Goldsmiths
UNIVERSITY OF LONDON

Using deep learning to classify animals at the family/subfamily level in camera trap images

James Lee

Supervisor: Antoaneta Serguieva

Submitted in partial fulfilment of the requirements
for the degree of Master of Science (MSc) in Data Science
of the University of London

Goldsmiths, University of London

August 2021

I certify that this project and the research to which it refers, are the result of my own work

ABSTRACT

Camera traps are a vital source of information for ecologists, biologists and conservationists. They can capture millions of images of animals in their natural habitat without human intervention. However, analysing the images produced by camera traps is a time-consuming operation. Recently, a number of studies have attempted to perform automated image analysis using convolutional neural networks. While results have been positive (up to 97% accuracy in image classification in some cases), there remains a problem of generalisation. Models perform well when faced with images produced by camera traps that they have previously been trained on. However, performance suffers once the models attempt to classify images from new locations. This is a result of overfitting.

To attempt to address this issue, this study has developed a sequence of classifiers. The first classifies animals by their family/subfamily taxonomic level, rather than by their species. This allows the classifier to be exposed to a wider range of locations during training, with the aim of reducing overfitting. The next classifier is then designed to classify only animals from a certain subfamily/family. This classifier returns the species of the animal. Three of these species-level classifiers were developed for this project, classifying images from the families Canidae, Felinae and Sciuridae. Alongside these models, one benchmark model was also developed, designed to take images from all families and return the species label.

While the species-level models showed improved results when compared directly against the benchmark model (91%, 88% and 91% accuracy for Canidae, Felinae and Sciuridae respectively, versus 74% accuracy for the benchmark model), when the extra step of first classifying the images by family was taken into account, the results were actually worse than the benchmark results (a 77% accuracy for the family-level model effectively reduced the species-level accuracies to just 70%, 68% and 70% respectively).

Furthermore, the out-of-sample results were mixed, with 66% and 53% accuracy for the Canidae models, but just 1% accuracy for the Sciuridae model.

These results suggest that the approach of developing family-specific classifiers does not help with the issue of generalisability, although the study does acknowledge that the results obtained were somewhat limited by the computational power available, and that future work may be able to improve upon these results.

ACKNOWLEDGMENTS

The author would like to thank Dr. Sarah Rauchas, Ms. Antoaneta Serguieva, and Mr. Radu-Andrei Nedelcu of the University of London for their advice and assistance in relation to this project.

This project has relied on data provided by the camera-trap network "Snapshot Safari" managed by the University of Minnesota Lion Center and hosted on Zooniverse and the Minnesota Supercomputing Institute. This study was made possible by the provision of data from the Snapshot Karoo, Snapshot Kgalagadi, Snapshot Enonkishu, Snapshot Camdeboo, Snapshot Mountain Zebra, and Snapshot Kruger projects.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Camera traps are devices which take photographs every time they are triggered by some action or activity, typically some form of motion. While camera traps have been in existence in one form or another since the 1890s ('Camera traps and science - how did we get here?', 2014), advances in technology mean that they are now more affordable and accessible. This has led to an explosion in their use, in particular when it comes to the study of wildlife. The non-invasive nature of camera traps enables researchers to collect vast amounts of data without interfering with the animals' behaviour in any way. This data can be used for a multitude of research purposes, including studying animal diversity, the abundance of a particular species, species' behaviour, and the interaction between species (Wearn and Glover-Kapfer, 2017).

However, while camera traps make it simple to generate huge amounts of data, it is then necessary to extract information from that data. This typically involves humans viewing the images one-by-one and recording any pertinent information, such as the type and number of animals present in an image. This can be extremely time consuming, and, if not performed by volunteers, costly. The well-known Snapshot Serengeti dataset, which consisted of 1.2 million images in 2013, required over 28,000 volunteers for annotation at the time (Swanson *et al.*, 2015). As this dataset now contains over 7 million photographs, it is easy to imagine the amount of labour that would be required to manually analyse each image.

Recently, a number of studies have attempted to use machine learning techniques to remove this time-consuming process by automatically extracting information from camera trap images. In particular, deep learning, a special branch of machine learning utilising artificial neural networks, has been used to perform both object detection (*is there an animal in the image?*) and image classification (*which animal is it?*).

While these studies have reported high levels of accuracy, they are not without issues, primarily the problem of overfitting and generalisation to new locations (see section 2.2 for details). This project, which focuses on the latter task of image classification, aims to address these issues through the use of sequential convolutional neural network (CNN) classifiers, first classifying an animal at the family/subfamily level, before a second model classifies the animal at the species level. In doing so, the study aims to investigate whether CNNs dedicated to classifying animals from a specific family/subfamily can provide more accurate results than one generic CNN animal classifier. A related question, which will also be explored, is whether family/subfamily specific CNNs generalise better to new locations.

Therefore, the aims of this project are:

- to develop a series of family/subfamily specific convolutional neural networks (CNN) to classify images of animals, using camera trap images from various locations; and
- to evaluate the performance of these family/subfamily classifiers against generic CNN animal classifiers.

2. BACKGROUND AND RELATED WORK

2.1. Deep Learning

Neural networks are so named due to the influence of the anatomy of the brain on their design. In the brain, neurons are connected to other neurons via synapses. These synapses carry signals from one neuron to another, which stimulates (or inhibits) that neuron's activity. Depending on the strength of the signal and the neurons connected, a certain action then results. While a single neuron alone lacks any real power, it is the combination of these neurons that enables animals to perform such diverse and often complex tasks, such as image recognition, with ease (*Overview of neuron structure and function (article) | Khan Academy*, no date).

Neural networks attempt to mimic this structure. A neural network is made up of at least three layers of neurons (or nodes). The first layer, the input layer, takes the inputs to the network (such as images). The last layer is the output layer, which outputs the results of the network (such as image classifications). Any layers in between are known as hidden layers. These layers form the mapping between inputs and outputs (see Figure 1). In a simple, feed forward network, each node in one layer is connected to every node in the next layer. These connections have weights plus an additional bias term, which are typically passed through some activation function, before the output is passed along to the next layer. As with signal strengths in the brain, different sizes of weights and bias terms will result in a different output. A neural network learns the correct weights using an algorithm called backpropagation. Backpropagation involves calculating the loss of the network, which is the difference between the expected output and the actual output. It then uses another algorithm, gradient descent, to update the weights and biases, working backwards through the network, in such a way as to minimise this loss.

Neural networks are very powerful systems, with the potential to perform learning tasks in a huge range of applications. Computer vision is one field in which neural networks excel due to the complex patterns that can be learned. A particular type of neural network that is used most frequently in computer

vision is the convolutional neural network (CNN) (Lecun *et al.*, 1998). CNNs, unlike fully connected neural networks, pay attention to the structural information contained in an image, such that the proximity of one pixel to another is considered an important attribute (Nielsen, 2015). A series of convolutions (filters) are passed over the image in order to detect features, such as edges. For image classification tasks, the final (output) layer of a CNN will have as many nodes as there are classes, each of which typically uses the SoftMax activation function. This outputs a value between 0 and 1, which can be considered the probability that the image which has been passed through the network is of that particular class.

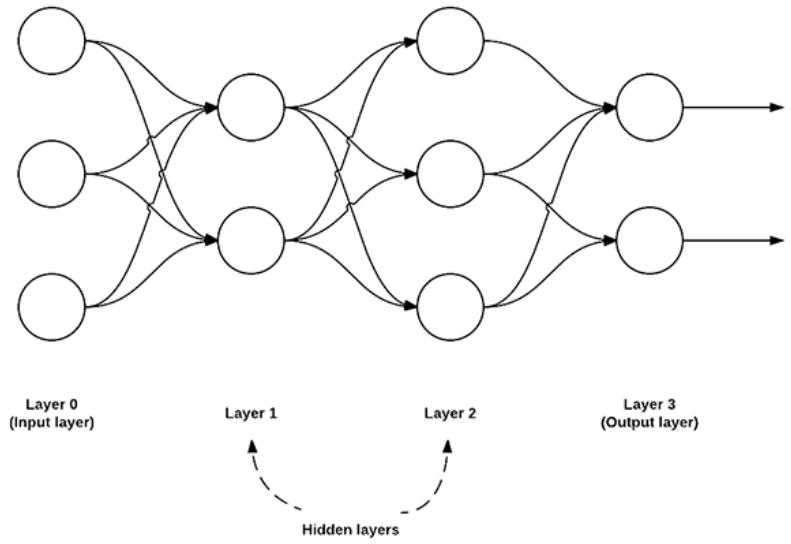


Figure 1: Neural network example with two hidden layers (Rosebrock, 2016)

CNNs can achieve very high levels of accuracy in image classification tasks. However, to do so, they generally have to be very deep (have many layers). This results in a large number of parameters for the network to learn. For example, the Inception-V3 architecture (Szegedy *et al.*, 2015) has 312 layers and 23,817,352 trainable parameters. Training such a network is computationally expensive. One way to reduce the amount of computation required is to use a technique known as transfer learning.

> Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem. For instance, features from a model that has learned to identify racoons may be useful to kick-start a model meant to identify tanukis (*Keras documentation: Transfer learning & fine-tuning*, 2020).

To utilise transfer learning, a pretrained CNN, with all of its learned weights and biases, is used for the new task. Only the final layers are modified, to allow the network to make classifications based on the new dataset. The parameters for these layers must still be learned, but this involves significantly less

computation than training the full network. Optionally, some (or even all) of the earlier layers may also be retrained. This can result in better performance, but at a computational cost.

2.2. Related Work

One of the largest studies into camera trap image classification with deep neural networks was conducted by Norouzzadeh *et al*. in 2018. This study utilised the Snapshot Serengeti dataset, which at that time consisted of 3.2 million images of 48 different animal species. The study implemented a two-stage process. First, an object detection model was used to determine whether there was an animal present in the image or not (camera traps may be triggered by some movement other than an animal and so capture an 'empty' image). Then all of the non-empty images were passed through an image classifier model to determine the animal classifications. The study found this two-stage process to give better results than simply attempting to classify all of the images, including empty ones. The authors reported a highest accuracy of 93.8%, rising to 96.6% if the classifications were limited to those the model was confident about (using a threshold of 43%).

Given these results, the approach of using an object detector before the classification stage to filter out empty images seems promising. However, this naturally increases the amount of computation involved. Given that the current project already involves developing two separate classifiers (family-level and species-level, see section 3.2.2 for details), any further computation is to be avoided at this stage. Utilising an object detector may be an interesting approach for any future work building on this project.

Norouzzadeh *et al.*, 2018, also examined transfer learning, concluding that it was only helpful (in terms of improving accuracy) on smaller datasets. Due to the size of the dataset used in the current project, transfer learning may in fact help to increase the accuracy of the models. Furthermore, when assessing transfer learning, Norouzzadeh *et al.* did not consider the computational cost, a factor that is considered important in this project.

Tabak *et al*. (2019) conducted a similar study, using 3.37 million images from 5 different US states, plus a further 5,900 images from Canada, which were used as out-of-sample data to evaluate model performance. Twenty-seven different species were classified using the ResNet-18 architecture. The results reported 97.6% accuracy on the test data. However, the accuracy reduced to 82% on the out-of-sample data. The authors suggested that this may be due to the different environments found in the out-of-sample data that were not present in the training data.

Greenberg (2020) supports this view in his discussion of the current state of automated animal classification using neural networks on camera trap images. According to Greenberg, while studies report high levels of accuracy, this accuracy does not hold up when the models are used on unseen data.

In fact, this is a recurring issue identified in the literature. Camera traps are generally static, resulting in images with largely unchanging backgrounds. A consequence of this is that a neural network may pay too much attention to these backgrounds during training; "Models can quickly learn the features of specific camera 'stations' (the spatial replicate in camera trap studies) such as the general background instead of learning features of the animal itself" (Whytock *et al*., 2021, p. 2). As a result, the same animal against an entirely new background is often not recognised by the model. Beery, van Horn and Perona (2018) investigated this issue using images from 20 camera traps in different locations. They reported a 97% increase in the error rate when classifying images from unseen locations versus locations seen during training.

The problem is one of generalisability. In terms of the neural network, a lack of generalisability is a result of the model overfitting the training data. The model has learned noise in the images, such as the background scenery, rather than specific features of the animals. The model then becomes 'confused' when faced with new backgrounds, resulting in poor performance. For an animal classifier model to be useful, it must be able to generalise to new locations. Otherwise, at the extreme end of the spectrum, a separate model would be required for every single camera trap, with each model requiring thousands or even millions of labelled training images. This, of course, would be infeasible.

In their later paper, Tabak *et al*. (2020) saw improved results on out-of-sample data in some cases (Canada - 91% accuracy) but still faced difficulties in others (Missouri 36%, Caltech 56%). The report stated:

> Transferability of machine learning models remains a complication for implementing these models more broadly to camera trap data and, in many cases, it is most productive for scientists to build models that are trained directly on their study sites…By including more diverse datasets when we train future models, we may be able to train a model that can be accurate in more locations. (Tabak *et al*., 2020, pp. 10380-10381)

Schneider *et al*. (2020) were primarily concerned with producing a more practical image classifier, one which required fewer images to train and performed well in new locations. They used 47,000 images, containing 55 different species from 36 locations across Canada. While their model performed well on this relatively small dataset (95.6% top-1 accuracy), they again saw a significant decrease in accuracy when

faced with images taken from new (untrained) locations (68.7%). The authors concluded that this was a result of overfitting.

Along with generalising to new locations, the study identified two further critical factors affecting performance: size of the training set, and imbalanced datasets. The authors proposed a minimum figure of 1,000 images per class in order to achieve 95% recall, while acknowledging that this figure should be taken only as an indicated starting point and may not generalise to all datasets and models.

Whytock *et al*., 2021 also focused on the issue of generalisation. They used a dataset of 347,000 images of 26 different forest mammals and birds, from multiple camera traps in different countries in Central Africa. ResNet-50 was used as the model architecture. After training, the model was evaluated on out-of-sample data consisting of 24,000 images from 227 different camera stations (again, located in forests in Central Africa). Their results showed close to or above 95% accuracy on the out-of-sample data. While these results do appear promising, the authors acknowledged that this may be partially due to the similarity between camera trap locations in Central African forests. The paper goes on to state that:

> forest camera traps in Central Africa are often deployed in very similar settings and using standard protocols (e.g. typically attached to trees 30–40 cm above ground level), with animals captured at a predictable distance from the camera (usually on a path) with a general background of green and brown vegetation. This is in contrast to camera trap images from more open habitats, where animals are often detected across a wide range of distances and backgrounds (Whytock *et al*., 2021, p. 10)

At this point it is reasonable to conclude that generalisation remains one of the biggest issues for camera trap image classification. One often-proposed solution is to develop location-specific models, so that camera traps in one location are trained on images from that location and are only used to classify images from the same location. While the results of Whytock *et al*., 2021 suggest that this approach can achieve success, it is not without its difficulties. The obvious question is how to define separate locations. Is it sufficient to have different models for different biomes (one for tundra, one for desert, and so on)? Or is the requirement closer to the situation discussed above of having a separate model for every camera trap? The reality is likely somewhere in between.

This project instead implements an alternative solution. While domain-specific models are employed, the domain in this case is the family/subfamily taxonomic rank of the animal, rather than location. First, a CNN classifies the images according to which subfamily/family of animal is present (referred to from this point on as the family-level classifier). Depending on the results, the image is then passed through a CNN designed to classify by species only images of that particular subfamily/family (referred to from this point on as the species-level classifier). The justification for this approach is

threefold. Firstly, it reduces the number of classes the species-level classifier needs to recognise, enabling it to focus on learning the more subtle differences between species of the same subfamily/family. Secondly, it enables the family-level classifier to be exposed to a larger range of locations for a single class. As an example, the single class Pantherinae contains, amongst others, lions, jaguars and tigers. These animals typically reside in very different locations to one another, yet to the family-level classifier, they are all of the same class. This should force the classifier to pay less attention to the location and instead focus on shared characteristics of the animals.

Finally, it is thought that this approach will help to address the issue of animal rarity. Rare and endangered animals are captured by camera traps much less frequently than common animals. This leads to a scarcity of training data, which negatively impacts the performance of an image classifier when it has to classify images of such animals (Schneider *et al*., 2020; Shahinfar, Meek and Falzon, 2020; Whytock *et al*., 2021). Returning to the example of Pantherinae, while camera trap images of the critically endangered Amur Leopard may be in short supply, images of Pantherinae in general are more prevalent. The result is an abundance of training data for the family-level classifier, followed by a species-level classifier with only a small number of classes to recognise, thus (hopefully) reducing the number of training images required to perform to a high level.

