Implementation & Testing – Model Development Phase 4

## 4.1

Following a brief period of additional research into the *EfficientNet* pre-trained architectures, it became apparent that two particular members of the family had the greatest potential for successful implementation. Subsequently, the decision was made to draft an entirely new program, intended to optimally implement these two most promising architectures, *EfficientNetV2S* and *EfficientNetB0*. These pre-trained models are promising due to their combinations of size, speed and accuracy [3], and the hope is that this promise will translate into highly competitive performance results.

The [new code](https://github.com/yungroms/y3_proj/blob/89b4a412c3cbd6ffe5f6ce891884f0032992de72/4.1%20(6.2)/4.1.py) version created for phase 4 shares many similarities with its predecessors, including the general layout and order of operations. The library importation section differs slightly: *OS* has been replaced with *Pathlib* to handle input and output directories; *Seaborn* has been imported to improve data visualisation; and the desired layers and models have been imported specifically to allow for cleaner implementation.

A screen shot of a computer

Description automatically generated

Figure 43 Importation 4.1

Within the data preprocessing section, the *Keras* function image\_dataset\_from\_directory is used to handle the splitting of the dataset into training and validation sets. Unlike previous versions, codes 4.1 and onwards will utilise datasets which have already been separated into training/validation and test sets, meaning the code just needs to handle the splitting of the training/validation sets into respective training and validation sets. This has been done to ensure that all tests utilise the same set, meaning models are tested using the same images.

A screen shot of a computer program

Description automatically generated

Figure 44 Dataset Split 4.1

The next section contains the specification of the hyperparameter values, as well as the start of the model definition, which begins will the instantiation of the pre-trained *EfficientNet* architecture.

A computer screen shot of a program code

Description automatically generated

Figure 45 Model Definition 4.1

After the pre-trained architecture is included within the definition of the ‘full’ model, additional layers are then added to serve as final layers which flatten and transform the pre-trained model’s values into the desired output format.

A black screen with white text

Description automatically generated

Figure 46 Output Layers 4.1

The model is then compiled, with the specification of important parameters such as the optimisation method, learning rate and loss function.

A black screen with white text

Description automatically generated

Figure 47 Model Compilation 4.1

Next comes a new inclusion, which aims to improve the data recoding process of the program: the *CSVLoggerCallback*. The purpose of this useful feature is to automatically document the training and validation results after each epoch, in a cleaner and simpler fashion than the implementations in previous versions.

A screen shot of a computer screen

Description automatically generated

Figure 48 Logger Callback 4.1

Model training is implemented next, including in its parameters the training and validation datasets, the number of epochs, and the newly-created logging callback. Following this, the model saving process is defined.

A screen shot of a computer program

Description automatically generated

Figure 49 Model Training 4.1

The final sections of code 4.1 are also similar but improved in comparison to previous versions, containing the data visualisation process for plotting the accuracy and loss line graphs, followed by the definition of a new function for the testing process called *evaluate\_model*. This function incorporates the evaluation of the model using the pre-established test dataset, as well as the creation of the test results’ confusion matrix.

A computer screen shot of text

Description automatically generated

Figure 50 Evaluate Model 4.1

For test run 4.1, the learning rate will be 0.001, the batch size will be 32, and the number of epochs will equal 10. Additionally, the chosen dataset will be a new version of the mushroom dataset named [*shrooms\_ds\_split*](https://www.kaggle.com/datasets/yungroms/shrooms-ds-split), updated to be pre-split into testing and training/validation sets.

## 4.1 Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 1.1358641386032104 | 0.6073740124702454 | 0.8104385733604431 | 0.7142857313156128 |
| 2 | 0.7446956634521484 | 0.7466169595718384 | 0.6739667654037476 | 0.7778649926185608 |
| 3 | 0.6397733688354492 | 0.7766228914260864 | 0.6440829038619995 | 0.779434859752655 |
| 4 | 0.5722954273223877 | 0.7983918190002441 | 0.5954170227050781 | 0.7849293351173401 |
| 5 | 0.5141079425811768 | 0.8254559636116028 | 0.6367316246032715 | 0.7786499261856079 |
| 6 | 0.4740147888660431 | 0.8336929082870483 | 0.6287651062011719 | 0.784144401550293 |
| 7 | 0.45026329159736633 | 0.8450676798820496 | 0.5533455014228821 | 0.8006279468536377 |
| 8 | 0.4089750051498413 | 0.8558540940284729 | 0.5666468739509583 | 0.802982747554779 |
| 9 | 0.38659313321113586 | 0.8636987805366516 | 0.5268163681030273 | 0.8218210339546204 |
| 10 | 0.3500184118747711 | 0.8758580088615417 | 0.5276644825935364 | 0.8163265585899353 |

As can be seen in the training and validation results table above, test run 4.1 has produced very promising results. Using the *EfficientNetV2S* pre-trained architecture, the model created in 4.1 has achieved the best scores of any model trained and tested on a mushroom dataset. The previous top results were recorded during stage 3.3, which achieved accuracies of 0.72, 0.78 and 0.76 for training, validation and testing respectively, alongside loss scores of 0.83, 0.66 and 0.68. In comparison, test run 4.1 recorded accuracies of 0.87, 0.82 and 0.82 and losses of 0.35, 0.52 and 0.57. These are significant improvements which must be attributed to the *EfficientNetV2S* architecture as well as the improved code.

|  |  |
| --- | --- |
| loss | 0.5741644501686096 |
| accuracy | 0.820588231086731 |

Additionally, the model created during 3.3 was trained on *shrooms\_ds\_max*, which is substantially larger than the dataset used to train the model of 4.1. This is an indication of the utility of EfficientNetV2S within this project, providing promise that utilising the larger dataset to train and test a model with this configuration could produce even better results.

