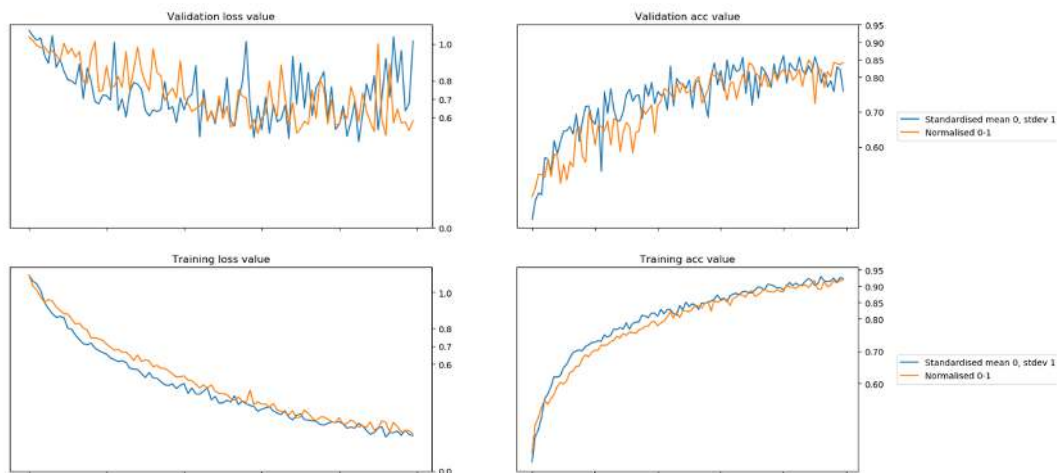


Figure 5. Comparison of normalisation vs standardisation



### Data augmentation on training data

Finally, in order to expose the neural network to a wider variety of object orientations and to increase sample size, we augment the existing data-set with perturbed versions of the existing images. This would make it less likely that the neural network recognises unwanted characteristics in the dataset [4].

In line with the practices of existing top performing deep learning models such as VGG, ResNet and Inception, we tried to vary the colour, brightness and perform horizontal flips only [5].

However, we found that varying the colour and brightness for our dataset gave worse accuracies. Since our classification problem was on insects, horizontal and vertical flips and rotations would be applicable. In addition, colour is very important in insect identification and should probably not be augmented too much or at all. So finally, we augmented the following only: Horizontal and vertical flips, rotation, and zoom.

### Deep Learning Modelling

#### Loading of image set

We chose to use the ImageDataGenerator class from Keras as images can be loaded for a single dataset in batches, meaning that we could easily scale up to load large image datasets [6].

#### Baseline with popular deep learning architectures

Before creating custom deep learning models, we wanted to have a baseline based on the existing top-performing deep learning architectures. Therefore, we used pre-trained models for VGG16, ResNet50 and Inception V3 from *tensorflow.keras.applications*. We left out the last fully connected layer, froze all existing layers and added our custom classification layer to our number of classes. The training and validation loss and accuracy are shown in *Figure 6*.

Overall accuracy on the test-set was 95.26%, 97.31% and 88.85% for VGG16, ResNet50 and InceptionV3 respectively (*See Table 2*). InceptionV3 performed the worst, probably because it was meant to be trained on default image size 299x299 while our resized images were only 224x224. As the objective was not to use transfer learning but to build our own models, we did not tune InceptionV3 or try with a 299x299 training set.

*Table 2. Overall accuracy on Test-data for popular deep learning architectures*

VGG16	ResNet50	InceptionV3
-------	----------	-------------

Best accuracy (on testing dataset): 95.26%				Best accuracy (on testing dataset): 97.31%				Best accuracy (on testing dataset): 88.85%			
	precision	recall	f1-score		precision	recall	f1-score		precision	recall	f1-score
bees	0.9807	0.9769	0.9788	bees	0.9923	0.9962	0.9942	bees	0.8712	0.9885	0.9261
butterflies	0.9068	0.9731	0.9388	butterflies	0.9580	0.9654	0.9617	butterflies	0.8655	0.9154	0.8897
moths	0.9752	0.9077	0.9402	moths	0.9689	0.9577	0.9632	moths	0.9429	0.7615	0.8426
avg / total	0.9542	0.9526	0.9526	avg / total	0.9731	0.9731	0.9731	avg / total	0.8932	0.8885	0.8861
[[254 4 2] [ 3 253 4] [ 2 22 236]]				[[259 1 0] [ 1 251 8] [ 1 10 249]]				[[257 3 0] [ 10 238 12] [ 28 34 198]]			

