

Master of Technology (IS)

Continuous Assessment 2

Project Report

Team Member:

CAO LIANG A0012884E

TAN CHIN GEE A0195296M

GENG LIANGYU A0195278M

Contents

[Executive Summary 3](#_Toc20678299)

[Business Problem Background 3](#_Toc20678300)

[Project Objectives & Success Measurement 3](#_Toc20678301)

[Project Solution Design 4](#_Toc20678302)

[Project Implementation 4](#_Toc20678303)

[Project Performance & Validation 23](#_Toc20678304)

[Project Conclusions: Findings & Recommendation 24](#_Toc20678305)

# Executive Summary

This project constructs deep-learning model to determine the public domain images classification on cat, bird and dog.

# Business Problem Background

The project requires the students to not use the pre-built image dataset from the public sources such as Kaggle, and the images should be downloaded from public domain without royalty and copyright. Also the number of images for each class should be at least 1000.

The project also requires the students to use only deep-learning model to resolve the classification problem.

# Project Objectives & Success Measurement

We decided to select 3 classification classes: cat, bird, and dog for the project. We proceeded to download the required public domain images and kept at least 1000 images for each classification class, and build the deep-learning model to train and validate the image classification model.

The measurement for the project will depend on the prediction accuracy on the test dataset, and it should show at least 10% or more improvement on classification accuracy over the initial unoptimized model to be successful, or it should reach at least 75% classification accuracy after tuning the model.

# Project Solution Design

The project should prepare the required image dataset, then filters out the outlier images, then construct the training dataset and test dataset, and then design and implement deep-learning model and perform hyper-parameter tuning, and finally perform the comparison and select the final classification model with the highest accuracy.

The project follows the supervised learning process as below.

Image Data Collection

Pre-Processing

Sampling

Training Dataset

Test Dataset

Deep-learing Model Training

Hyper-parameter Optimization

Post-Processing

Final Deep-Learning Model

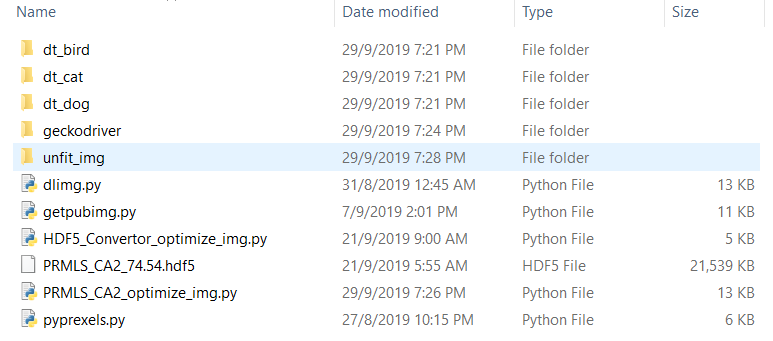
# Project Implementation

The project performs the steps below.

1. Image Data Collection

We have tried three different ways to download the required images, and have settled on using the third method after the first two methods proved unsuccessful.

All the files including applications and images files can be found at “pre-processing” folder.



* 1. <https://www.pexels.com/api/>

We first try to download from pexels.com by using an existing python PyPexels package to download images. But there are some major drawbacks in this approach. Firstly, there is a limit on the number of images that can be downloaded every day (<= 200). Secondly, there was not enough images of our selected classes i.e. cat, bird and dog in this website. Third and most seriously, the image quality for our selected classes was not good. Especially for bird and cat, many (>50%) of the images were irrelevant images (e.g., bird-eye view images of scenery).

We have developed the “pyprexels.py” application for testing and downloading images from the website, but we gave up the solution based on the reasons above.

* 1. <https://snappygoat.com>

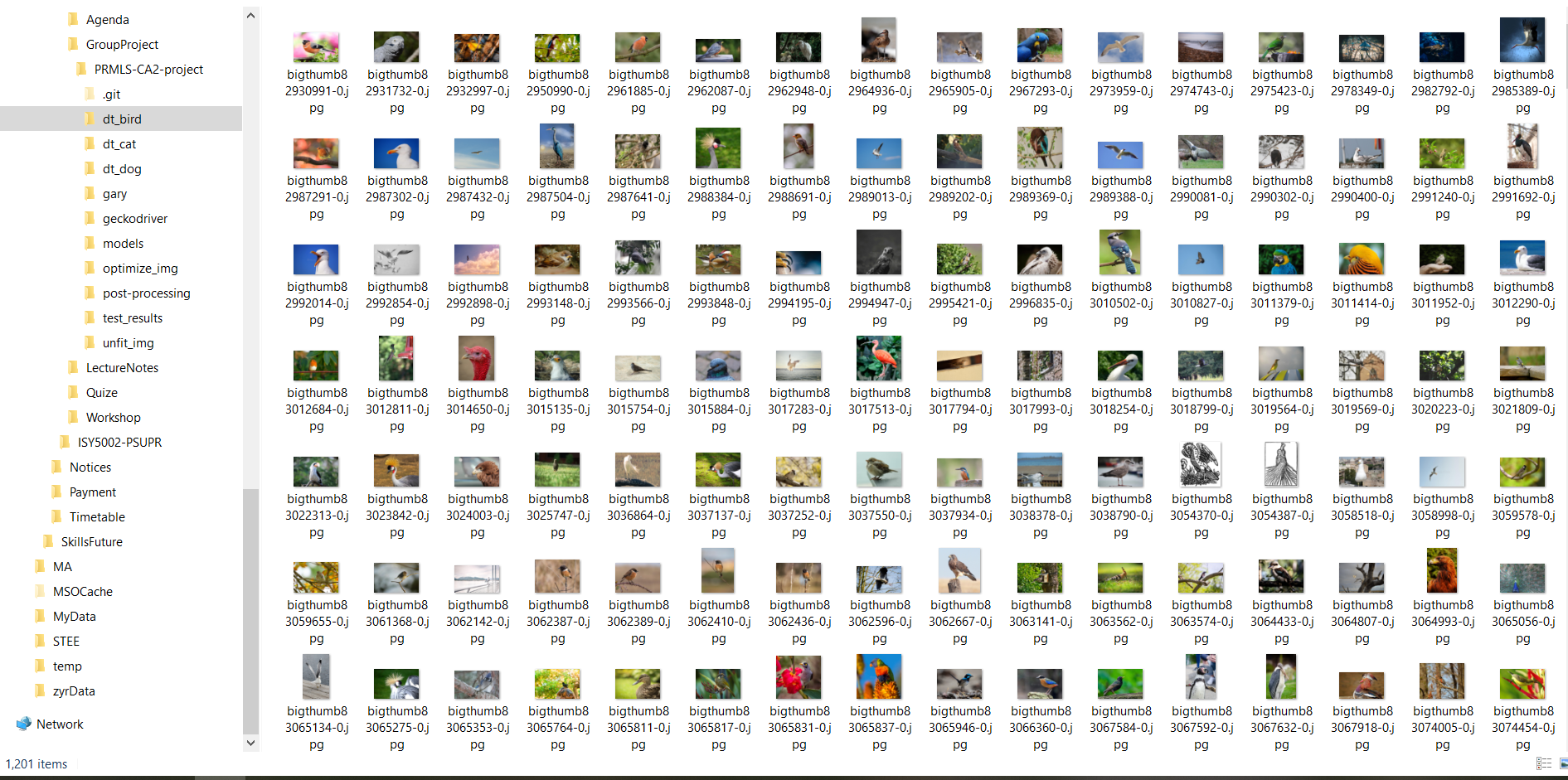
Next, we chose to try out the “snappygoat.com” web site and developed the “searchimg.py” application to download the images. We still encountered some issues in using this method. Firstly, the images for each class were not enough (<1000). Secondly, some of the images were irrelevant because people had given the wrong tag to their uploaded pictures and resulting in the wrong images being downloaded. Thirdly and most seriously, we have to pay the fee in order to use these images for our project. So we also give up this solution.