A graph of different colored lines

Description automatically generated with medium confidence

Figure 51 Accuracy & Loss 4.1

Although the results of 4.1 are promising, room for improvement can be seen in the visualisations of its results. Firstly, in the accuracy and line graphs above, the separation between the training and validation lines is suggestive of overfitting, whereby the model is learning its training data too specifically, and may be comparatively underachieving on the validation and test sets. Also, the intersection of the training and validation lines occurs rather early on in both graphs. This suggests that the learning rate might be too high, and that potential improvement could be achieved by lowering it slightly.

A blue squares with white text

AI-generated content may be incorrect.

Figure 52 Confusion Matrix 4.1

Room for further improvement can be found within the confusion matrix, whose appearance seems to conflict the success of this round of testing, as the desirable diagonal line of high values is barely present. Instead, this matrix appears to represent an inaccurate model, which performed reasonably well on certain classes such as *Lactarius* and *Amanita*, but that accuracy must be improved across entire dataset. Alterations must be made to improve the performance metrics of this model.

## 4.2

Before substantial changes are implemented, the exact same code will be executed, this time utilising *EfficientNetB7*, the other promising member of the *EfficientNet* architectural family. The hyperparameter values will be kept identical to the previous one, to isolate the architecture as the only altered variable during versions 4.1 and [4.2](https://github.com/yungroms/y3_proj/blob/89b4a412c3cbd6ffe5f6ce891884f0032992de72/4.2%20(6.4)/4.2.py).

As explained during previous versions, switching architectures is as simple as changing the specification of the importation from keras.applications.



Figure 53 EfficientNet Importation 4.2

The only other necessary step is to adjust the definition of the model to match the choice of imported architecture.

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure 54 Model Definition 4.2

## 4.2 Results

Below are the results of version 4.2, utilising the *EfficientNetB7* pre-trained model. Once again, it appears that the new code version created for phase 4 allows the *EfficientNet* architecture to achieve better results than those achieved with its implementation in phase 3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 1.1454973220825195 | 0.5922729969024658 | 0.8210660815238953 | 0.7260596752166748 |
| 2 | 0.744627058506012 | 0.7428907752037048 | 0.709729015827179 | 0.7653061151504517 |
| 3 | 0.6187347173690796 | 0.7811335325241089 | 0.7632706761360168 | 0.7386185526847839 |
| 4 | 0.5369474291801453 | 0.8134928345680237 | 0.5996302366256714 | 0.7998430132865906 |
| 5 | 0.4531058967113495 | 0.8407530784606934 | 0.646831750869751 | 0.7880690693855286 |
| 6 | 0.408512145280838 | 0.8633065223693848 | 0.6600754261016846 | 0.784144401550293 |
| 7 | 0.3473513722419739 | 0.8772308230400085 | 0.6196209788322449 | 0.795918345451355 |
| 8 | 0.31165143847465515 | 0.8897823095321655 | 0.5546718239784241 | 0.8202511668205261 |
| 9 | 0.28371602296829224 | 0.9029221534729004 | 0.5934820175170898 | 0.8116169571876526 |
| 10 | 0.24702267348766327 | 0.9123357534408569 | 0.5574294328689575 | 0.8233909010887146 |

The results recorded are similar to 4.1, which utilised its architectural relative *EfficientNetV2S*, with the training metrics of 4.2 actually surpassing its predecessor, the validation and test accuracies being identical, and the validation and test losses being slightly worse.

|  |  |
| --- | --- |
| loss | 0.5930107831954956 |
| accuracy | 0.8176470398902893 |

The slightly larger distance between the training and validation/test results of 4.2 might be indicative of a little more overfitting, as seen in the plots below. But in general, the results of 4.1 and 4.2 are comparable, demonstrating the similar abilities of the two *EfficientNet* versions in use.

A graph of different colored lines

AI-generated content may be incorrect.

Figure 55 Accuracy & Loss 4.2

Unfortunately, the confusion matrix again dampens the success of round 4.2, appearing almost identical to the matrix of 4.1. The desired diagonal line of true positives is absent, meaning that this model is also struggling to predict the classes with much success. This issue could be due to the fact that the *shrooms\_ds\_split* dataset is poorly balanced, meaning that the model is fed more images of certain classes. This would then cause the model to become overly familiar with some classes and not familiar enough with others.

A blue squares with white text

AI-generated content may be incorrect.

Figure 56 Confusion Matrix 4.2

This issue can either be solved by balancing the dataset, as has been done previously with *shrooms\_ds\_max*, or by calculating and applying class weights to the model during training, enabling the under-represented classes to be given greater importance/emphasis. Both these options will be explored at a later stage of phase 4.

## 4.3

The next code version, [4.3](https://github.com/yungroms/y3_proj/blob/26c900e22d5d1cdba91a58fecca52322fa47bf12/4.3%20(6.7)/4.3.py), returns to the use of the EfficentNetB7 architecture, alongside some additional implementations and an alternative dataset.