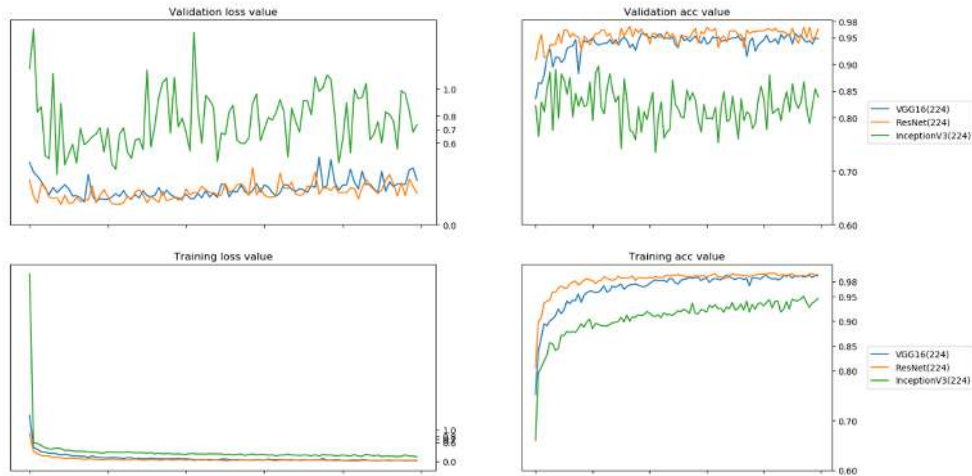


Figure 6. Comparison of popular pretrained models

## Design of Custom Deep Learning models

We will use the commonly used neural network architectures CNN and ResNet. A CNN architecture is useful for our dataset and is a good starting point since it could predict objects and faces in images using industry benchmark datasets with accuracies of up to 95% [8].

Similarly, ResNet has had great success in ImageNet competitions and have been shown to achieve better accuracy than VGGNet and GoogleNet while being more computationally efficient [9]. Indeed in our baseline, ResNet gave the highest accuracy by just using the pretrained weights (See Table 2).

Each of us built our own custom deep learning models in parallel, with the following general methodology to attempt to train and test a model that could achieve the highest possible accuracy for our image set.

- Try an architecture with different number of layers and filter size, with input size of 64x64 or 112x112 (due to time and GPU limitations), and 50-100 epoch
- If training and validation accuracies low and loss high → Tune learning-rate → If does not improve, try new architecture
- If training accuracy remains low and loss remains high after more than 20 epochs → Abort and try a different architecture

- If training and validation accuracies at end of epochs were still increasing, and loss is decreasing  
□ increase number of epochs
- If training accuracy is high but validation accuracy is low □ Over-training □ Tune learning rate
- If training accuracy and validation accuracy is good □ fine-tune hyper-parameters

Due to time and resources restrictions, the number of architectures and models that we could attempt were limited. Finally, only 7 models were shortlisted for further fine tuning.

### Fine Tuning of Models

For shortlisted models, the following processes were taken to fine-tune the model:

- Increased input image resolution to 224x224
- Decreased batch size as it usually leads to a better accuracy, limited by our GPU. Batch size of 32 was the minimum that we could use.
- For models that had very high training accuracy but much lower test accuracy, Dropout, L2 regularisation, and Batch normalization were added to improve generalisation and prevent overfitting.
- Learning rate (step size) was also varied to try to increase accuracy.
- Epoch size was increased from 100 to 150 for some models as loss did not seem to have plateaued.

### Model Selection and Performance

Table 3 shows the performance of the models based on training, validation and test accuracy for the final models after fine-tuning. Model CNN1W gave the highest test accuracy of 91.67% and is our chosen model. This model used a variable learning rate, 150 epochs and has a total of around 26 million parameters. The confusion matrix is given below in Figure 7. Comparisons of the models were also done and are shown in Figure 8 and Figure 9.

It was noted that the validation and training accuracy for CNN1E, CNN2E and CNN3E were on a clear upward trend even at epoch 100, and this means that accuracy could have been improved further with more epochs. Unfortunately, due to time constraints, this was not attempted (See Figure 9).

