COMP5130 Principles of Data Science Assignment 2

Alexander Daniel Mars, SID: 450347582, Unikey: amar6958

Setup

The goal of this project is to build a model that can classify hundreds of mushroom species from their images; a model capable of predicting more than tens of species has never before been published[1,2]. Since this project involves a novel dataset and model, there is no well-defined benchmark against which performance can be measured. Instead, multiple models will be tested, therefore the null hypothesis is that these models will have equal levels of performance. The alternative hypothesis is that the models are not equal, in which case at least one model will be better than the others. Performance will be measured on a singular test set that is not involved in model training. Mcnemar's test will be used to perform pairwise comparisons between model performance to confirm or reject the null hypothesis[3]. Additional metrics will include precision, f1-score, and recall. Models will be trained using a validation set to fine tune parameters. An 80:10:10 split of the original data will be used for training, validation, and test sets respectively.

Approach

As this was an image classification problem, convolutional neural networks (CNNs) were used as they are the gold standard for object recognition[4]. Four models were trained including three ResNet models that aimed to classify all images at the species level. The fourth model was a hierarchical ResNet model which first aimed to classify mushrooms at the genus level, and then at the species level for any genera that had more than one species. Where possible, transfer learning was utilised. Transfer learning in this context means using ResNet models that have been pre-trained on the ImageNet dataset, and re-training them for specific use on images of different mushroom species. This approach is effective as the first few layers in a CNN learn general features of the image, such as corners, lines, and colours, whilst the last few layers can be specialised to certain tasks[5]. Model one (M1) was pre-trained ResNet34 at the species level, model two (M2) was pre-trained XResNet50, model three (M3) was XResNet18, and model 4 (M4) was a pre-trained ResNet34 at the genus level, with pre-trained ResNet18 used for within genus (species) classification (Figure 1). For M4, the entire training set of images was used for the genus-level model, with the set of images for each genus used to train the species-level classifiers.

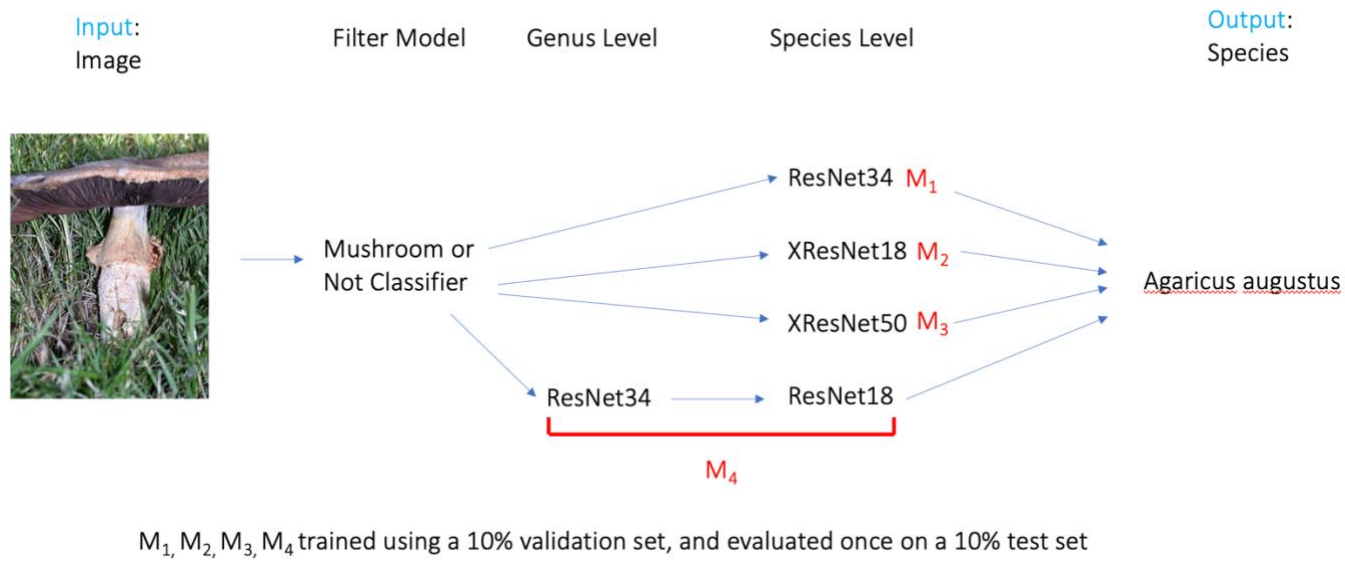


***Figure 1**: An overview of the models used in this project. The models take in an unlabelled image and try to return the species label for that image. First each image was passed through a 'mushroom or not' classifier with images predicted as 'not a mushroom' discarded. The remaining images were passed to four different models. M4 is a hierarchical model which makes predictions first at the genus level, and then at the species level for the predicted genus. The remaining models make predictions only at the species level.*