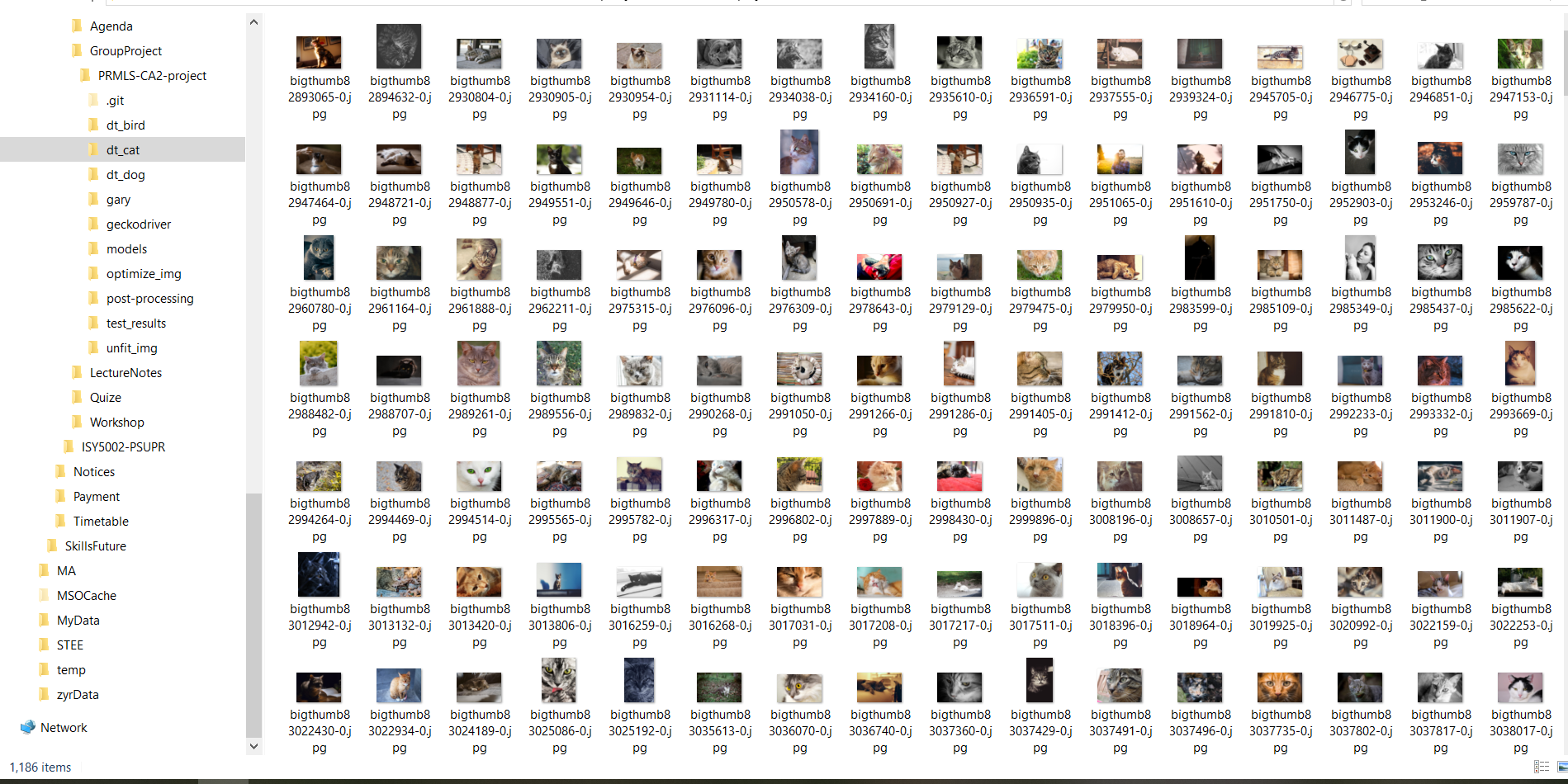
* 1. <https://www.dreamstime.com>

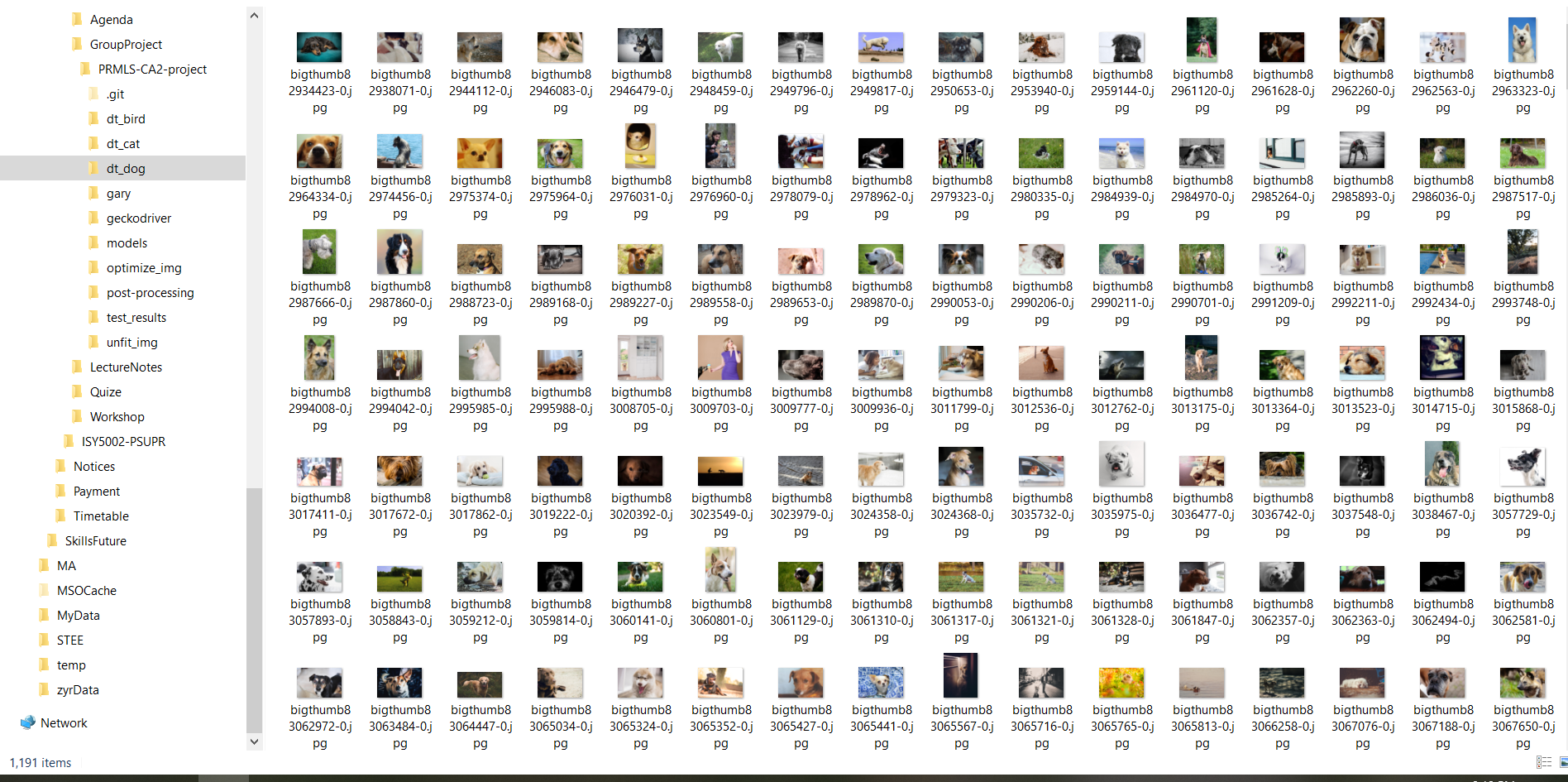
This is the solution we are using for the project. The site “dreamstime.com” not only provides enough images but also most of the images were good enough to be the candidate image data for the project. And we also learn that the images from “dreamstime.com” are truly royalty free for the project without any charge.

We developed the “getpubimg.py” application to download all the images for the project, and each class has 1200 images. Since the website html content is embedded in the script and cannot be parsed by html parser such as BeautifulSoup. We adopted selenium, a program that automates web browser, to implement the solution by simulating human action on the firefox browser. Also the website takes certain measures to prevent auto-download scrapper, so we have used a timer to wait for 15 seconds after simulating the event to click the next page link to bypass the restriction, and we created a separate thread to download the images based on the image source URLs.

The images downloaded were stored in three different folders (“dt\_cat”, “dt\_bird”, and “dt\_dog”) with names that represented their classes. The screen shots for “dt\_bird”, “dt\_cat”, “dt\_dog” folder files with their images are shown below.