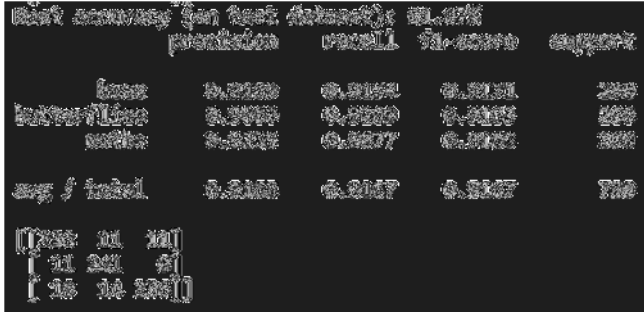


Figure 7. Confusion matrix for CNN1W

					Accuracy		
index	Model	training epochs	learning rate	no. params	Train	Validation	Test
	Baseline Pretrained Model						
90	VGG16	100	fixed	40,408,899	99.00%	95.67%	95.26%
91	ResNet50	100	fixed	126,352,259	99.20%	96.96%	97.31%
92	Inception V3	100	fixed	74,235,683	88.66%	89.42%	88.85%
	Model using size of 224x224						
0	CNN1E	100	fixed	6,719,043	89.54%	83.01%	83.72%
1	CNN2E	100	fixed	2,930,935	85.82%	83.65%	83.33%
2	CNN3E	100	fixed	1,210,051	86.18%	84.46%	83.97%
11	CNN1W	150	variable	26,295,683	99.24%	89.74%	91.67%
12	CNN2W	150	variable	104,952,963	98.68%	89.58%	87.95%
15	ResNet1	150	variable	2,787,171	81.25%	80.45%	79.87%
16	ResNet2	150	variable	2,301,187	85.82%	83.97%	83.33%

Table 3. Summary of best performing models

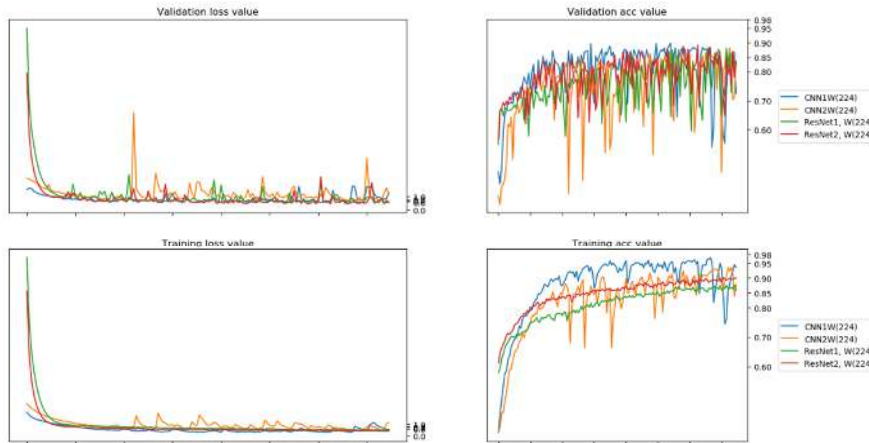


Figure 8. Best performing models for Wilson (150 epochs)

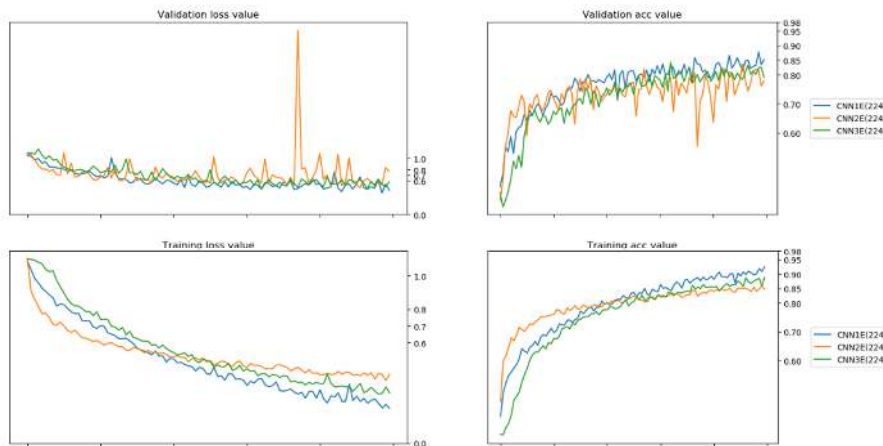


Figure 9. Best performing models for Edmund, Meiying (100 epochs)

## Challenges

### Computational Limitations

Training the deep neural networks required a good GPU card. Even with a good GPU card, each epoch duration ranged from around a few seconds to a couple of minutes, depending on the complexity of the architecture. This severely constrained the number of different architectures that we could try, as well as in tuning the hyper-parameters, example, batch size is limited to 32 for the configuration we were using. Any size larger than 32, would cause the GPU to trigger out of resource error and be unable to train the model.

## Discussions and Conclusions

In this project, we have used different depth and image resolution, as well as different architectures for the Butterfly, Moth and Bees classifications. Our highest accuracy model (91.67%) was less than the accuracy of ResNet50 (97.31%). This was likely due to the better training set and layers for ResNet50 – the numerous classes helped the neural network model to extract better features, coupled with the numerous layers, enabled it to achieve better accuracy. Due to our limited computing resources and time, we could not attempt a ResNet architecture that was as deep as ResNet50.