This approach of having specific classifiers for specific groups of animals is actually suggested as potential future work in the paper by Whytock *et al*. (2021). However, whereas Whytock *et al*. suggest rather vague groupings (bird, small carnivore, cat, African Elephant, pangolin etc.), this project uses the subfamily/family class to get more consistent, scientific groupings on the basis of shared physical characteristics.

As an aside, not all species have a subfamily. This project uses the subfamily label whenever it exists and sufficient images are available, and otherwise uses the family label. For the sake of clarity, the remainder of this paper refers only to the family level.

Animal rarity can also be addressed by using transfer learning. According to Norouzzadeh *et al*. (2019)

> Transfer learning is highly beneficial when we have a limited number of labeled samples to learn a new task (for example, species classification in camera trap images when the new project has few labeled images), but we have a large amount of labeled data for learning a different, relevant task (for example, general-purpose image classification). (Norouzzadeh *et al*., 2019, p. 3)

Many of the existing CNN architectures have been pretrained on the ImageNet dataset, a diverse database of over 14 million images, from goldfish to doormats to espressos (ImageNet, 2021). The paper by

Shahinfar, Meek and Falzon, 2020, which used images from three distinct datasets from Australia, Africa and North America, employed six different CNN architectures (ResNet-18, ResNet-50, ResNet-152, DnsNet-121, DnsNet-161, and DnsNet-201), all of which were pretrained on the ImageNet dataset. The paper examined the number of images needed for an accurate classifier, concluding that, with transfer learning; "150-500 images per class is sufficient to achieve reasonable classification accuracy for project-specific camera trap models" (Shahinfar, Meek and Falzon, 2020, p. 8). The authors also reported that shallow tuning, which simply replaces the last layer of the CNN, was sufficient to achieve high accuracy, although they did state that deep tuning (replacing more layers) may have offered a better performance with more epochs.

Willi *et al.*, (2019) also employed transfer learning. However, rather than using the ImageNet dataset, they used models pretrained on the Snapshot Serengeti dataset. They found that "Transferring information from CNNs trained on large datasets ("transfer-learning") was increasingly beneficial as the size of the training dataset decreased and raised accuracy by up to 10.3%." (Willi *et al.*, 2019, p. 80).

This project employs transfer learning at two stages. Firstly, the family-level classifier is pretrained on the ImageNet dataset. The species-level classifiers are then built from this family-level classifier, giving them the benefit of pretraining on both the ImageNet dataset and the dataset created for this project.

3. METHODOLOGY

3.1. Data

The 'Labeled Information Library of Alexandria: Biology and Conservation' "is a repository for data sets related to biology and conservation" (*LILA BC (Labeled Image Library of Alexandria: Biology and Conservation)*, no date). All of the datasets are released under the Community Data License Agreement.

Many of these datasets contain camera trap images from various locations worldwide. A number of these camera trap datasets have been used in the studies referenced in section 2.2. However, these studies tend to use just one dataset, or occasionally train on one dataset before testing on a separate one. As discussed, this can lead to overfitting, due to the camera trap learning details of the particular location, or features common to camera trap locations in those environments (Whytock *et al.*, 2021, p. 10).

To reduce such overfitting, this project instead uses a combination of images from various datasets: Island Conservation Camera Traps (123 locations from 7 islands in 6 countries) (*Island Conservation*, no date), Snapshot Karoo, Snapshot Cambedoo, Snapshot Kruger, Snapshot Mountain Zebra (all South Africa) (*Zooniverse*, no date), Snapshot Kgalagadi (Namibia, South Africa & Botswana) (*Zooniverse*, no date), Snapshot Enonkishu (Kenya) (*Zooniverse*, no date), Snapshot Serengeti (Tanzania) (Swanson *et al.*, 2015), ENA24-detection (Eastern North America) (Yousif, Kays and He, 2019), WCS Camera Traps (12 countries over 4 continents) (*Saving Wildlife and Wild Places - WCS.org*, no date), Wellington Camera Traps (New Zealand) (Anton *et al.*, 2018), Caltech Camera Traps (Southwestern United States) (Beery, van Horn and Perona, 2018), North American Camera Trap Images (United States) (Tabak *et al.*, 2019). This approach allows the models to be trained on a wide range of images featuring different animal species against diverse backgrounds.

In general, the accuracy of a neural network increases with the amount of training data. This is particularly true with image classifiers (Pokhrel, 2019). However, the computational expense also increases as more training images are used. As discussed further in section 3.2, the models were built and trained on Google Cloud Platform (*Cloud Computing, Hosting Services, and APIs*, no date) in order to take advantage of the increased computing power available. However, this computing power does not come free. Since the study was conducted using the author's personal account, it was necessary to strike a balance between using a large dataset to improve performance, and using a smaller dataset to reduce costs. Based on the findings of Schneider *et al.*, 2000, discussed above, a figure of 1,000 images per species was deemed to be appropriate, especially when taking into account the suggestion by Shahinfar, Meek and Falzon, 2020 that, in fact, 150-500 images per class is sufficient when transfer learning is employed.

It was originally intended to classify all species that had at least 1,000 images across the datasets. 112 such species were found and 1,000 images of each were chosen at random, (see section 3.2.1 for details of this process). 505 of these images were found to be corrupted and so were deleted, resulting in a dataset of 111,495 images.

In order to perform machine learning, a dataset must be split into three parts: training, validation and testing. As the name suggests, the training set is used to train the model. While the model is training, its performance is evaluated using the validation set. If the model fits well on the training set but performs poorly on the validation set, it is a sign that the model is overfitting to the training data, and some alterations must be made, such as changing hyperparameters of the model or adding more training data. Finally, after all such alterations have been made, the test set is used to evaluate the true performance of the model. While traditional machine learning typically uses a 70/30 train-test split, computer vision tasks

with neural networks tend to have more data, and so can afford to use larger splits for training, even up to using 98% of the data for training (*Splitting into train, dev and test sets*, no date). Given the size of the dataset used in this project, a balance was struck by using 80% of the data for training, 10% for validation and 10% for testing.

This first train/validation/test set was built for the benchmark model, a species-level classifier (see section 3.2.2 for details), by selecting at random 80% of the images from each species for training, 10% for validation and 10% for testing. The next neural network to be built was the family-level classifier. For convenience, the same images from the benchmark datasets were copied and assigned to the appropriate family category in order to form the train/validation/test set for this model (57 categories in total from the 112 different species). This approach left the overall dataset split the same (80/10/10), but the number of images per class naturally changed (see Appendix for details).

Upon attempting to train the benchmark model, it quickly became apparent, even with the use of a GPU, that training would simply be too expensive for this number of images, Therefore, it was necessary to reduce the size of the datasets. Rather than reducing the number of training images for each category, it was decided that a better approach would be to simply focus on a smaller number of categories (species). Examining the family-level dataset revealed that 33 of the families contained only one species. While these images would be useful for the benchmark model, they would be of no use for the final species-level classifiers, as any classifier dedicated to one of those families would only have one species to categorise, which would always result in 100% accuracy. Therefore, these families and species were removed from both the family-level and species-level training, validation and test datasets. This left 62,922 images for training, 7,855 images for validation, and 7,882 images for testing.

The final models to be built were the three species-level classifiers (Canidae, Felinae and Sciuridae – see section 3.2.2). Because the family-level classifier was used as a pre-trained model to create these three species-level classifiers, the validation images used in the family-level classifier could not be reused for validation of the species-level models (one of the golden rules of machine learning is that a model should never be evaluated on data which it has already been trained on). Instead, the training and validation sets were merged and used for training. Despite the fact that the family-level model had been exposed to the test set during evaluation, this data could still be considered unseen for the purposes of the species-level classifiers, due to the fact that the images were not incorporated into the training and so did not impact upon the network's learned parameters. Consequently, the test sets could be used as the validation sets for the species-level classifiers. It was then simply a case of choosing at random 100 new images for each species (within the Canidae, Felinae and Sciuridae groups) to form the test sets for the three species-level models.

Finally, the Missouri Camera Traps dataset (Zhang *et al.*, 2016) was used as out-of-sample data in order to gauge how both the benchmark model and species-level classifiers performed on images from camera traps they had not seen during training. From the three families used for the final classifiers, three species were found to be present in the Missouri dataset, one for each classifier. All such available images were used for testing (Fox – 501 images, Ocelot – 539 images, American Red Squirrel – 639 images).