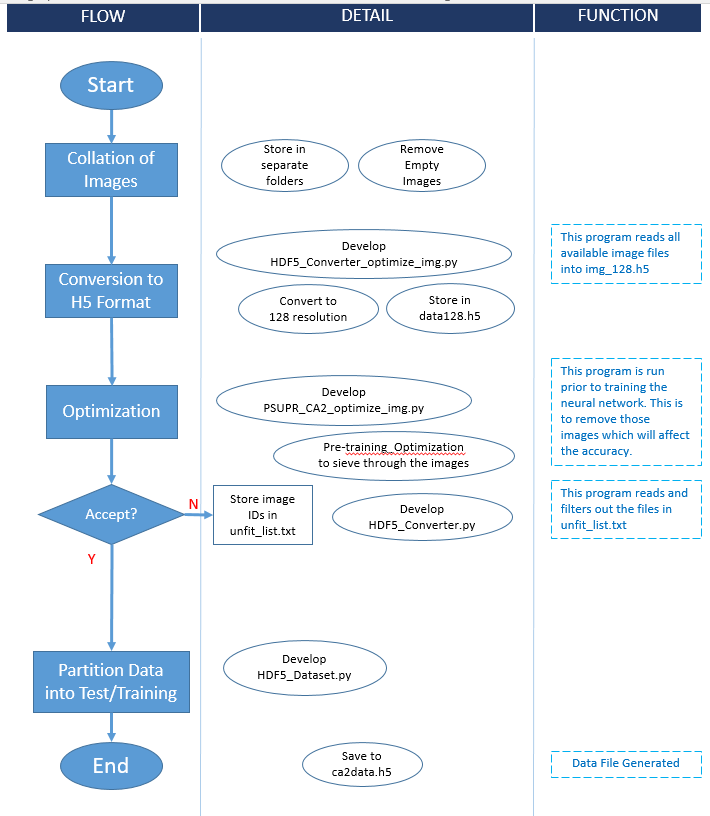






1. Pre-Processing

We have performed the following steps to prepare images dataset and the process flow is shown below.



2-1 Remove empty images

We noticed that some of the image files contains only blank images, so all these images are removed.

2-2 Save image data and class labels

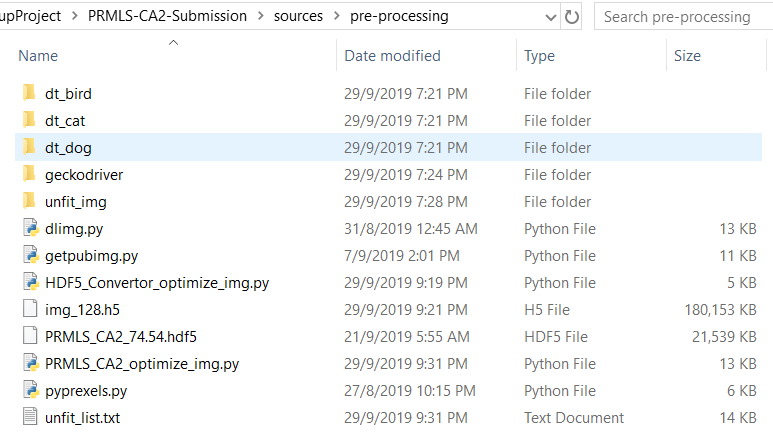
We developed the “HDF5\_Convertor.py” to read images file, converts image resolution to 128x128, and saves its content and class labels to “data128.h5” hdf5 format file.

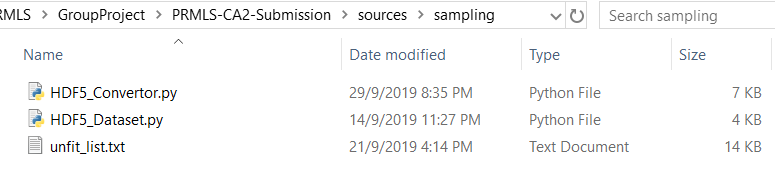
After several rounds of training and testing with the deep-learning base model, we noticed the accuracy is not much improved and the model was unstable with many spikes in the accuracy of the training model. So we decide to filter out the outlier images which have low predicted probability (< 0.1) for their classes.

We developed “HDF5\_Convertor\_optimize\_img.py” to read all available image files and store them into “img\_128.h5”, and ran “PSUPR\_CA2\_optimize\_img.py” to classify the images with pre-trained base model, and stored the list of identification number of all the outlier images to “unfit\_list.txt”. Specifically, the pre-trained model weight parameters are loaded from “PRMLS\_CA2\_74.54.hdf5” file, and all 469 images with below 0.1 prediction probability are recorded to “unfit\_list.txt”.

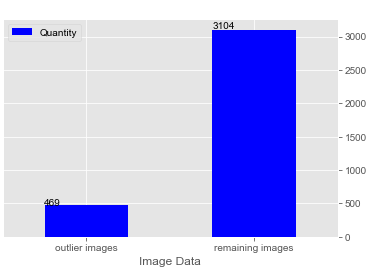
We then modify the “HDF5\_Convertor.py” to read and filter out the images in “unfit\_list.txt”, and re-generate the “ data128.h5” file.

The screen shots for these files are shown below.

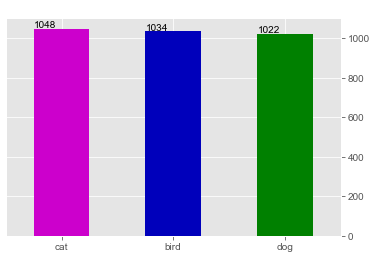




The figure below shows the number of outlier images (469) and remaining images (3104).



The reminding images still have enough samples (>=1000) for each class to meet the assignment requirements as shown below.



We also analysed the outlier images and learnt the reasons that made them hard to learn by the deep-learning model.

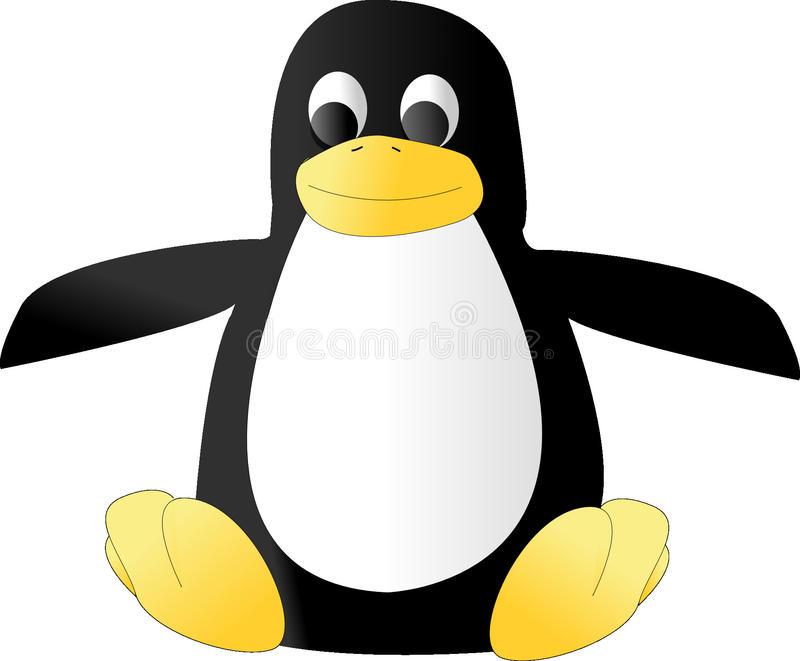
1. Targets blended into their environment because of their camouflaged coat colour