Leaving the mushroom datasets for a moment, this version introduces a new version of the *LeafSnap* dataset. Whereas previous versions had utilised only the lab images, which were consistent and clear but limited in volume, this version of the dataset merges the lab and field images for each class, massively increasing the number of images available for use, while sacrificing a little consistency between images. In accordance with a successful integration into the code, the dataset has also been split into training/validation and test sets, with the split between training and validation occurring inside the code. Again, this simply ensures that the set of test images remains consistent. This new dataset, named *LeafSnap\_15\_merged\_split*, can be found [here](https://www.kaggle.com/datasets/yungroms/leafsnap-15-merged-split).

One new inclusion within this code is the classification report, which provides the additional useful performance metrics: precision, recall, F1-score, and support. The creation of the report is handled by classification\_report, an import from *SciKit* *Learn*’s metrics package, which calculates these performance metrics for each class and overall. These metrics are then neatly written to a CSV file, so they can be analysed and tabulated or visualised at a later stage. Explanations of these additional performance metrics can be found here, within the Methodology & Design section of the project.

A screen shot of a computer screen

Description automatically generated

Figure 57 Classification Report 4.3

The only additional feature implemented within 4.3 is the function to store the confusion matrix in CSV format, in addition to the already existing PNG format. Doing so provides an additional copy of the confusion matrix data, enabling a tabular view of the matrix, as well as the creation of new visualisations in the event that something goes wrong with the ones created by the program.

Hyperparameter values within code 4.3 remain unchanged from the previous version.

## 4.3 Results

The table below contains the training and validation results of the test run for version 4.3. Based on the accuracies and losses of the training and validation sets, the EfficientNetB7-implemented model has learned the dataset’s classes and features very well. By the end of the first epoch the results look great, and even with fluctuations along the way, the results in the final row show the improvement of the model’s performance throughout the training process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 0.6041288375854492 | 0.819915235042572 | 0.09760291874408722 | 0.9787685871124268 |
| 2 | 0.09363394975662231 | 0.9751059412956238 | 0.06681859493255615 | 0.9766454100608826 |
| 3 | 0.05764128640294075 | 0.9867584705352783 | 0.10196834802627563 | 0.9639065861701965 |
| 4 | 0.03619294613599777 | 0.9920550584793091 | 0.058931030333042145 | 0.9787685871124268 |
| 5 | 0.08439228683710098 | 0.9692796468734741 | 0.09217304736375809 | 0.966029703617096 |
| 6 | 0.043456628918647766 | 0.9862288236618042 | 0.030875694006681442 | 0.9893842935562134 |
| 7 | 0.017142845317721367 | 0.9973517060279846 | 0.04061240330338478 | 0.9830148816108704 |
| 8 | 0.016026057302951813 | 0.9968220591545105 | 0.07946506887674332 | 0.9681528806686401 |
| 9 | 0.018259607255458832 | 0.9941737055778503 | 0.03020964190363884 | 0.9851379990577698 |
| 10 | 0.010882783681154251 | 0.9978813529014587 | 0.04407785087823868 | 0.9830148816108704 |

These excellent metrics are reinforced by the test results, which are the remarkably low loss of 0.007 and the perfect accuracy of 1.0. Based on the tables above and below, this test has been a complete success.

|  |  |
| --- | --- |
| loss | accuracy |
| 0.007476646453142166 | 1.0 |

Further confirmation of excellent performance can be seen in the graphs below. Although the fluctuation is more pronounced when visualised, particularly for the validation lines, the results still look very promising.

A graph of a graph of a graph

AI-generated content may be incorrect.

Figure 58 Accuracy & Loss 4.3

What is less promising is the first edition of the newly-implemented classification report. Ideally, each of the three key metrics, precision, recall and F1-score, will be values somewhere less than but close to 1. Even though the above results are very promising, the classification report table is full of values barely above 0.

This truly is conflicting data. It can be assured that the test data is entirely separate from the training/validation sets, meaning they are new. The results above indicate that the model performed brilliantly in the test, however according to the table below, it failed to learn or predict any class with much success. The code will have to be re-verified to eliminate the possibility of it being the source of this indiscretion, and to confirm that the difference in suggested performance is a legitimate cause of concern.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| acer\_negundo | 0.1 | 0.1 | 0.1 | 10.0 |
| acer\_palmatum | 0.0 | 0.0 | 0.0 | 6.0 |
| aesculus\_pavi | 0.0 | 0.0 | 0.0 | 7.0 |
| asimina\_triloba | 0.0 | 0.0 | 0.0 | 10.0 |
| cercis\_canadensis | 0.0 | 0.0 | 0.0 | 9.0 |
| chionanthus\_virginicus | 0.0 | 0.0 | 0.0 | 9.0 |
| gleditsia\_triacanthos | 0.0 | 0.0 | 0.0 | 8.0 |
| ilex\_opaca | 0.091 | 0.091 | 0.091 | 11.0 |
| liriodendron\_tulipifera | 0.1 | 0.1 | 0.1 | 10.0 |
| ostrya\_virginiana | 0.2 | 0.2 | 0.2 | 10.0 |
| prunus\_sargentii | 0.0 | 0.0 | 0.0 | 6.0 |
| ptelea\_trifoliata | 0.273 | 0.273 | 0.273 | 11.0 |
| quercus\_montana | 0.1 | 0.1 | 0.1 | 10.0 |
| styrax\_japonica | 0.0 | 0.0 | 0.0 | 6.0 |
| ulmus\_pumila | 0.0 | 0.0 | 0.0 | 6.0 |
| macro avg | 0.058 | 0.058 | 0.058 | 129.0 |
| weighted avg | 0.070 | 0.070 | 0.070 | 129.0 |

The confusion matrix below supports the classification report in its assertion that the accuracy and loss scores are misleading, and that the model is far from performing optimally. Once again, the diagonal line of true positives is non-existent, with values spaced sporadically within the matrix, suggesting poor generalisation and accuracy.