3.2. Technical Approach

3.2.1. Data Preparation

Alongside the actual datasets, LILA BC provides metadata for each dataset in JSON format, containing the file name and the classification (label). One of the main issues during the data preparation stage was resolving the different naming conventions used in the various datasets. For example, while species of zebra can be found in the WCS dataset under the scientific name of the species (such as 'equus grevyi'), the Karoo Snapshot dataset uses the English name ('Mountain Zebra') while the Snapshot Serengeti dataset simply lists the generic 'Zebra'. In order to resolve this, the classifications from the JSON files were combined into a single CSV file. A column was then added for the appropriate species label, along with a column for subfamily and family. It was then a case of manually checking each classification in order to determine and record the correct and consistent species, subfamily (where appropriate) and family labels. The subfamily and family levels were taken from Wikipedia ('Wikipedia, the free encyclopedia', 2021). The author acknowledges and accepts the widely recognised issues with relying on Wikipedia as a source. However, it was felt that the information should be of sufficient accuracy for the purposes of this project. It is further accepted that manual relabelling of the data is a time-consuming process. Future projects could potentially automate this step. However, this would require a combination of web scraping and natural language processing, which was considered to be beyond the scope of this project.

Once finalised, the CSV file was combined with the JSON metadata files in a Jupyter Notebook in order to create a Pandas dataframe containing the filename, name of the dataset, species name, subfamily and family for every image: over 13 million rows in total. This dataframe was then reduced so as to only contain the species with at least 1,000 images, but this still left over 11.5 million potential images. As discussed above, the decision was made (initially) to use just 1,000 images from each species (112,000 images in total), taken proportionately from each dataset. Where a dataset had more images than required (for example, Snapshot Serengeti had 53,607 images of African Elephants, with only 310

needed), the sample method from Python's random module was used to select the necessary number of images.

Windows Powershell was used along with AzCopy and the Shared Access Signature URLs provided by LILA BC in order to download to local disk only the chosen images. Three main folders were created (training, validation, test), each containing subfolders named after the species to be classified. Alongside this, three other folders were created (training_family, validation_family, test_family) to hold the data for the family-level classifiers. Images were downloaded directly into the appropriate species folder, and then copied into the appropriate family folder to form the family-level datasets.

For the species-level test data, a dataframe of unused images was created, from which 100 images were chosen at random for each species required, using the same approach as for the benchmark model in order to get an equal number of images from each dataset (wherever possible). The unused images dataframe was also used to select the out-of-sample images from the Missouri Camera Trap dataset. These images were also downloaded to local disk using AzCopy and organised into subfolders according to their correct classification. This use of subfolders named according to their classification was essential in order to use Keras's ImageDataGenerator function during training and evaluation of the model (see section 3.2.2).

Finally, a Google Cloud Storage Bucket was created, and the credentials added to the Jupyter Notebook. The 'glob' module plus Google Cloud's 'storage' method were used to copy the files from local disk to the Bucket, making sure to preserve the folder structure of the datasets.

3.2.2. Model Building and Training

In order to benefit from transfer learning, the pre-existing MobileNetV2 architecture (Sandler *et al.*, 2019) trained on the ImageNet dataset (ImageNet, 2021) was used. MobileNetV2 is specially designed for use on mobile devices. It is smaller than other existing CNN architectures, such as InceptionResNetV2 (Szegedy *et al.*, 2016) and thus requires less computation. Despite only having a depth of 157 layers and 3.5 million trainable parameters (compared to 572 layers and over 23.8 million trainable parameters for InceptionResNetV2), MobileNetV2 still achieves a 70% top-1 accuracy on the ImageNet dataset according to Keras (versus 80% for InceptionResNetV2) (*Keras documentation: Keras Applications*, no date). Furthermore, Schneider *et al.*, (2020) achieved 93% accuracy in their camera trap classification study using MobileNetV2.

The pre-trained model (referred to from now on as the 'base model') was loaded with the 'include_top' parameter set to false. This loaded the model without the final two layers; a Global Pooling 2D layer and a final Dense layer. In order to determine the best hyperparameters, 10% of the benchmark dataset was uploaded to Google Drive (*Cloud Storage for Work and Home – Google Drive*, no date) and mounted on a Google Colab notebook. Google Colab (*Google Colaboratory*, no date) is an online service that provides a limited amount of free computing resources, including access to GPUs or TPUs. A number of different architectures and hyperparameters were then explored, including different final layer configurations, different image pre-processing techniques, and different optimisers. The best performing network architecture consisted of four new layers added to end of the base model: a flatten layer to convert from an input shape of 7x7x1280 into a linear array of length 62,720; a dense (fully connected) layer, a dropout layer (with a dropout rate of 0.2) to combat overfitting; and finally, another fully connected layer with the SoftMax activation function in order to output the final class predictions. The shape of this output layer was determined by the number of classes for the particular classifier. This final layer architecture was adopted from the Coursera course, 'Convolutional Neural Networks in TensorFlow' (*Exploring dropouts*, no date).

In addition to these output layers, pre-processing layers were added to the beginning of the network in order to scale the input pixels between -1 and 1, the expected scaling for MobileNetV2 (*Keras documentation: MobileNet and MobileNetV2*, no date; *tf.keras.applications.mobilenet_v2.preprocess_input*, no date). The final architecture can be seen in Figure 2 (with the base model layers suppressed).