1. Targets dressed in human attire



1. Target images are either drawings or cartoons



1. Images are not of the targets



1. Targets are obscured by objects (e.g., tree branches)



1. Target images contained other objects (e.g., human)



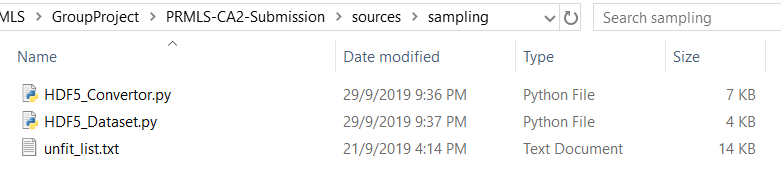
1. Targets are small inside their images

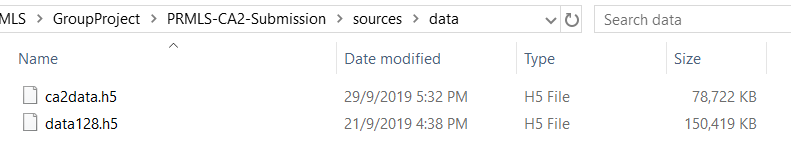


1. Sampling

We developed the “HDF5\_Dataset.py” to read image data and class labels from “data128.h5” file, and then divided them into training dataset and testing dataset, and saved to “ca2data.h5” hdf5 file.

The figure below shows these files.





1. Deep-Learning Model Training

We developed the “PRMLS\_CA2\_Base.py” and implemented deep-learning ResNet model as the base model.

The model is shown at “PRMLS\_CA2\_model\_final.pdf” below.



It contains total 1,886,659 parameters, and inside them there are 1,881,827 trainable parameters and 4,832 non-trainable parameters.

1. Hyper-parameter Optimization

The very first time after we developed the base deep-learning model with all known best practice parameters and trained the model, we achieved a testing accuracy of 73.36%.

Then we kept adjusting the hyper-parameter to improve the model accuracy as described in the following sections.

5-1 Select activation function

We chose Relu activation function because it is better at updating and preventing gradient vanish compared to other activation function such as Sigmoid.

5-2 Set weights initialization value

We chose He initialization to set the weight initialization values, because this will avoid assigning a value that is too small or too large.

5-3 Restrict weights

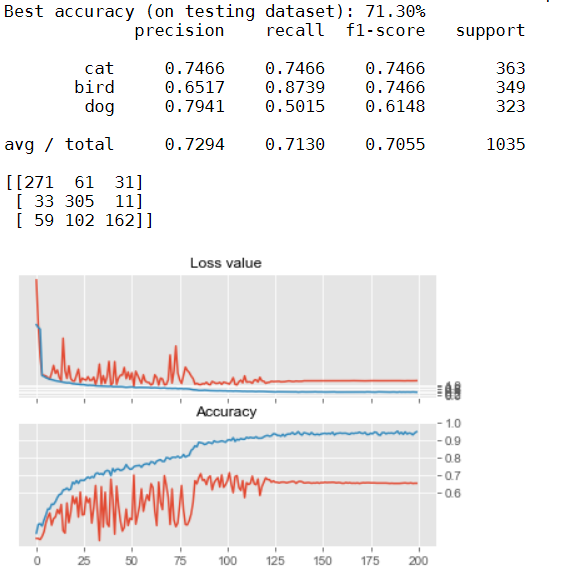
We selected L2 regularization to control the magnitude of the model weights by penalizing large weight values to keep so to keep the weights value small.

5-4 Add dropout

We have tested adding dropout between layers, and the accuracy results were improved. The following shows the performance before and after dropout layers are added.

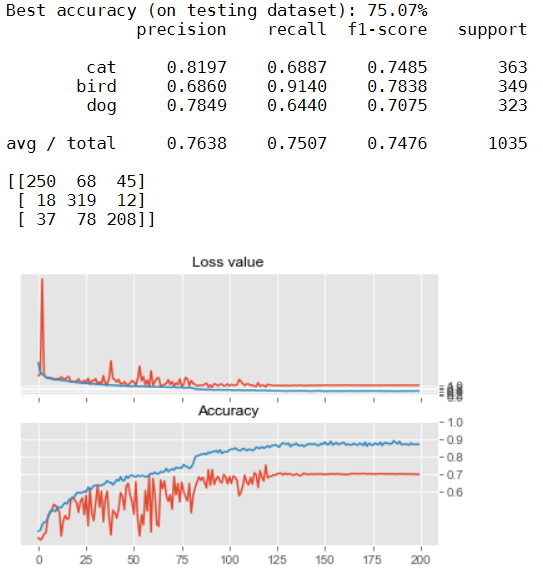
1. Without dropout layers

The accuracy without drop layers only reached 71.30%.



1. Add dropout layers

The accuracy reached 75.07% after adding dropout layers.



5-5 Perform batch normalization

We use the batch normalization to normalize the output for the convolutional layer.

5-6 Design residual network

We follow the ResNet design rules to design the residual network to improve the deep-learning network performance. We designed the residual layers by: (1) specifying the number of residual blocks from a range between 3 to 11, and (2) specifying the number of ResNet from between 3 to 5.

We kept the values of certain parameters constant in specifying the residual layers. Some of these parameters and their values are:

* Kernel size = 3
* Strides = 1
* Kernel initializer = he normal

The first stage of the residual layers does not contain a downsample block. But each of the subsequent stage comprises an initial downsample block with the reminding blocks being simple blocks.