A screenshot of a crossword puzzle

AI-generated content may be incorrect.

Figure 59 Confusion Matrix 4.3

One possible source of this poor performance could be the dataset imbalance, which was pointed out during the conclusion of a previous phase 4 test run. This must be addressed through the implementation of a solution: either through manually balancing the dataset, or through integrating method of calculating class weights, to support under-represented classes and diminish the impact of larger ones. That being said, the next test will use the same code as 4.3, however *EfficientNetV2S* will be utilised as the pre-trained model. An attempt to tackle the class imbalance issue will be implemented following this next test, in versions 4.5 and 4.6.

## 4.4

Even though the results of version 4.3 were underwhelming, suggesting the need for analysis and improvement, version [4.4](https://github.com/yungroms/y3_proj/blob/26c900e22d5d1cdba91a58fecca52322fa47bf12/4.4%20(6.8)/4.4.py) will continue the phase 4 pattern of implementing both chosen *EfficientNet* models into each code version, to satisfy this phase’s main target of determining which of the two is best for the project. After implementing *EfficientNetB7* in the previous version, it is now time to test the implementation of *EfficientNetV2S* within this current code version.

The process of implementing an alternative architecture has been covered in previous sections of the project. However, if brief, all that needs to be done is to specify the desired model within the code’s importation section, then instantiate it within the complete model definition.

A computer screen shot of a program code

Description automatically generated

Figure 60 Model Definition 4.4

## 4.4 Results

The table below contains the loss and accuracy data for both the training and validation sets. Again, it appears as though the models containing the *EfficientNetV2S* and *EfficientNetB7* architectures are highly comparable in terms of performance. The loss values for both training and validation sets can be approximated to 0.03, while the accuracy values approximate to 0.99, or 99%. Although only marginal, these results do surpass those achieved during 4.3, suggesting that *EfficientNetV2S* is the optimal pre-trained architecture, certainly within the version of code used within versions 4.3 and 4.4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 0.7184862494468689 | 0.7934321761131287 | 0.16662226617336273 | 0.9532908797264099 |
| 2 | 0.1375671923160553 | 0.9608050584793091 | 0.06190510839223862 | 0.9893842935562134 |
| 3 | 0.05877101793885231 | 0.9862288236618042 | 0.05029682442545891 | 0.9893842935562134 |
| 4 | 0.04904114082455635 | 0.991525411605835 | 0.03829913213849068 | 0.9915074110031128 |
| 5 | 0.04308179020881653 | 0.9894067645072937 | 0.03887119144201279 | 0.9915074110031128 |
| 6 | 0.0273716002702713 | 0.9947034120559692 | 0.044218435883522034 | 0.9893842935562134 |
| 7 | 0.03577205538749695 | 0.992584764957428 | 0.029522456228733063 | 0.993630588054657 |
| 8 | 0.029924970120191574 | 0.9936440587043762 | 0.03187692537903786 | 0.9915074110031128 |
| 9 | 0.047307584434747696 | 0.9841101765632629 | 0.03998721018433571 | 0.9851379990577698 |
| 10 | 0.029829248785972595 | 0.9899364113807678 | 0.029376517981290817 | 0.993630588054657 |

The excellent results above are replicated, if not exceeded, within the test set. A perfect accuracy score of 1.0 (100%) was achieved, while the loss score of the test set was even lower than those achieved during the training or validation sets, reaching the remarkably low value of 0.006.

|  |  |
| --- | --- |
| loss | 0.005861646495759487 |
| accuracy | 1.0 |

The success of 4.4 is reiterated by the visualisations of the accuracies and losses of both the training sets. In comparison with the graphs produced during 4.3, the plots of 4.4 show less fluctuation as well as less space between the training and validation lines. This indicates more consistent learning and less overfitting, both of which are indications of the superiority of the *EfficientNetV2S* architecture.

A graph of a graph of a graph

AI-generated content may be incorrect.