Here 'XResNet' refers to a set of ResNet models modified to speed up model convergence[6]. Currently, only XResNet50 is available with pre-training on ImageNet, this is why ResNet34 was chosen instead of XResNet34. XResNet18 was included as 'wildcard' model; having the smallest depth out of all CNN models tested it should be the fastest to train.

Pre-trained ResNet18 on the other hand was chosen for the task of within genus classification as there will only be up to a handful of species per genus, thus the increased depth of the other models is unlikely to be necessary.

One additional data-cleaning step was conducted prior to training these four models. A 'mushroom or not' classifier was created using ResNet34 to filter out junk images from the mushroom observer database. Despite the filtering steps in stage 1 of the project, some high-confidence observations contain microscopy images of spores, or images where the mushroom of interest is far away. This occurs because users may upload multiple images with each observation to the mushroomobserver.org, however all images are linked to the same confidence rating. To create a 'mushroom or not' classifier, three images from each of the top 1000 species gathered in stage 1 were randomly selected and manually screened for junk images. These junk images were added to a 'not mushroom' class whilst the rest were kept in a 'mushroom' class. Additionally, a number of mushroom species were selected at random and screened for junk images to add to the 'not mushroom' class in order to reduce the class imbalance between 'mushroom' and 'not mushroom'. Examples of these junk images can be seen in the Jupyter notebooks. A 10% validation set was used to tune parameters during training the mushroom or not classifier. The mushroom or not classifier was trained to 95% accuracy (as measured on the validation set). The entire set of images was then passed through the classifier and images predicted as not mushroom were removed. Approximately 20,000 images were removed, reducing the image set to about 140, 000 images. Classes with less than 30 images were subsequently removed, leading to 992 classes (species) being retained from the 1000 that were collected in stage 1. After removing these images, the 10% test set was created and set aside until all models were trained.

The FastAI library was used for loading and splitting the data into training and validation sets[7]. Using the FastAI library, state of the art parameter tuning methods can be incorporated into automated model training protocols[7]. This includes using discriminative learning rates, and automatically finding the best learning rate, weight decay, and momentum for a model[8]. All models were trained using an automated call back which saved the model state after the epoch where validation loss was lowest (example code from FastAI given in Appendix 1).

A singular test set was deemed more appropriate than using cross-fold validation simply because of the scale of the model and expected training time. Using a single test set meant that classical approaches for hypothesis testing such as the student's t test would not be appropriate, thus Mcnemar's test was employed.

Results

As expected, XResNet18 was the fastest to train, taking 191 minutes, whilst the slowest was the hierarchical model, taking approximately 350 minutes to train. This is because the hierarchical model first learned the entire training set at the genus level (362 classes) and then learned 146 additional models for each genus that contained more than one species. ResNet34 achieved the highest accuracy on the validation set of 64%. A summary of the results is given in Table 1.

***Table 1**: Summary of training results for the four models. Accuracy is measured on the validation set.*

Model	Validation Loss	Training Loss	Accuracy	Epoch	Time (minutes)
ResNet34 (M1)	1.52	1.09	0.644	20	250
XResNet18 (M2)	2.05	0.6	0.58	20	191
XResNet50 (M3)	1.73	0.85	0.62	10	300
Genus-Model (M4)	1.21	0.63	0.71	20	350

To assess the goodness of fit of each model, the training loss and validation loss were plotted against the number of epochs (Figure 2). No models appeared to overfit to the training data as the validation loss decreased across all epochs. The rate of decrease of both validation and training loss appeared to slow over the last few epochs. This could indicate that the models were converging to an optimal fit, although further training could have led to a better fit overall. Each model's loss curves appeared to level out when the models were switched from a 'frozen' state to an 'unfrozen' state. This is to be expected as unfreezing the model means training the weights of all layers in the CNN, whilst in the frozen state, only the final few layers are being trained.