We increased the number of filters from 16 for the initial ResNet to 32, 64, 128, and 256 for each of the subsequent ResNet.

The layers at the front which start from input layer will capture the lower details features, such as pixel and textures, but the layers at towards the end will capture the higher level features, such as lines, patterns, and figures.

The key to adjust the layers is to understand the model and know whether it lacks the lower details features or higher level features. We can always adjust the layers and observes the training loss values and accuracy values to learn the model. From our testing experiences, before adding the 5th ResNet layers which have 256 filters, the loss values and testing accuracy history show that the model is not stable, but after adding the 5th ResNet layers, the model becomes more stable, and also the accuracy rate is improved from maximum 77% to 80%. From these signs, we learn that the original model lacks the higher level details, so increase the ResNet layers at the end of the output is the right direction.

5-7 Set learning scheduler

We create and utilize the learning scheduler to vary the learning rate, so learning rate is reduced when the learning epochs increase.

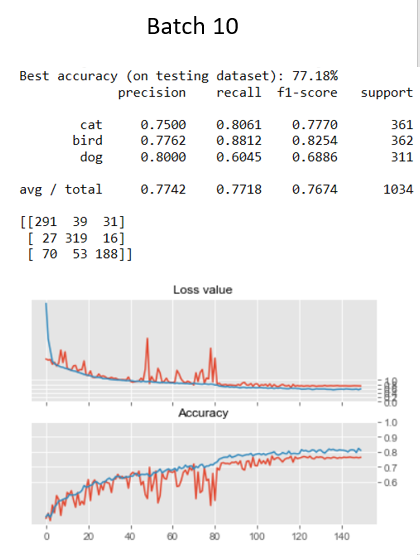
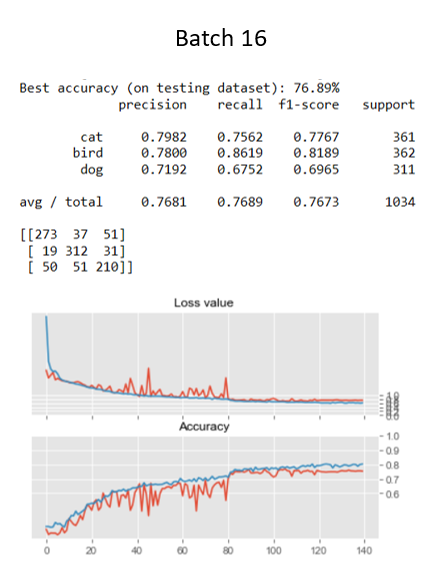
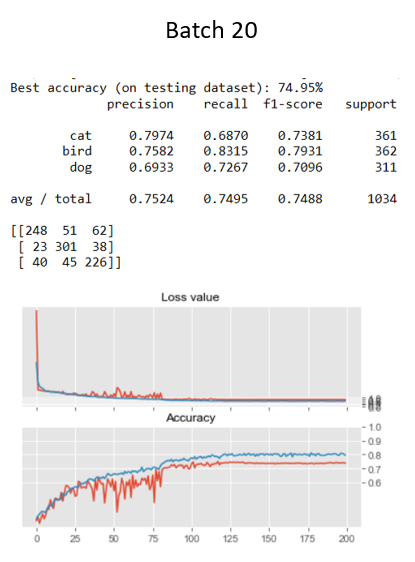
5-8 Use image augmentation

We use the image augmentation to generate the randomly varied images in the beginning of each epoch, so as to force the deep-learning model to learn the features that are relevant to the classification. The range of the image augmentation, e.g. the width\_shift\_range, height\_shift\_range and rotation\_range, have been adjusted to bigger value (> 0.1), but the accuracy is not observed to be improved. We also tested changing the range of brightness, but it lowered the test accuracy results so we did not include it in the final model.

5-9 Adjust batch size

We chose a small batch size (10) to improve the accuracy, and also it still allowed the laptop GPU used for training to work as normal. The batch size was reduced from 32 to 10.

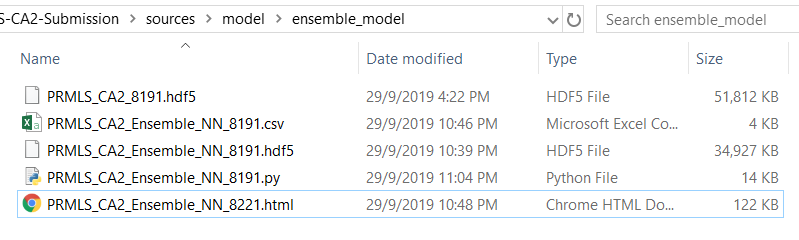
The batch size also impacts the learning and testing accuracy. The figures below show the accuracy is improved from 74.95% to 77.10% by decreasing the batch size from 20 to 10. So the final model batch size is 10 based on the testing above.



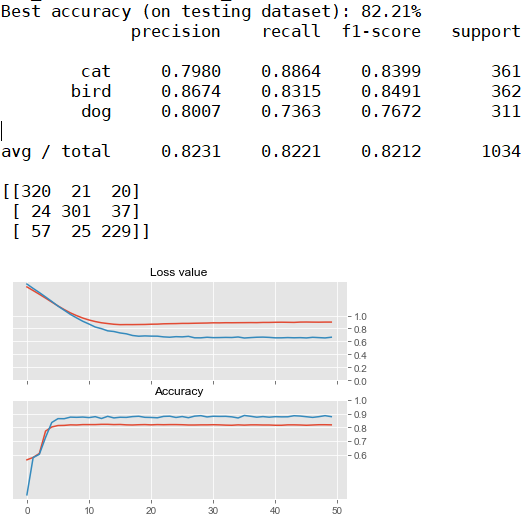
5-10 Ensemble models

We also tested the use of an ensemble model in classifying images. The ensemble model is constructed using the integrated stacking method, which combines the 2 pre-trained models output and constructs common dense layer to finally output the classification results.