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
tf_op_layer_Cast (TensorFlow [(None, None, None, 3)]   0
_________________________________________________________________
tf_op_layer_RealDiv (TensorF [(None, None, None, 3)]   0
_________________________________________________________________
tf_op_layer_Sub (TensorFlowO [(None, 224, 224, 3)]     0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
flatten (Flatten)            (None, 62720)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              64226304
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 79)                80975
=================================================================
Total params: 66,565,263
Trainable params: 65,027,279
Non-trainable params: 1,537,984
_________________________________________________________________
```

Figure 2: Benchmark neural network summary

The base model was loaded with weights learned on the ImageNet dataset. In order to preserve the details learned by the base model, the first 150 layers were frozen to prevent the weights from being updated during training. This left the last nine layers of the network to be trained which, due to the flatten layer, meant a total of over 65 million trainable parameters.

As well as providing the class labels, which are taken from the subfolder name in which an image is located, Keras's ImageDataGenerator was used to perform data augmentation on the training images. Data augmentation is a common technique in image classification, used in order to reduce overfitting. It "artificially increases the size of the training set by generating many realistic variants of each training instance" (*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition [Book]*, 2019, p. 450). As the model trains, random transformations are applied to the images in order to generate these variants and expose the model to more data than it would otherwise see. Common transformations include rotating the image, flipping the image, zooming in to the image, and more. For details of the data augmentation applied in this project, see the Appendix.

Several optimisers and learning rates were experimented with using the Google Colab platform, including SGD and RMSProp (Hinton, no date). However, Adam (Kingma and Ba, 2017) was found to give the best performance, with a relatively low learning rate of 0.0001.

Due to the heavy computational cost involved in training CNNs, a GPU was required to perform the model training and evaluation. Google Colab limits the runtime to just 12 hours per day, which would not have been sufficient for training on the full datasets. For this reason, Google Cloud's AI Platform Notebooks was used to carry out the project. Details of the instance used can again be seen in the Appendix.

The first model created was the benchmark model. This model was trained, validated and tested on the species-labelled datasets. The purpose of this model was to set a benchmark for classifying species from this particular dataset and with this particular network architecture, against which the species-level, family-specific classifiers could be evaluated. The model was trained for 50 epochs, with the full model and its performance metrics being saved after each epoch (using code adapted from PyImageSearch, Rosebrock, 2019).

Next the family-level model was created, using the same model architecture as the benchmark model, but with a smaller output layer due to the smaller number of classes. This model was also trained for 50 epochs. As can be seen in section 4, the training and validation accuracy of the family-level model started to diverge around epoch 46. As a result, the family-level model with its weights from epoch 45

was used to build the three species-level models. This time, only the last layer of the model was replaced so that the output size matched the number of classes for each classifier.

Ideally, 24 separate species-level classifiers would have been built, one for each family classified by the family-level classifier. However, once again, computational expense prevented this. Instead, the three largest family groups (in terms of number of species) were chosen for the analysis. These were Canidae, Felinae, and Sciuridae, each containing six different species.

3.2.3. Model Evaluation

For the evaluation stage, each model was reloaded from the epoch where it displayed the best balance between training accuracy and validation accuracy (see section 4). The benchmark model and the three species-level models were then evaluated against their test set and out-of-sample datasets. The family-level dataset was only evaluated against its test set, in order to ensure that it offered a reasonable performance and thus was suitable for use as the base model of the species-level classifiers. The overall performance of the family-level classifier was not the focus of the study, and so its performance on out-of-sample data was not considered important for the analysis.