Figure 61 Accuracy & Loss 4.4

Again, since no analysis has been performed nor any improvements made since 4.3, version 4.4 has produced another underwhelming classification report. Every value within the columns of each of the three important metrics is nowhere near desirable, suggesting in contrast to the results above that the model is poor at accurately classifying any of the 15 categories. As stated previously, one source of this incongruency between results could be the imbalance of the dataset, and so the priority within future versions will be to attempt to address this shortcoming.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| acer\_negundo | 0.0 | 0.0 | 0.0 | 10.0 |
| acer\_palmatum | 0.0 | 0.0 | 0.0 | 6.0 |
| aesculus\_pavi | 0.143 | 0.143 | 0.143 | 7.0 |
| asimina\_triloba | 0.3 | 0.3 | 0.3 | 10.0 |
| cercis\_canadensis | 0.0 | 0.0 | 0.0 | 9.0 |
| chionanthus\_virginicus | 0.0 | 0.0 | 0.0 | 9.0 |
| gleditsia\_triacanthos | 0.0 | 0.0 | 0.0 | 8.0 |
| ilex\_opaca | 0.0 | 0.0 | 0.0 | 11.0 |
| liriodendron\_tulipifera | 0.1 | 0.1 | 0.1 | 10.0 |
| ostrya\_virginiana | 0.0 | 0.0 | 0.0 | 10.0 |
| prunus\_sargentii | 0.0 | 0.0 | 0.0 | 6.0 |
| ptelea\_trifoliata | 0.091 | 0.091 | 0.091 | 11.0 |
| quercus\_montana | 0.0 | 0.0 | 0.0 | 10.0 |
| styrax\_japonica | 0.167 | 0.167 | 0.167 | 6.0 |
| ulmus\_pumila | 0.0 | 0.0 | 0.0 | 6.0 |
| macro avg | 0.053 | 0.053 | 0.053 | 129.0 |
| weighted avg | 0.054 | 0.054 | 0.054 | 129.0 |

As was the case with 4.3, the confusion matrix produced in 4.4 does not reinforce the optimistic accuracy and loss data, instead supporting the poor assessment of the classification report. Effort must be made to improve the alignment of the classification and confusion matrix data with the accuracy and loss data.

A screenshot of a crossword puzzle

AI-generated content may be incorrect.

Figure 62 Confusion Matrix 4.4

## 4.5

In an attempt to positively impact the confusion matrix and classification report, [4.5](https://github.com/yungroms/y3_proj/blob/26c900e22d5d1cdba91a58fecca52322fa47bf12/4.5%20(6.9)/4.5.py) includes the functionality to calculate and apply class weights within its code. By applying the concept of weighting, the imbalance between classes with different volumes can be reduced, ensuring under-represented classes are emphasised appropriately during the training process.

Aside from this new functionality, the other aspects of 4.5’s code are the same as 4.3-4.4, including the hyperparameter values and the input dataset. *EfficientNetB7* is the architecture version used in this test.

The calculation of the class weights is handled by the compute\_class\_weight function imported from the utils package of *Scikit-learn*:



Figure 63 Class Weighting Importation 4.5

Once calculated, the class weights are then stored in a dictionary:

A computer screen with text on it

Description automatically generated

Figure 64 Class Weighting Implementation 4.5

This dictionary is then included as a parameter within the fit function during model training:

A computer screen with white text

AI-generated content may be incorrect.

Figure 65 Model Training 4.5

## 4.5 Results

The training and validation results table below contains very promising accuracy and loss data. In comparison with the previous code version used in 4.3 and 4.4, which did not contain class weighting, 4.5 displays better scores in earlier epochs, while ending with comparably outstanding values. This suggests that the inclusion of class weights has a positive impact on the predictability of the model, allowing it to comprehend the entire dataset sooner.

That being said, the results from the final epoch are actually slightly worse than those of the penultimate epochs, with peak performance occurring between epochs 7 and 9.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 0.5868857502937317 | 0.8389830589294434 | 0.11270599812269211 | 0.966029703617096 |
| 2 | 0.0859081894159317 | 0.977224588394165 | 0.11162427812814713 | 0.9617834687232971 |
| 3 | 0.06531272828578949 | 0.9809321761131287 | 0.05562799051403999 | 0.9830148816108704 |
| 4 | 0.05372973904013634 | 0.9830508232116699 | 0.058138228952884674 | 0.9808917045593262 |
| 5 | 0.050654087215662 | 0.9809321761131287 | 0.05131078511476517 | 0.9893842935562134 |
| 6 | 0.03206135705113411 | 0.9888771176338196 | 0.0506136491894722 | 0.9830148816108704 |
| 7 | 0.01424815971404314 | 0.9968220591545105 | 0.044224608689546585 | 0.9830148816108704 |
| 8 | 0.011650074273347855 | 0.9973517060279846 | 0.02639131247997284 | 0.987261176109314 |
| 9 | 0.006913952063769102 | 0.9989407062530518 | 0.025180019438266754 | 0.9915074110031128 |
| 10 | 0.029858706519007683 | 0.9888771176338196 | 0.05928351357579231 | 0.9787685871124268 |

The test results do suggest that the slight dip in the final epoch is not detrimental, however, as the test loss and accuracy are closer to the values found at the training and validation peaks as opposed to the tenth and final epoch.

|  |  |
| --- | --- |
| loss | 0.014814283698797226 |
| accuracy | 0.9922480583190918 |

The training and validation plots for accuracy and loss clearly visualise the peak around the 8th epoch, as well as the slight dip towards the end. This visualisation suggests that the optimal number of epochs may be slightly less than the currently used value of 10.