The ensemble model application to run is “PRMLS\_CA2\_Ensemble\_NN\_8191.py” and the relevant files for ensemble models are shown below.



The accuracy of the ensemble model achieves 82.21% and shows improvement on the individual model with 81.91% accuracy.



It contains total 8,764,997 parameters, and inside them there are 63 trainable parameters and 8,764,934 non-trainable parameters.

We decide not to adopt the ensemble model as the final model since in practice the ensemble model will requires the far more resources to train and execute. But it is still a good knowledge to learn that the ensemble model will normally perform with the better accuracy than its member deep-learning model which construct the ensemble model.

5-11 Problems encountered and challenges

1. Different Model Training Result

We noticed that even for the same model design, the model accuracy after each training was very different, sometimes even reaches 4% to 5% differences in accuracy. So we have to save the individual model weights in separate files, so that we can replicate the same accuracy with the model design, and avoids losing the best model.

And we have lost some good models due to not properly saving the weight files of these models and thus cannot replicate the same accuracy result.

1. Determination on good parameters

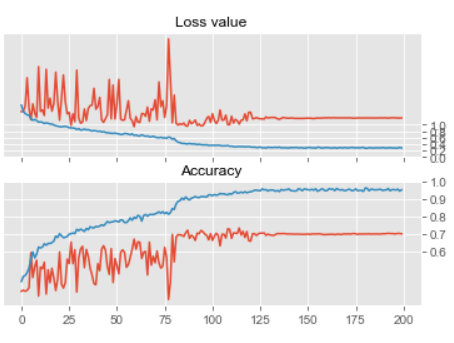
We observed that the model performance differed for each separate model training process. In order to determine the parameters to be changed, we only chose one parameter to change at each time, for example, learning rate. Then we performed several trainings.

We will consider the model best accuracy across several trainings, and analysed the loss value and accuracy training diagram, then decide whether to keep the changes to the model.

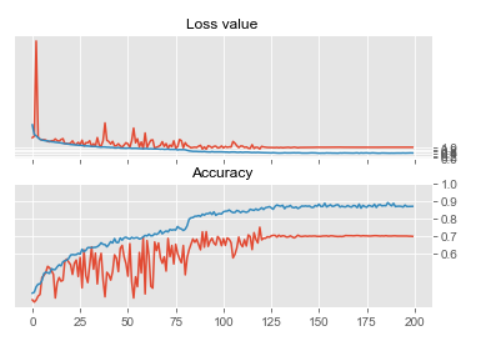
After that, we continued to pick the next available parameter and performed the same evaluation process again. But after several testing rounds, we found out that there are many factors that affect the accuracy of the model, and thus we should consider the combination of different parameters during model evaluation. Therefore, we learnt that we need to understand the relation among the parameters and adjust them together and adopted this as the better approach.

1. Unstable model

We noticed that the base model is unstable from training loss value and accuracy chart below.



So in order to fix the issue, we have tried many different ways, e.g. changed the learning rate, added drop-out layers, added more layers, and found out that adding drop-out layers and adding one ResNet layer helped to stabilize the model. The figure below shows the training result after adding drop-out layers and adding one ResNet layer.

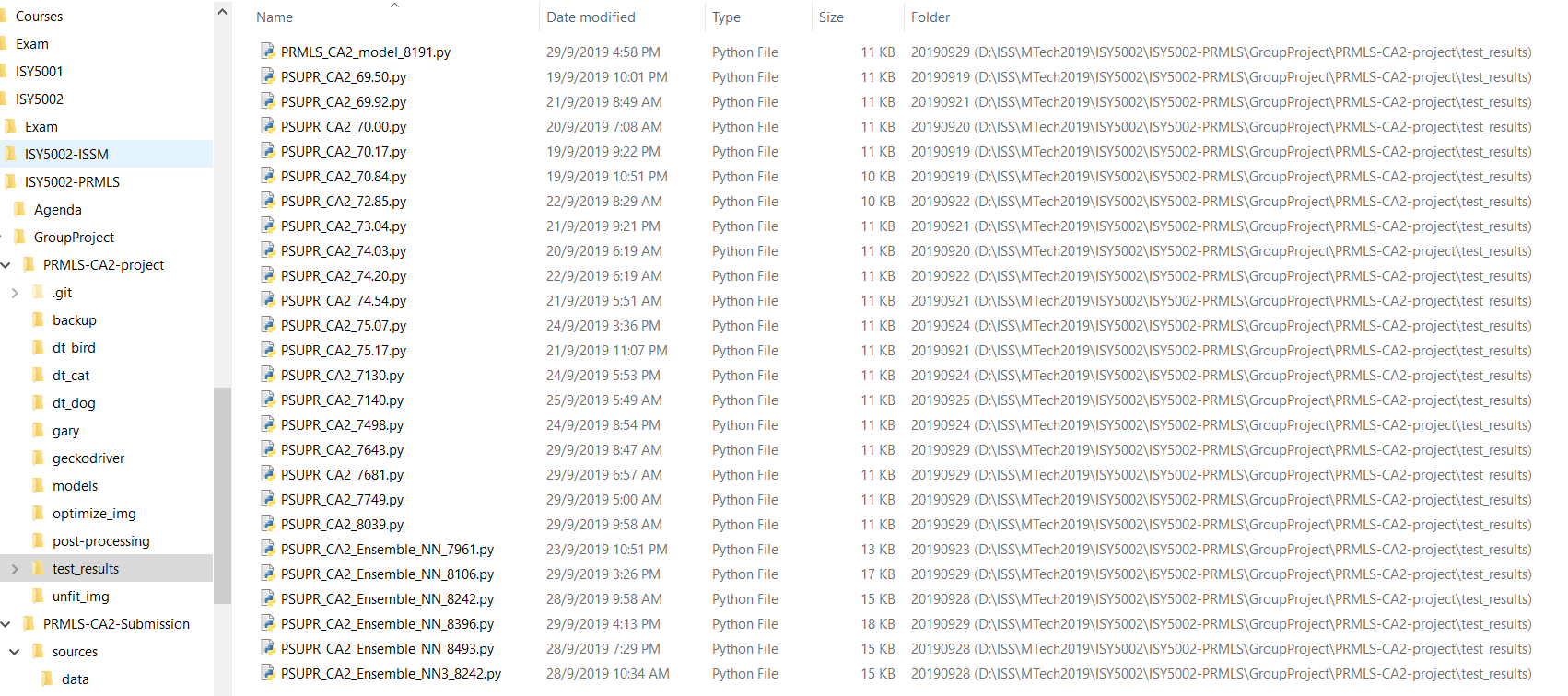


And we notice that the ensemble model is much more stable than the individual deep-learning model of its members.