Keras's ImageDataGenerator was used during the evaluation process, this time without data augmentation, in order to feed the test images to the model along with the correct labels. A dataframe was created to record the final loss, accuracy, and top five accuracy of each model; top five accuracy being the accuracy of the model when its top five predictions are evaluated against the ground truth. A further dataframe was then generated to record the model's prediction for each image, alongside the correct label and the file path of the image being predicted.

3.3. Ethical Considerations

Camera traps occasionally capture images of humans. However, for each of the datasets used in this project, LILA BC states that "The original data set included a "human" class label; for privacy reasons, we have removed those images from this version of the data set" (*LILA BC (Labeled Image Library of Alexandria: Biology and Conservation)*, no date). This removes any concerns of privacy that may have otherwise been associated with the project.

One concern that potentially remains is that of facilitating illegal poaching. Highly accurate, automated classifiers could potentially be used by poachers to locate and track endangered animals, leading to an increase in poaching. Falzon *et al.*, (2019) cite this as their reason for not allowing end-users to train their own models using their ClassifyMe software

> …ClassifyMe could be used to rapidly scan camera trap images whilst in field to detect the presence of particular species such as African elephants which are threatened by poaching. To address this concern, a host of security features were incorporated into ClassifyMe. (Falzon *et al.*, 2019, p. 4)

ClassifyMe also requires verified security credentials to access and use the model for classification. This combined approach of limiting access and preventing model modifications appears to represent the best approach to prevent misuse, and it is recommended that this be adopted for any model put into production following on from this project.

4. RESULTS

4.1. Training Results

During the training phase, each model's performance was recorded against six separate metrics: training loss, validation loss, training accuracy, validation accuracy, training top five accuracy and validation top five accuracy. Figures Figure 3Figure 7 show the training and validation accuracy per epoch. Tables showing the full metrics for each model can be found in the Appendix.

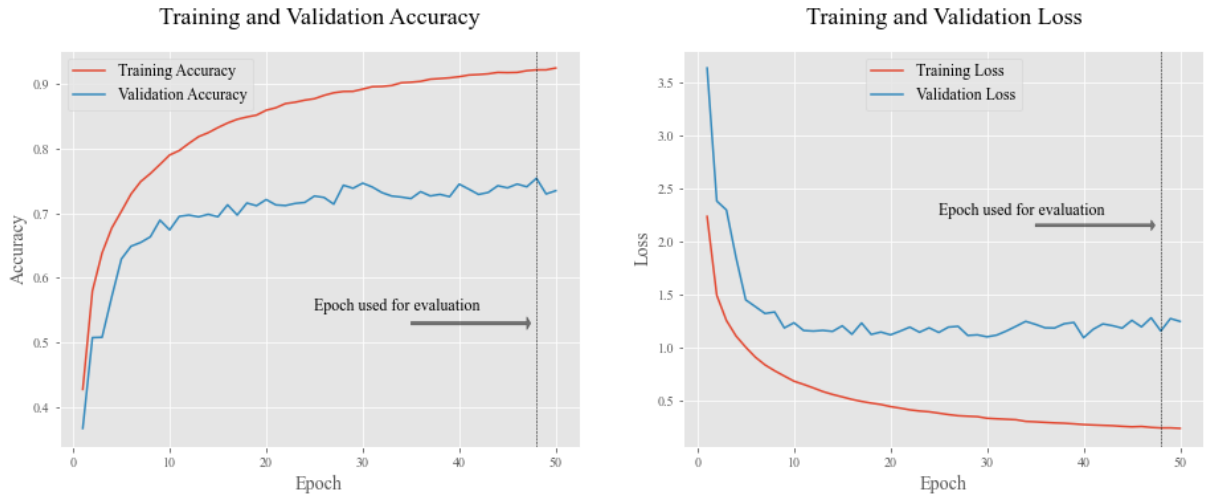


Figure 3: Benchmark model training

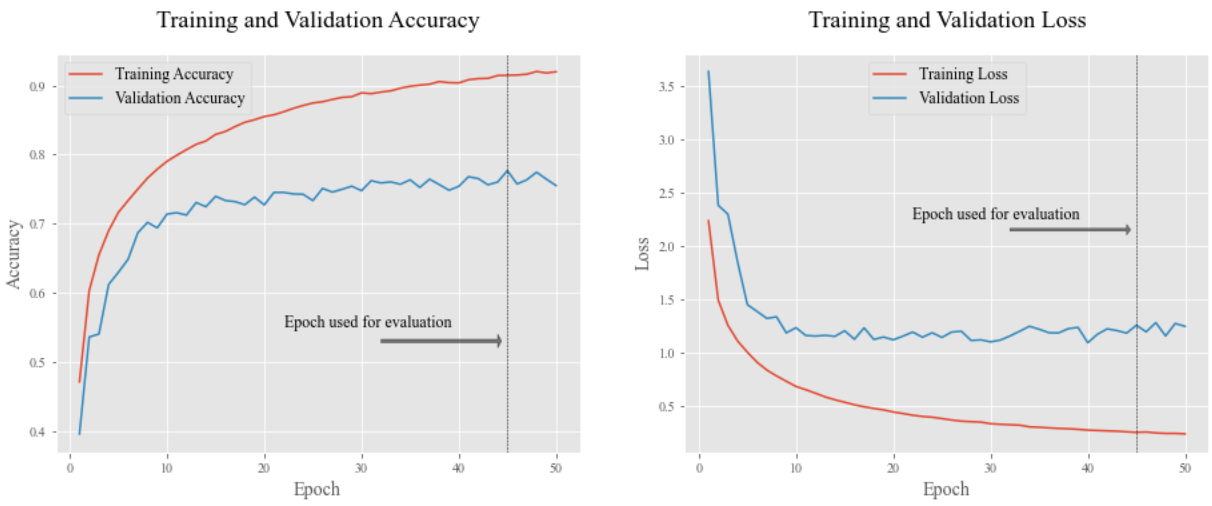


Figure 4: Family-level model training

As can be seen from the plots, despite the use of dropout and data augmentation, all of the models did display some level of overfitting, as indicated by the divergence of the training and validation accuracy. When training a neural network, deciding how many epochs to train for often involves striking a balance between increasing the overall accuracy and preventing this divergence. The benchmark model showed the best performance at epoch 48, with training accuracy of 0.92 and validation accuracy of 0.75. The family model achieved 0.91 training accuracy and 0.78 validation accuracy by epoch 45 (Table 1 contains full details of the best epoch for each model).

The three species-level models showed significantly greater validation accuracy fluctuation when compared to the benchmark and family-level models. This could indicate that the learning rate for the Adam optimiser was set too high for these particular models. Future work may wish to experiment with using a lower learning rate for the species-level models. However, for this analysis it was decided to keep the hyperparameters the same in order to aid comparison between models. All three species-level models achieved training accuracy of 0.99. Despite the fluctuations just mentioned, the Canidae, Felinae, and Sciuridae models also achieved validation accuracies of 0.89, 0.86 and 0.90 respectively.

Figure 5: Canidae model training

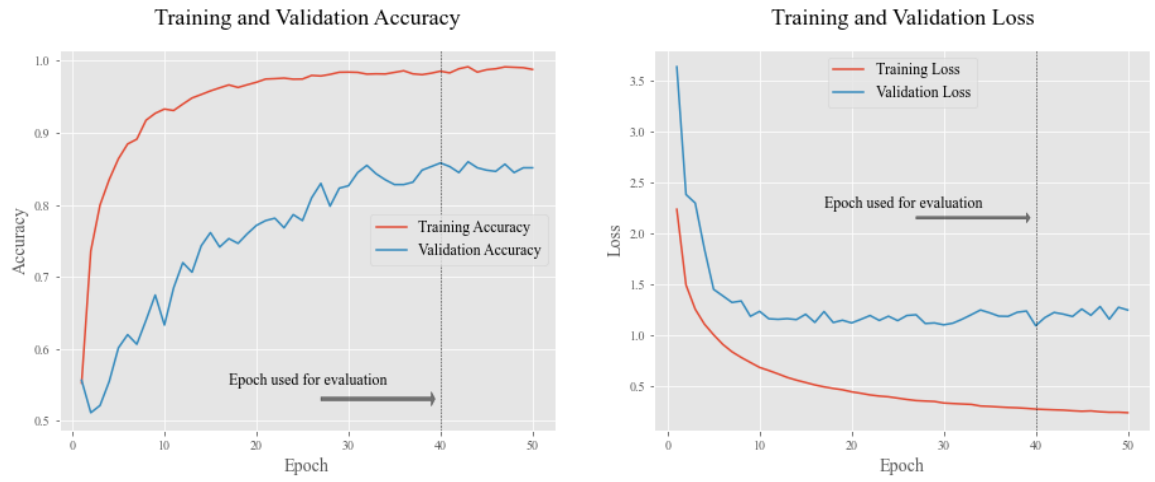


Figure 6: Felinae model training

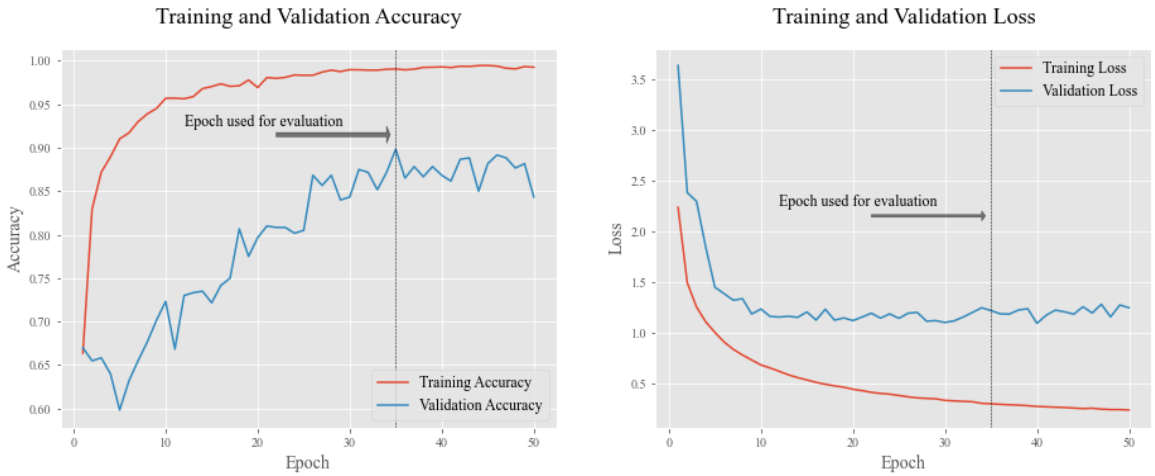


Figure 7: Sciuridae model training

Classifier	Epoch	Training Loss	Training Accuracy	Top 5 Training Accuracy	Validation Loss	Validation Accuracy	Validation Top 5 Accuracy
Benchmark	48	0.24	0.92	1.0	1.15	0.75	0.94
Family-Level	45	0.25	0.91	1.0	0.94	0.78	0.96
Canidae	43	0.02	0.99	1.0	0.41	0.89	1.00
Felinae	40	0.04	0.99	1.0	0.70	0.86	1.00
Sciuridae	35	0.03	0.99	1.0	0.38	0.90	1.00

Table 1: Optimum training and validation metrics per model

4.2. Test Results

The results of the testing phase are summarised in Table 2 and Figure 8. As can be seen, the benchmark model reported the lowest accuracy of all the models, both for top-1 accuracy and top-5 accuracy. The highest accuracy was reported by the three species-level models, each of which achieved 100% top-5 accuracy.

Classifier	Loss	Top-1-Accuracy	Top-5-Accuracy
Benchmark	1.24	0.74	0.93