A graph of a graph

Description automatically generated with medium confidence

Figure 66 Accuracy & Loss 4.5

Unfortunately, the promising conclusions derived from the data above is not reflected in the classification report or confusion matrix below. Once again, poor values can be found throughout the classification report table, with no class achieving scores greater than 0.2, and the averages at the bottom barely exceeding 0.05. Clearly, implementing class weighting has had little positive effect on these performance metrics, which would have been the desirable outcome.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| acer\_negundo | 0.2 | 0.2 | 0.2 | 10.0 |
| acer\_palmatum | 0.0 | 0.0 | 0.0 | 6.0 |
| aesculus\_pavi | 0.0 | 0.0 | 0.0 | 7.0 |
| asimina\_triloba | 0.0 | 0.0 | 0.0 | 10.0 |
| cercis\_canadensis | 0.0 | 0.0 | 0.0 | 9.0 |
| chionanthus\_virginicus | 0.0 | 0.0 | 0.0 | 9.0 |
| gleditsia\_triacanthos | 0.0 | 0.0 | 0.0 | 8.0 |
| ilex\_opaca | 0.0 | 0.0 | 0.0 | 11.0 |
| liriodendron\_tulipifera | 0.0 | 0.0 | 0.0 | 10.0 |
| ostrya\_virginiana | 0.1 | 0.1 | 0.1 | 10.0 |
| prunus\_sargentii | 0.167 | 0.167 | 0.167 | 6.0 |
| ptelea\_trifoliata | 0.091 | 0.091 | 0.091 | 11.0 |
| quercus\_montana | 0.2 | 0.2 | 0.2 | 10.0 |
| styrax\_japonica | 0.0 | 0.0 | 0.0 | 6.0 |
| ulmus\_pumila | 0.0 | 0.0 | 0.0 | 6.0 |
| macro avg | 0.051 | 0.051 | 0.051 | 129.0 |
| weighted avg | 0.054 | 0.054 | 0.054 | 129.0 |

As is the case for the classification report table above, the confusion matrix shows results which are equally conflicting with the accuracy and loss results. The inclusion of class weighting appears to have had negligible positive impact on the confusion matrix, as the distribution of values is no better than in the previous two rounds of testing. Clearly, more improvement is required within the model’s code to increase the alignment of the first results section with the second.

A screenshot of a crossword puzzle

AI-generated content may be incorrect.

Figure 67 Confusion Matrix 4.5

## 4.6

In continuation of the phase 4 trend of testing both *EfficientNet* versions within the code before applying updates, test [4.6](https://github.com/yungroms/y3_proj/blob/26c900e22d5d1cdba91a58fecca52322fa47bf12/4.6%20(6.10)/4.6.py) will simply include *EfficientNetV2S* as the pre-trained architecture. Based on the results of 4.5, little optimism is held regarding the outcome of the classification report and confusion matrix, but promising results are anticipated for the training and validation loss and accuracy results.

The figure below displays the simple variation required to implement *EfficientNetV2S* in place of *EfficientNetB7*:

A screen shot of a computer program

Description automatically generated

Figure 68 Model Definition 4.6

## 

## 4.6 Results

As expected, test run 4.6 has produced excellent accuracy and loss results during training and validation. From the 2nd epoch onwards, the training and validation accuracy values around 95% or higher. From the 3rd epoch onwards, both the training and validation loss values are less than 0.1. In the 10th and final epoch, the validation results of 99.6% for accuracy and 0.017 for loss are representative of a model with fantastic predictability.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 0.7032853960990906 | 0.7976694703102112 | 0.17450560629367828 | 0.9426751732826233 |
| 2 | 0.13693077862262726 | 0.9661017060279846 | 0.07915088534355164 | 0.987261176109314 |
| 3 | 0.07867415249347687 | 0.9735169410705566 | 0.04280387610197067 | 0.9957537055015564 |
| 4 | 0.048738569021224976 | 0.9872881174087524 | 0.04487476870417595 | 0.9893842935562134 |
| 5 | 0.03554752469062805 | 0.992584764957428 | 0.040064722299575806 | 0.9893842935562134 |
| 6 | 0.0530436672270298 | 0.9862288236618042 | 0.029003949835896492 | 0.993630588054657 |
| 7 | 0.030787236988544464 | 0.9931144118309021 | 0.035568684339523315 | 0.9915074110031128 |
| 8 | 0.04747902229428291 | 0.9883474707603455 | 0.030998704954981804 | 0.9915074110031128 |
| 9 | 0.04850604012608528 | 0.9894067645072937 | 0.016278285533189774 | 0.9957537055015564 |
| 10 | 0.03503787890076637 | 0.9904661178588867 | 0.016674816608428955 | 0.9957537055015564 |

The training and validation results of the final epoch are reflected in the results of the test set, in which an accuracy of 100% and a loss of 0.0039 were achieved. The model has performed equally impressively on this previously unseen data as it did with the training and validation sets.

|  |  |
| --- | --- |
| test\_loss | 0.003856297116726637 |
| test\_accuracy | 1.0 |

The training and validation plots of the accuracy and loss graphs below also reflect the brilliant results found in the above tables. In comparison to the graphs of 4.5, it appears as though 4.6 has produced even better results, as there is less separation between the training and validation plots, and no dip in performance in the last epoch. As has been the case in the last three version pairs, it seems that *EfficientNetV2S* slightly outperforms its architectural relative *EfficentNetB7*.