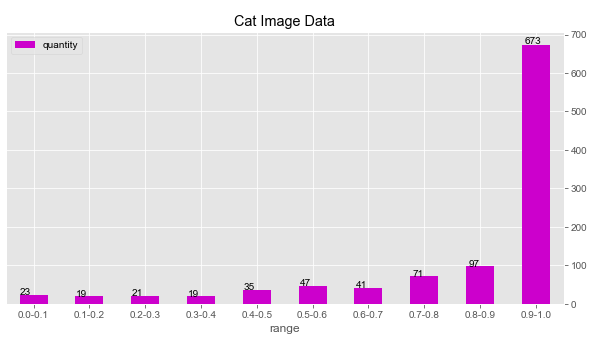
1. Challenge

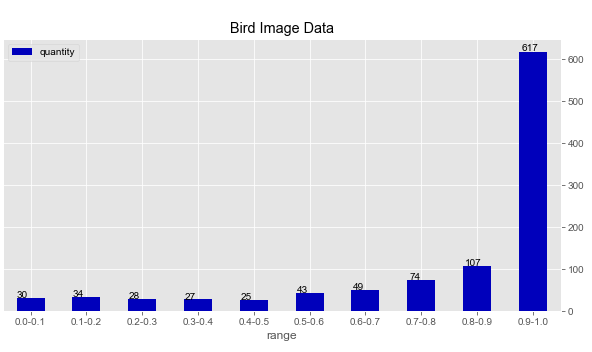
The biggest challenge we faced was the computer resource constraints. The normal training time for our deep-learning model took 2 hours to 4 hours. And we do not have enough time to improve the model. So the best model and best performance is the best one after we have performed many tests. Some of the python application files with different models which have been tested are listed below to show the tests we have performed.

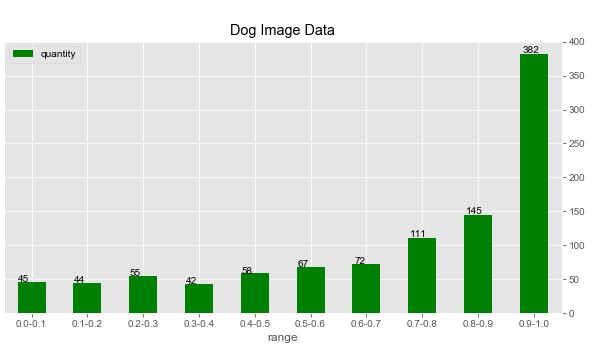


1. Post-Processing

After processing with best ensemble model, we develop “sources\post-processing\PSUPR\_CA2\_img\_analysis.py” to analyse the image classification results which are shown as below.







As the charts shown, there are still some low-quality images for cat (23), bird (30) and dog (45) which prediction probabilities are below 0.1. Therefore, it is possible to improve the classification accuracy by removing some of them.

1. Final Deep-Learning Model

The final deep-learning model is developed in the training script “PRMLS\_CA2\_training.py” and testing script “PRMLS\_CA2\_testing.py” which is the fine-tuned version based on the base model.

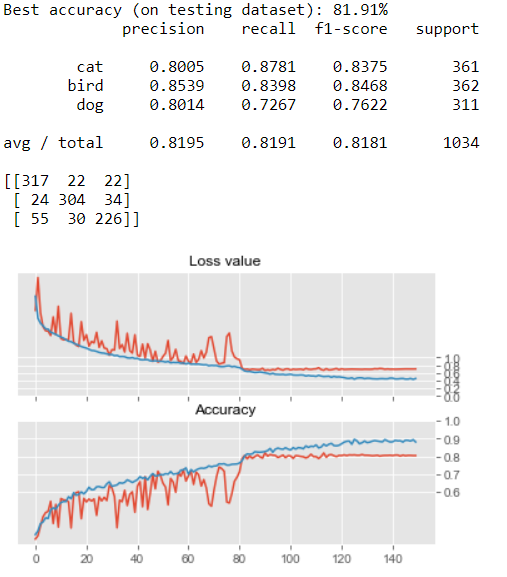
The model is shown at “PRMLS\_CA2\_training\_model.pdf” below.



It contains total 4,382,467 parameters, and inside them there are 4,376,483 trainable parameters and 5,984 non-trainable parameters.

# Project Performance & Validation

The project final deep-learning ensemble model archives 81.91% accuracy on the test dataset, so it meets the project performance requirement. The validation result is shown below.



# Project Conclusions: Findings & Recommendation

The project shows that the deep-learning ResNet model is able to archive good image classification results, but adjusting hyper-parameter to improve the classification results are not easy. The ensemble model is one of the recommended ways which achieves the better accuracy on the individual model. Also another recommended way to improve the performance is to adjust the hyper-parameters together and not just take them one by one when performing hyper-parameter tuning.

We also notice that the original dataset has vast impact on the final accuracy results. Without picking up the adequate dataset, the deep-learning model becomes unstable and hard to improve. So the effort spent on the dataset preparation is justifiable, even if it seems like a waste of time. And we need to understand the dataset before performing the model training.

The experiences from assignment also teach us how to justify the variance and bias regardless they are at dataset or model training. We have to justify the variance and bias from the training loss values and accuracy history, and adjust the parameters accordingly.