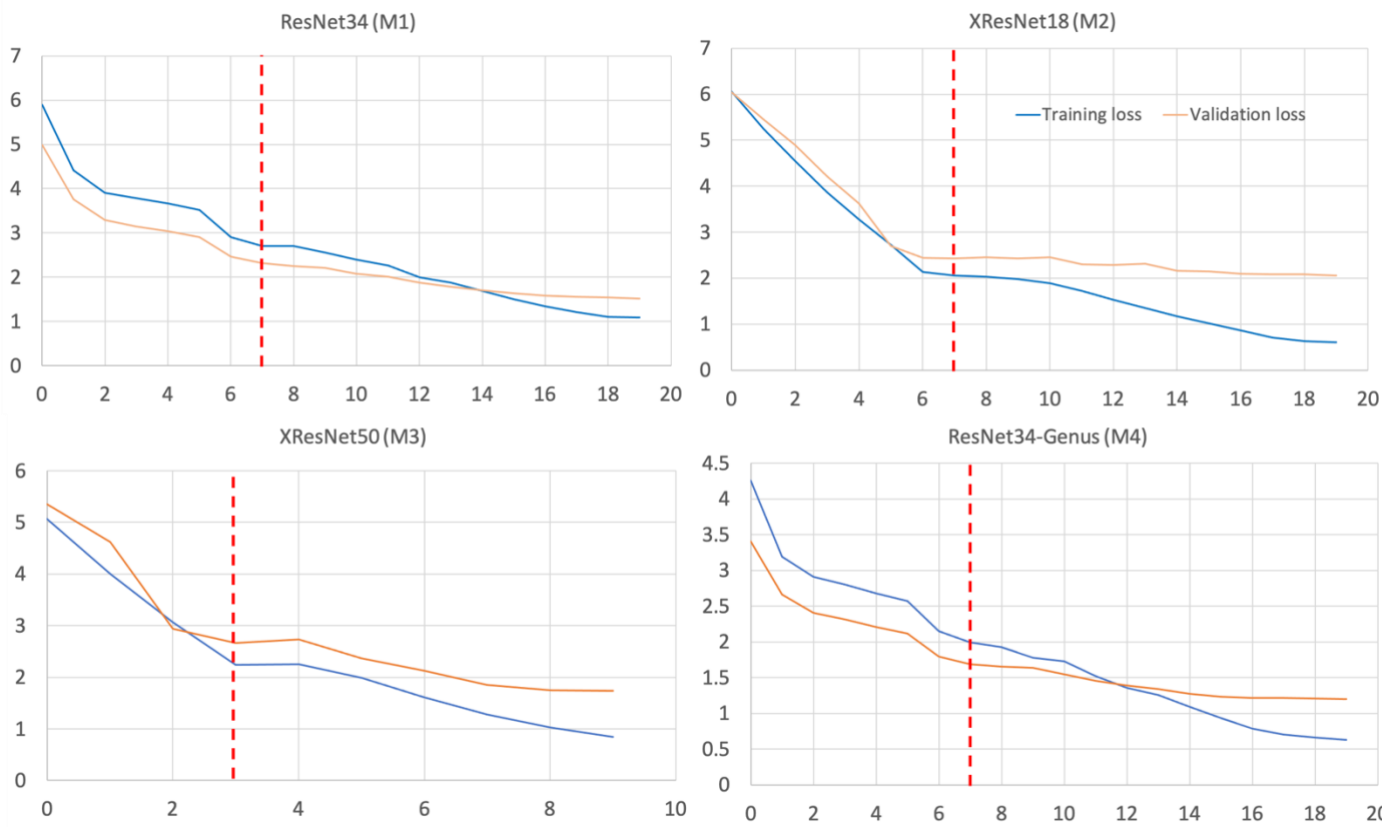
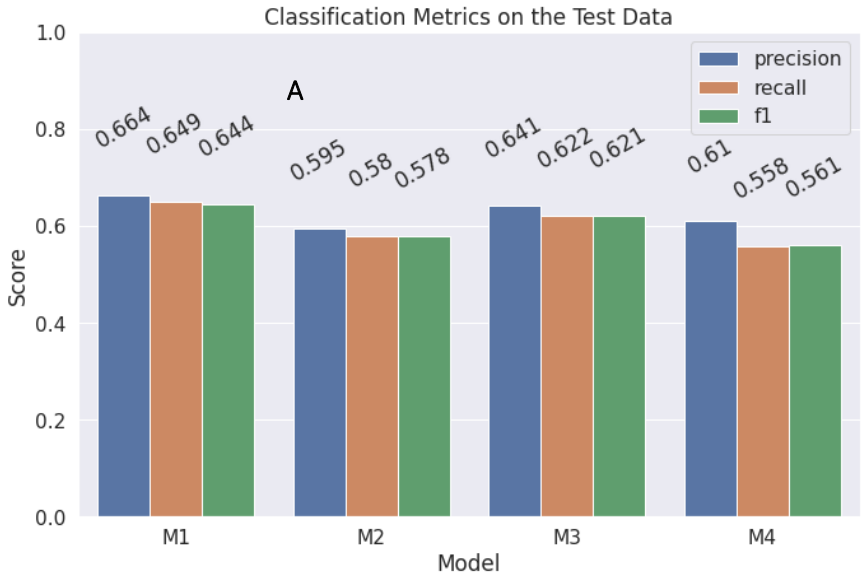