A graph of a graph of a graph

Description automatically generated with medium confidence

Figure 69 Accuracy & Loss 4.6

Unfortunately, the optimistic results above do not follow through to the classification report. In fact, unlike in the accuracy and loss results, *EfficientNetV2S* actually performs slightly worse than *B7* in important metrics such as precision, recall and F1-Score. The highest value within the table below is 0.167, a little less than the highest value scored in 4.5, which was 0.2. Similarly, 4.5 achieved a weighted average of 0.054, while 4.6 only achieved a weighted average of 0.047.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| acer\_negundo | 0.1 | 0.1 | 0.1 | 10.0 |
| acer\_palmatum | 0.167 | 0.167 | 0.167 | 6.0 |
| aesculus\_pavi | 0.0 | 0.0 | 0.0 | 7.0 |
| asimina\_triloba | 0.0 | 0.0 | 0.0 | 10.0 |
| cercis\_canadensis | 0.0 | 0.0 | 0.0 | 9.0 |
| chionanthus\_virginicus | 0.0 | 0.0 | 0.0 | 9.0 |
| gleditsia\_triacanthos | 0.125 | 0.125 | 0.125 | 8.0 |
| ilex\_opaca | 0.0 | 0.0 | 0.0 | 11.0 |
| liriodendron\_tulipifera | 0.1 | 0.1 | 0.1 | 10.0 |
| ostrya\_virginiana | 0.1 | 0.1 | 0.1 | 10.0 |
| prunus\_sargentii | 0.167 | 0.167 | 0.167 | 6.0 |
| ptelea\_trifoliata | 0.0 | 0.0 | 0.0 | 11.0 |
| quercus\_montana | 0.0 | 0.0 | 0.0 | 10.0 |
| styrax\_japonica | 0.0 | 0.0 | 0.0 | 6.0 |
| ulmus\_pumila | 0.0 | 0.0 | 0.0 | 6.0 |
| macro avg | 0.051 | 0.051 | 0.051 | 129.0 |
| weighted avg | 0.047 | 0.047 | 0.047 | 129.0 |

The confusion matrix of test 4.6 is comparable to 4.5, clearly suggesting poor predictive success across all classes. One important detail which has yet to be stated in regards to the underwhelming classification reports and confusion matrices seen so far is that the test set sizes have always been very small. With minimal input in testing, poor performance appears more obviously within these metrics. Although unlikely to drastically improve the outcomes of these metrics, the utilisation of larger test sets will at least increase the amount of testing done by the model, giving it more of a chance to prove its predictive ability.

A screenshot of a crossword puzzle

Description automatically generated

Figure 70 Confusion Matrix 4.6

## 

## 4.7

Although substantial improvement has not yet been made regarding the classification reports or confusion matrices, highly promising accuracy and loss results have been attained during recent tests.

In order to improve the underwhelming resultant metrics of versions 4.5 and 4.6, one suggestion was made at the end of 4.6, that being to increase the input data volume. In order to do this, the code of [4.7](https://github.com/yungroms/y3_proj/blob/26c900e22d5d1cdba91a58fecca52322fa47bf12/4.7%20(6.14)/4.7.py) will implement the [*shrooms\_ds\_max\_split*](https://www.kaggle.com/datasets/yungroms/shrooms-ds-max-split) dataset, in order to test if the larger input dataset could positively impact its corresponding classification report and confusion matrix.

In the tests of previous versions that utilised this larger dataset, the number of epochs was increased correspondingly, to ensure the model had sufficient time to maximise its accuracy. The same approach is taken for test 4.7, which will utilise 20 epochs for training and validation, as opposed to other recent versions which utilised 10 epochs. In relation to this, a learning rate scheduler has also been implemented, to ensure that the model does not become susceptible to overfitting.

A computer screen shot of white text

Description automatically generated

Figure 71 Learning Rate Scheduler 4.7

Based specifically on the graph plots of 4.5, which demonstrated a slight decline in performance between the penultimate and final epochs, a second improvement was suggested: to implement early stopping. The aim of this implementation is to halt the learning process if the validation loss score declines for a specified number of epochs, then restore the architecture’s weight values to those of the best-performing epoch.

The final new inclusion within version 4.7 is the re-introduction of data augmentation, which aims to perceivably expand and diversify the input dataset, hopefully leading to a more robust and learned model.

A screen shot of a computer program

Description automatically generated

Figure 72 Data Augmentation 4.7

The last few pairs of phase 4 versions sought to compare the performance of two pre-trained architectures, *EfficientNetV2S* and *EfficientNetB7*. Based on the test results of these versions, the decision has been made to utilise only one of these architectures moving forward, that being *EfficientNetV2S*.

## 4.7 Results

Below is the training and validation results table containing the accuracy and loss values for each epoch. Although the number of epochs was initially set at 20, it appears as though the newly implemented early stopping took effect towards the end, as epoch 19 is the final epoch recorded within the table.