Family-Level	0.93	0.77	0.96
Canidae	0.60	0.91	1.00
Felinae	0.50	0.88	1.00
Sciuridae	0.42	0.91	1.00

Table 2: Test results by model

These results seem encouraging. Accuracy of 91%, as recorded by the Canidae and Sciuridae models, is comparable to that achieved by Norouzzadeh *et al*. (2018), but with a much smaller dataset and a significantly simpler neural network architecture. However, it must be acknowledged that the species-level classifiers effectively had an easier task than both the benchmark model and the model developed by Norouzzadeh *et al* (and other such models referred to in section 2.2). These models each had just six categories to choose from, compared with 79 categories for the benchmark model and 48 categories for Norouzzadeh *et al*.

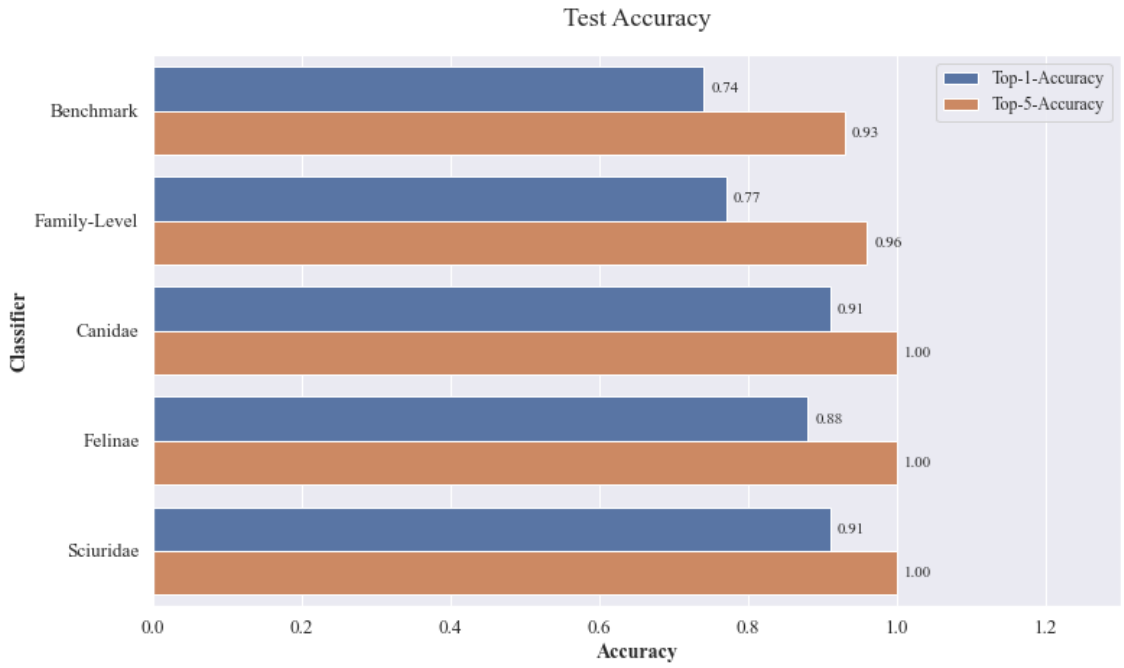


Figure 8: Test results bar plot by model

While this should not necessarily count against the species-level classifiers, as they will, by their very nature, have less categories to consider than a model which is built to classify all animals, it does mean that the family-level model should also be considered when calculating the true accuracy of any species-level classifier. Afterall, in order to be passed to the correct species-level classifier, an image must first be classified by family. The logical way to take this into account when analysing the results is to multiply the species-level accuracy by the family-level accuracy. This approach gives an overall accuracy of just 0.70 for the Canidae and Sciuridae models (0.77 x 0.91) , and 0.68 for the Felinae model (0.77 x 0.88), which is in fact lower than the benchmark model accuracy of 0.74.

As well as accuracy, each model was evaluated in terms of precision, recall, and F1 score. Precision is a measure of the likelihood of the model's prediction being correct, while recall records the likelihood that a particular class will be correctly predicted by the model. The F1 score is the harmonic mean of the precision and recall scores. Table 3 and Figure 9 show the weighted average of these three metrics (weighted by number of images) for each model.

Classifier	Precision	Recall	F1-Score	No. Images
Benchmark	0.75	0.74	0.74	7882.0
Family	0.78	0.77	0.77	7879.0
Canidae	0.91	0.91	0.91	597.0
Felinae	0.88	0.88	0.88	598.0
Sciuridae	0.91	0.91	0.91	595.0

Table 3: Weighted average precision, recall and F1 score for each model

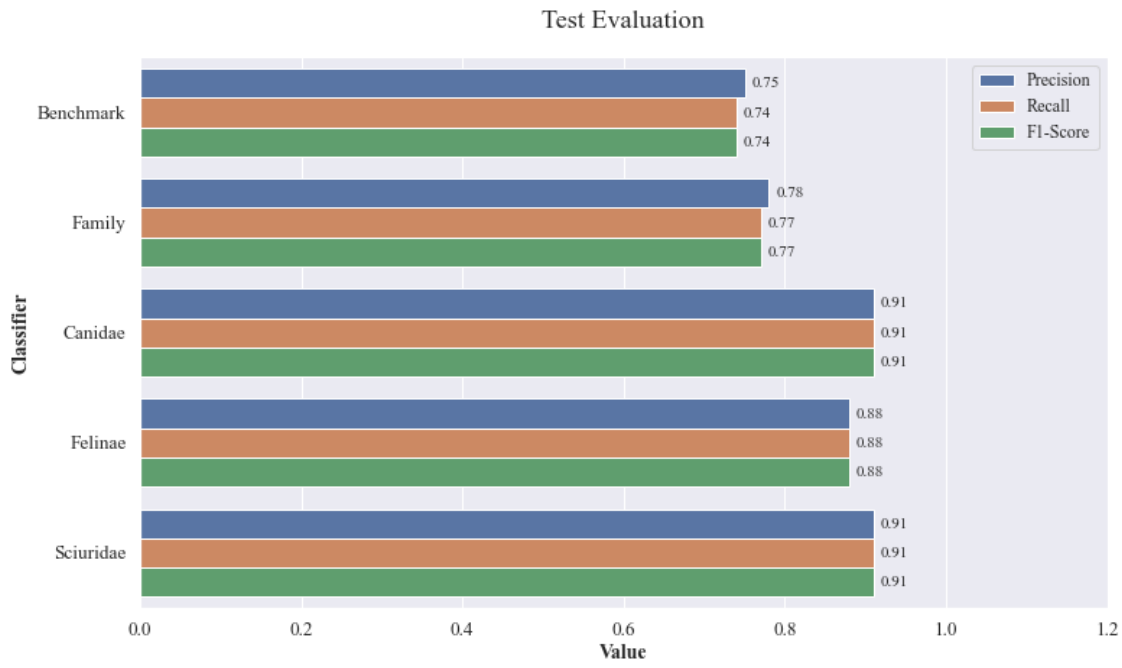


Figure 9: Weighted average precision, recall and F1 score for each model

In this case, the precision, recall and (therefore) F1 scores are all virtually identical for each model, and essentially equal to the overall accuracies. More interesting is to look at the precision and recall scores for individual classes. Table 4: Weighted average precision, recall and F1 score per class – CanidaeTable 4 shows the results for the Canidae classifier. Complete tables can be viewed in the Appendix.

The best results for the Canidae classifier appear to be for the class Culpeo, with 98% precision and 99% recall. We can be quite confident that the model will identify almost all instances of Culpeo, and any images it does label as Culpeo will indeed be so. Meanwhile, the class Fox shows the worst results,

with precision and recall of just 83% and 86% respectively. The model is likely to misclassify some true Fox images, while it will also classify some non-Fox images as Foxes.

Species	Precision	Recall	F1-Score	No. Images
Coyote	0.81	0.89	0.85	100.0
Culpeo	0.98	0.99	0.98	99.0
Dog	0.91	0.86	0.88	100.0
Fox	0.83	0.86	0.85	100.0
Jackal	0.94	0.96	0.95	100.0
Short-eared dog	0.99	0.88	0.93	98.0

Table 4: Weighted average precision, recall and F1 score per class – Canidae

Two other interesting classes observed in the table are Short-eared dog, and Coyote. The model shows a precision of 99% for the Short-eared dog class; if the model returns a label of short-eared dog for a particular image, we can be confident that this is correct. However, with a recall of just 88%, the model will miss a number of images of short-eared dogs. Conversely, the model will over-predict the Coyote class, such that it will miss fewer images, but will also return a number of supposed images of Coyotes that will actually be some other animal (recall 89%, precision 81%). The purpose of this project was to build general-purpose models, and therefore both a high precision and recall are desired. However, there may be specific-use cases where differences between precision and recall are acceptable or even encouraged. For example, if a conservationist wanted to ensure that they accessed every possible image of Coyotes, they would likely accept the trade-off of a higher recall versus lower precision, with the consequence that some of the images would in fact be of a different animal.