***Figure 2**: The training loss (blue) and validation loss (orange) were plotted against the number of epochs (x-axis) for each of the four models. The red dashed line indicates the epoch at which the model was 'unfrozen' during training. Unfreezing the models means allowing the weights across all layers of the CNN to be trained, prior to unfreezing only the final few layers are being trained. This is how transfer learning is achieved. Here M4 is not the full hierarchical model, it is only the first layer of the hierarchy; the model classifying at the genus level. There w ere 146 species-level models (ResNet18) which also make up M4 whose results are not displayed.*

To determine the best model after training, each model was evaluated once on the test set. This involved feeding the entire test set of images to each model and comparing the prediction made with the label of the image. Accuracy on the test set was measured as a percentage of the number of correct predictions out of the total number of images predicted. Precision, recall, and f1-score were calculated as weighted averages across all classes predicted (Figure 3A). ResNet34 (M1) achieved the highest accuracy (precision) of 66% whilst XResNet50 (M3) achieved the second highest precision of 64%. The hierarchical model (M4) achieved a precision of 61% with XResNet18 having the worst precision of 59%. M4 also showed the greatest difference between precision and recall with values of 61% and 56% respectively. This indicates that M4 was the least effective at correctly identifying a mushroom species from its image. Overall, the performance of each model appeared to be quite similar across precision, recall, and f1-score. Furthermore, precision on the test set was similar to the precision recorded to the validation set. This supports the result that each model was well fit. If the models were overfit, we would expect precision on the test set to be lower than scores reported on the validation set.

To determine whether the performance of each pair of models was significantly different , Mcnemar's test was conducted in a pairwise manner; comparing M1 with M2, M2 with M3, and M1 with M3 (Figure 3B). When using a single evaluation set, Mcnemar's test is a more appropriate method of comparison then a t-test[3]. Contingency tables based on the number of correct predictions per instance were constructed for the pairs M1-M2, M2-M3, and M1-M3. This requires that the order of predictions made on the test set was the same for models M1, M2, and M3. Subsequently, Mcnemar's test was not performed for comparisons with M4 as the order of predictions on the test set was different. With M4, predictions were first made at the genus level and grouped by the predicted genus. For each predicted genus, the within-genus model was then loaded to make all the predictions at the species level. This meant that each of the 146 within-genus models was only loaded into RAM once to make prediction on the entire test set (15,000 images) practical. However, it also meant that the order of predictions made was not the same as the other three models.



B

	M1	M2	M3
M1	1.0	10^{-99}	10^{-17}
M2	10^{-99}	1.0	10^{-46}

Figure 3: A: Barplot of precision, recall, and f1-score for each model as measured on the test set. These metrics are reported as the weighted average across all classes. B: Results from pairwise applications of Mcnemar's test. Each cell displays the p-value associated with performance Mcnmar's test between the models in the row and column labels.

The p-value from Mcnemar's test between all models was much less than 0.01. Thus, using this threshold the null hypothesis could not be retained. This means that the performance of models was not identical. Subsequently M1 appeared to be the best model as it had both the highest precision and highest recall.