According to the experimental results, we could draw a few conclusions:

- Increasing image resolution size improved accuracy significantly but was limited by available GPU and time.
- Increasing number of epochs was very helpful in increasing accuracy as even at 100 epoch, accuracy was still on an upward trend.
- Batch normalization, Dropout and L2 regularisation, in general, helped to enhance test accuracy performance and generalisation of both CNN and ResNets
- It is not about the number of trainable parameters. A model with much fewer parameters can overperform one with much more.
- If given more time and resources, parameters and accuracy on our models could have been optimised further.

## References

- [1] K. J. Gaston, "Taxonomy of taxonomists," *Nature* 356, 281–282, 1992.
- [2] K. J. G. a. M. A. O'Neill, "Automated Species Identification: Why Not?," *Philosophical Transactions: Biological Sciences*, vol. 359, no. 1444, pp. 655-667, 2004.
- [3] H. Broadley, "Who's who? The elusive difference between butterflies and moths," *That's Life [Science]*, 20 11 2017. [Online]. Available: <http://thatslifesci.com/2017-11-20-Who%E2%80%99s-who-The-elusive-difference-between-butterflies-and-moths-HBroadley/>. [Accessed 29 9 2019].
- [4] J. Brownlee, "How to Manually Scale Image Pixel Data for Deep Learning," 25 March 2019. [Online]. Available: <https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/>.
- [5] N. B, "Image Data Pre-Processing for Neural Networks," Medium, 11 Sept 2017. [Online]. Available: <https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258>.
- [6] J. Brownlee, "Best Practices for Preparing and Augmenting Image Data for CNNs," Machine Learning Mastery, 3 May 2019. [Online]. Available: <https://machinelearningmastery.com/best-practices-for-preparing-and-augmenting-image-data-for-convolutional-neural-networks/>.
- [7] J. Brownlee, "How to Load Large Datasets From Directories for Deep Learning in Keras," Machine Learning Mastery, 10 Apr 2019. [Online]. Available: <https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/>.
- [8] "Neural Networks for Image Recognition: Methods, Best Practices, Applications," missinglink.ai, [Online]. Available: <https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/>.
- [9] KOUSTUBH, "ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks," CV-Tricks.com, 3 August 2018. [Online]. Available: <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>.

## Appendix

### Steps to reproduce Conda environment

#### Windows, linux with GPU:

```
conda create -n mew python=3.6 numpy=1.15.1 opencv=3.4.2 matplotlib=2.2.3 tensorflow=1.13.1
tensorflow-gpu=1.13.1 cudatoolkit=9.0 cudnn=7.1.4 scipy=1.1.0 scikit-learn=0.19.1 pillow=5.1.0
spyder=3.3.2 cython=0.29.2 pathlib=1.0.1 ipython=7.2.0 yaml pandas keras keras-gpu pydot
graphviz tqdm
```

(Assume Cuda 9.0 is installed, and cudnn 7.1.4 installed, get cudnn from <https://developer.nvidia.com/rdp/cudnn-archive>)

---

#### Windows, linux without GPU:

```
conda create -n mew python=3.6 numpy=1.15.1 opencv=3.4.2 matplotlib=2.2.3 tensorflow=1.13.1
scipy=1.1.0 scikit-learn=0.19.1 pillow=5.1.0 spyder=3.3.2 cython=0.29.2 pathlib=1.0.1
ipython=7.2.0 yaml pandas keras pydot graphviz tqdm
```

### Steps to run training and testing scripts

The dataset included have already been pre-processed and resized to 224x224.

Ensure all scripts are in the same folder.

To run the training script,

- Locate ISY5002\_CA2\_03\_ModelTrain.py
- Change the index and size to train, for in the main() function (L93-L94)

```
def main():
    # ----- CHANGE THESE -----
    index = 1
    size = 224
    # -----
```

- Execute ISY5002\_CA2\_03\_ModelTrain.py
- Model and epoch details would be saved into <ModelName>.hdf5 and <ModelName>.csv respectively, where <ModelName> = CA2\_<index>\_<size> eg CA2\_1\_224

To run the test script,

- Locate ISY5002\_CA2\_04\_ModelTest.py
- Ensure that ISY5002\_CA2\_02\_ModelDefinitions.py is also in the same folder
- Similarly, change the index and size of the model to test for, in the main() function (L68-L69)



- Execute ISY5002\_CA2\_04\_ModelTest.py
- Accuracy for training, validation and test datasets would be evaluated and printed in the console.

To run the model comparison visualisation,

- Locate ISY5002\_CA2\_05\_ModelComparison.py
- Ensure that all csv files that you wish to compare are in the same folder and that they follow the naming convention of <ModelName>.csv, where <ModelName> = CA2\_<index>\_<size> eg CA2\_1\_224.csv
- Execute ISY5002\_CA2\_05\_ModelComparison.py