The results contained within this table are very strong: The values of the initial epoch are decent, but vast improvement was made within each column as the epochs progressed, producing very promising results by the 3rd and 4th epochs which continued until the end of the training process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| epoch | train\_loss | train\_accuracy | val\_loss | val\_accuracy |
| 1 | 0.9144284725189209 | 0.6934955716133118 | 0.4609358012676239 | 0.8450018763542175 |
| 2 | 0.45768609642982483 | 0.8415535688400269 | 0.32750093936920166 | 0.8914264440536499 |
| 3 | 0.3157256841659546 | 0.8904070854187012 | 0.23558269441127777 | 0.9206289649009705 |
| 4 | 0.2528485655784607 | 0.9136171936988831 | 0.20826363563537598 | 0.9344814419746399 |
| 5 | 0.20632946491241455 | 0.9299017190933228 | 0.2016143649816513 | 0.9385997653007507 |
| 6 | 0.15220916271209717 | 0.9503041505813599 | 0.17110732197761536 | 0.9483339786529541 |
| 7 | 0.13130581378936768 | 0.957229733467102 | 0.16023558378219604 | 0.9539498090744019 |
| 8 | 0.11799157410860062 | 0.9607861638069153 | 0.1519443690776825 | 0.956570565700531 |
| 9 | 0.1122802272439003 | 0.9625643491744995 | 0.14747922122478485 | 0.9580681324005127 |
| 10 | 0.1112806424498558 | 0.9639681577682495 | 0.1466628760099411 | 0.9595656991004944 |
| 11 | 0.1100851520895958 | 0.9645296931266785 | 0.14535628259181976 | 0.9588169455528259 |
| 12 | 0.09937489032745361 | 0.9672437906265259 | 0.14668355882167816 | 0.9576937556266785 |
| 13 | 0.10316438972949982 | 0.9675245881080627 | 0.14409106969833374 | 0.9588169455528259 |
| 14 | 0.09852111339569092 | 0.9702386260032654 | 0.14611268043518066 | 0.9573193788528442 |
| 15 | 0.10268872231245041 | 0.9683668613433838 | 0.1454615294933319 | 0.9558218121528625 |
| 16 | 0.09869015216827393 | 0.9692091941833496 | 0.14352281391620636 | 0.9595656991004944 |
| 17 | 0.10429789125919342 | 0.9648104906082153 | 0.14844660460948944 | 0.9576937556266785 |
| 18 | 0.108112633228302 | 0.9653720259666443 | 0.1457192599773407 | 0.9580681324005127 |
| 19 | 0.10222280770540237 | 0.9676181674003601 | 0.1446332037448883 | 0.9573193788528442 |

In fact, the loss and accuracy values achieved during 4.7’s testing are the best results attained by any version implementing a pre-trained architecture and mushroom dataset. The loss value achieved is 0.18, while the accuracy is around 95%. While these values are not quite as impressive as those achieved while using a variation of the *LeafSnap* dataset, it is worth remembering that the mushroom datasets contain realistic photographs in natural settings, while the *LeafSnap* datasets contain cleaner, less user-realistic images. Based on this fact, the accuracy of around 95% is an amazing result, as is the test loss of 0.18, indicating great promise for future deployment within an app for users.

|  |  |
| --- | --- |
| test\_loss | 0.18072621524333954 |
| test\_accuracy | 0.945147693157196 |

Again, the accuracy and loss graphs provide a visual accompaniment to the above tables, clearly demonstrating the successful training and validation results.

A graph of a graph

AI-generated content may be incorrect.

Figure 73 Accuracy & Loss 4.7

Unfortunately, it appears as though all implementations made within 4.7 did not have as substantial an impact in the classification report as was hoped for. Although this classification report is better than those generated during 4.5 and 4.6, containing higher values across all metric columns, these values are still far lower than what would be considered successful. The larger input dataset and the inclusion of augmentation has had the desired impact, and so the secret of aligning the classification report with the success of the accuracy and loss results has not yet been discovered.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class | Precision | Recall | F1-Score | Support |
| Agaricus | 0.0759493670886076 | 0.0759493670886076 | 0.0759493670886076 | 79.0 |
| Amanita | 0.05128205128205128 | 0.05063291139240506 | 0.050955414012738856 | 79.0 |
| Boletus | 0.1125 | 0.11392405063291139 | 0.11320754716981132 | 79.0 |
| Cortinarius | 0.1038961038961039 | 0.10126582278481013 | 0.10256410256410256 | 79.0 |
| Entoloma | 0.14285714285714285 | 0.13924050632911392 | 0.14102564102564102 | 79.0 |
| Hygrocybe | 0.11392405063291139 | 0.11392405063291139 | 0.11392405063291139 | 79.0 |
| Lactarius | 0.1038961038961039 | 0.10126582278481013 | 0.10256410256410256 | 79.0 |
| Russula | 0.1511627906976744 | 0.16455696202531644 | 0.15757575757575756 | 79.0 |
| Suillus | 0.07692307692307693 | 0.0759493670886076 | 0.07643312101910828 | 79.0 |
| macro avg | 0.10359896525263025 | 0.1040787623066104 | 0.10379990040586456 | 711.0 |
| weighted avg | 0.10359896525263027 | 0.10407876230661041 | 0.10379990040586456 | 711.0 |

The confusion matrix below reflects the outcome of the classification report, as even though the matrix is populated with higher values than its recent predecessors, these values do not align desirably into the diagonal line of true positives. Instead, they are distributed across the entire matrix, indicating that the separation between the two different collections of results (accuracy/loss and classification/confusion matrix) is as prominent as ever.

A blue and white grid with white text

Description automatically generated

Figure 74 Confusion Matrix 4.7

With the conclusion of phase 4, the project reaches an important point. The brilliant accuracy and loss values obtained during versions 4.6 and 4.7 suggest that it is time to move forward to the development of the application which will house the models produced by these versions. However, the comparably poorer classification reports and confusion matrices generated by these same versions indicate that further work is required, to improve these results to necessary standards.

Due to the fact that the final submission deadline is rapidly approaching, it is necessary that both tasks are carried out simultaneously. This is not a huge issue, as these tasks are mutually exclusive from each other, however emphasis must be placed on the development of the application, so that a fully-implemented version can be produced to demonstrate the abilities of the models which have been painstakingly developed throughout the project’s span.

Therefore, the next steps of the project will be to explore the development of an application prototype, while attempting to solve the discrepancies in the performance metrics.