It is possible that the performance of each model could be improved further by balancing the classes. As the dataset contained the 1000 most frequently observed mushroom species on mushroomobserver.org, there was a large difference in the number of images per class. For example, the most frequently observed species; Agaricus augustus, had over 2500 images, whilst the least frequent species only had tens of images. This is known as the 'long-tail' problem in machine learning[9,10]. A full classification report of the models after training showed that the precision for some species was close to 1.0, whilst for other species it was close to 0.0 (see Jupyter notebooks). This difference in precision between classes could reflect the imbalanced classes. Class balancing could involve creating artificial images for the classes with a low number of images[11], or using resampling techniques where images from the low count classes are seen more than once by the model during training[10].

Additionally, alternative methods of creating a hierarchical model could be explored. The motive for a hierarchical model is to allow for new species of mushroom to be incorporated without retraining the entire model. Classifying at the genus level may not be the best approach, instead classifying based on similar features such as colour, shape, or other features could be employed. An unsupervised learning algorithm could be used to create clusters that then define the hierarchy.

Alternatively, a set of models could be created where the classes within each model have a similar number of images. This would avoid the class imbalance problem but would require additional time at inference, as a prediction from each model would be required in order to find the most likely class. These ideas could be tested in future but were not explored in this project due to time constraints.

Conclusion

The goal of this work was to train a model capable of predicting hundreds of mushroom species from their images. A model capable of predicting more than tens of classes has never before been published. Multiple models were tested with ResNet34 (model M1) achieving the best result of 65% accuracy across 992 different species of mushroom. The null hypothesis that all models tested would have the same performance was rejected, thus the results of model M1 were deemed significant. This is a fine-grained classification task, and thus further improvement may require data augmentation such as class balancing., or more complex hierarchical models.

References

1. Chitayae, N. & Sunyoto, A. Performance Comparison of Mushroom Types Classification Using K-Nearest Neighbor Method and Decision Tree Method. *2020 3rd Int Conf Information Commun Technology Icoiact* **00**, 308–313 (2020).

2. Lu, C.-P. & Liaw, J.-J. A novel image measurement algorithm for common mushroom caps based on convolutional neural network. *Comput Electron Agr* **171**, 105336 (2020).

3. McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* **12**, 153–157 (1947).

4. Rawat, W. & Wang, Z. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation* (2017).

5. Zeiler, M. D. & Fergus, R. Visualizing and Understanding Convolutional Networks. *Arxiv* (2013).

6. He, T. *et al.* Bag of Tricks for Image Classification with Convolutional Neural Networks. *2019 Ieee Cvf Conf Comput Vis Pattern Recognit Cvpr* **00**, 558–567 (2019).

7. Howard, J. & Gugger, S. Fastai: A Layered API for Deep Learning. *Information* **11**, 108 (2020).

8. Smith, L. N. A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay. *Arxiv* (2018).

9. Wang, T. *et al.* Computer Vision – ECCV 2020, 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV. *Lect Notes Comput Sc* 728–744 (2020) doi:10.1007/978-3-030-58568-6_43.

10. He, H. & Garcia, E. A. Learning from Imbalanced Data. *Ieee T Knowl Data En* **21**, 1263–1284 (2009).

11. Frid-Adar, M. *et al.* GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing* **321**, 321–331 (2018).

Appendices

Appendix 1: Example code to load data and train a model using FastAI

```
mushroom_db = DataBlock(blocks = (ImageBlock, CategoryBlock), # inputs are images, outputs are categorie
                    get_items = get_image_files,    # get images from folder names
                    get_y = parent_label,           # get labels by processing filename
                    splitter = RandomSplitter(valid_pct=valid_pct, seed=0),
                    item_tfms = Resize(224),        # resize all images to 224 x 224
                    batch_tfms = aug_transforms()) # augment images to improve generalisation
mushroom_dl = mushroom_db.dataloaders(mushroom_images)
mushroom_dl.show_batch(max_n=9)
```

```
learn = cnn_learner(mushroom_dl, resnet34, metrics=accuracy)
```

```
learn.fine_tune(14, freeze_epochs=6, cbs=SaveModelCallback(monitor='valid_loss', fname='species-resnet34'))
```

epoch	train_loss	valid_loss	accuracy	time
0	5.888876	4.984786	0.148931	11:24
1	4.410781	3.762519	0.266958	10:33
2	3.904086	3.298206	0.321331	11:08