Similar results are observed in the other models. For example, the Felinae model identifies images of Cheetahs reasonably well (precision 92%, recall 91%), but over-predicts Cougars (82% / 89%) and under-predicts Ocelots (92% / 74%). The Sciuridae model performs very well on the American Red Squirrel (99% / 100%) but comparatively poorly on the Eastern Grey Squirrel (76% / 89%).

To get an idea of what mistakes a model is making, confusion matrices can be produced. Matrices for each classifier are found in the Appendix. Here, Figure 10 shows the predictions made by the Felinae classifier. There is clearly some logic to at least some of the mistakes. For example, three images of Cats have been incorrectly classified as Bobcats, and three as Cougars, but none have been classified as Cheetahs, Ocelots or Servals. These latter three species tend to have quite strong markings, with Cheetahs and Servals in particular having similarly black-spotted coats, whereas cougars tend to have plainer coats

and Bobcats more mottled coats, more similar to those of a typical (house) cat. The fact that the model is likely picking up on these factors can also be seen by looking at the classifications of actual Cheetah and Serval images. Eight out of nine incorrectly classified Cheetah images were classified as Servals, while eight out of eight incorrectly classified Serval images were classified as Cheetahs.

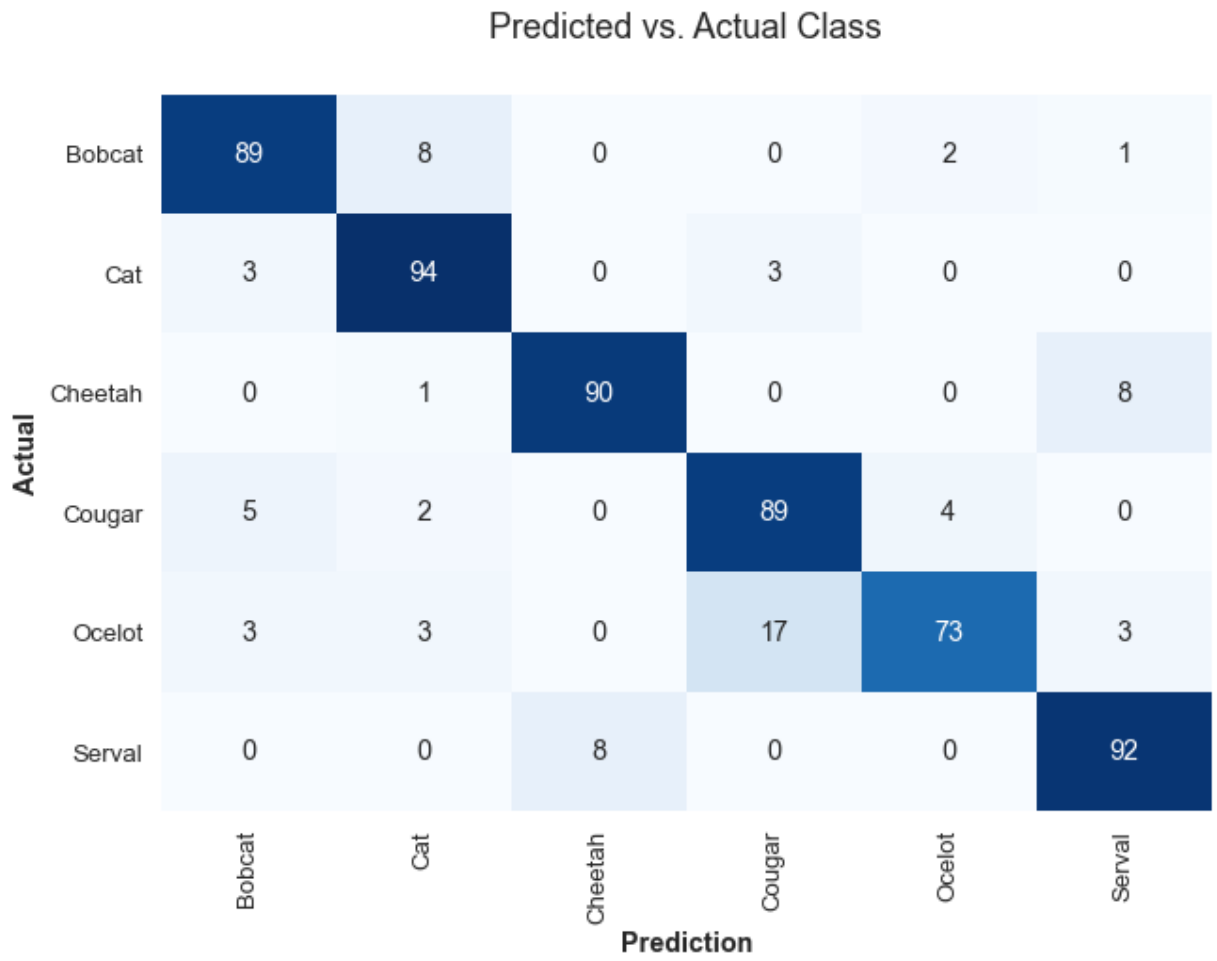


Figure 10: Felinae test images confusion matrix

Figure 11 shows four examples of misclassified images (one for each of the benchmark, Canidae, Felinae and Sciuridae models), while Figure 12 shows four correctly classified images. From the incorrect set, the image from the benchmark model and the image from the Felinae model are somewhat understandable, with just the Mule Deer's leg visible and seemingly nothing visible of the Serval. The other two images are more surprising, given that the Fox and California Ground Squirrel and quite prominent in the images. The four correctly classified images show that the models are sometimes capable of identifying animals even when just a portion of the animal is visible (Hartebeest, Bobcat) or when the image is blurry due to movement (Eastern Grey Squirrel).

Figure 11: Misclassified images. From top left to bottom right: (1) Benchmark Model. Predicted: American Red Squirrel. Actual: Mule Deer. (2) Canidae Model. Predicted: Coyote. Actual: Fox. (3) Felinae Model. Predicted: Bobcat. Actual: Serval. (4) Sciuridae Model. Predicted: Chipmunk. Actual: California Ground Squirrel.

Figure 12: Correctly classified images. From top left to bottom right: (1) Benchmark Model: Hartebeest. (2) Canidae Model: Short-Eared Dog. (3) Felinae Model: Bobcat. (4) Sciuridae Model: Eastern Grey Squirrel.

4.3. Out-of-Sample Results

One of the main aims of the project was to determine whether family specific neural networks would generalise better to new locations. To evaluate this, the benchmark model and three species-level models were also tested using out-of-sample data (Table 5, Table 6, Figure 13, Figure 14).

While the Canidae and Felinae models both outperformed the benchmark model, the Sciuridae model showed exceptionally poor performance, with accuracy of just 1%.

Classifier	Loss	Top-1-Accuracy	Top-5-Accuracy
Benchmark	7.914284	0.175104	0.427040
Canidae	1.547057	0.662675	1.000000
Felinae	2.277555	0.532468	1.000000
Sciuridae	14.581409	0.010955	0.884194

Table 5: Out-of-sample test results by model

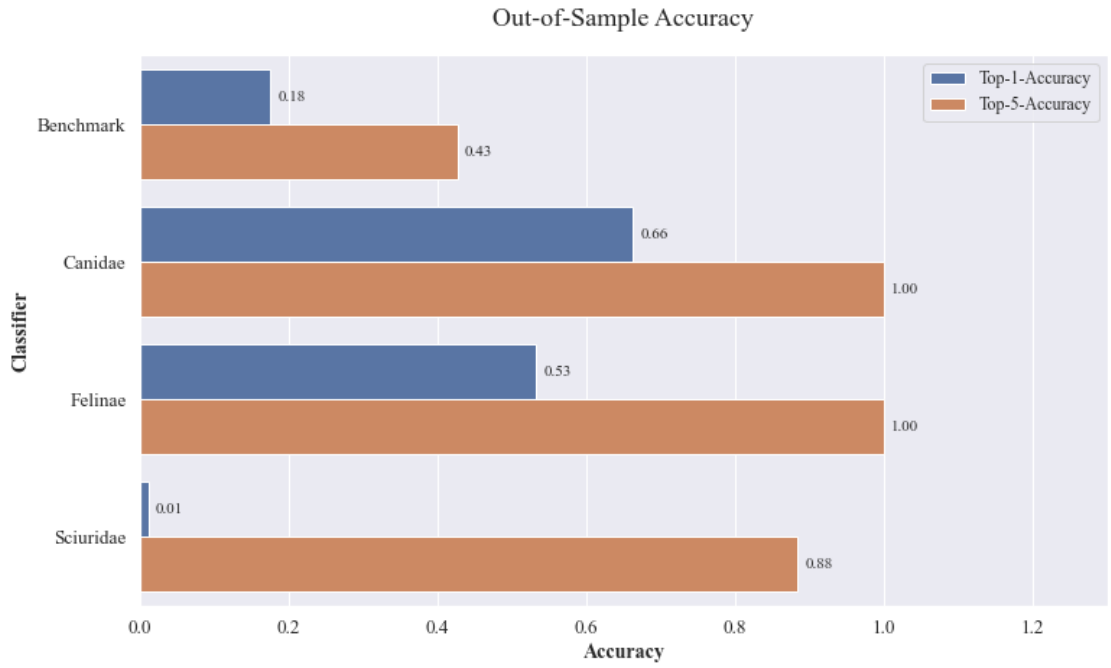


Figure 13: Out-of-sample results plot by model

Classifier	Precision	Recall	F1-Score	No. Images
Benchmark	0.57	0.18	0.25	1679.0
Canidae	1.00	0.66	0.80	501.0
Felinae	1.00	0.53	0.69	539.0
Sciuridae	1.00	0.01	0.02	639.0

Table 6: Out-of-sample weighted average precision, recall and F1 score by model

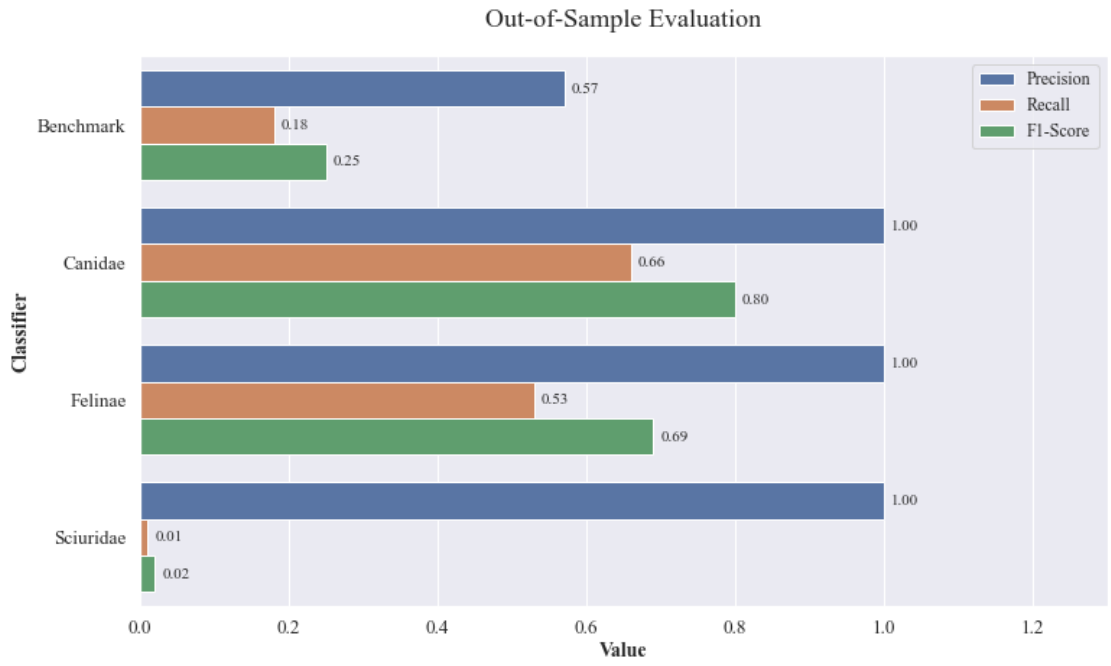


Figure 14: Out-of-sample weighted average precision, recall and F1 score by model

The confusion matrix for the Sciuridae out-of-sample classifications (Figure 15) reveals that the model almost exclusively predicted just two (incorrect) classes for each of the out-of-sample images: Carruthers's Mountain Squirrel and Southern Amazon Red Squirrel. The Southern Amazon Red Squirrel prediction would appear reasonable, given the similarity in appearance between this and the American Red Squirrel (the correct label). The prediction of Carruthers's Mountain Squirrel is less explainable, as these two species are quite distinctive from one another, particularly in colour.

The reported performance of a machine learning classifier can occasionally be deceptive. If the test set used to evaluate the model is unbalanced, the model can simply return the majority class(es) as its prediction for every test image, and still achieve a high accuracy, However, this cannot explain the poor performance of the Sciuridae model in this study. It is clear from the confusion matrix for the Sciuridae test results (see Appendix) that the test dataset was well balanced, and that the model returned a set of predictions which was relatively evenly distributed across all of the classes.

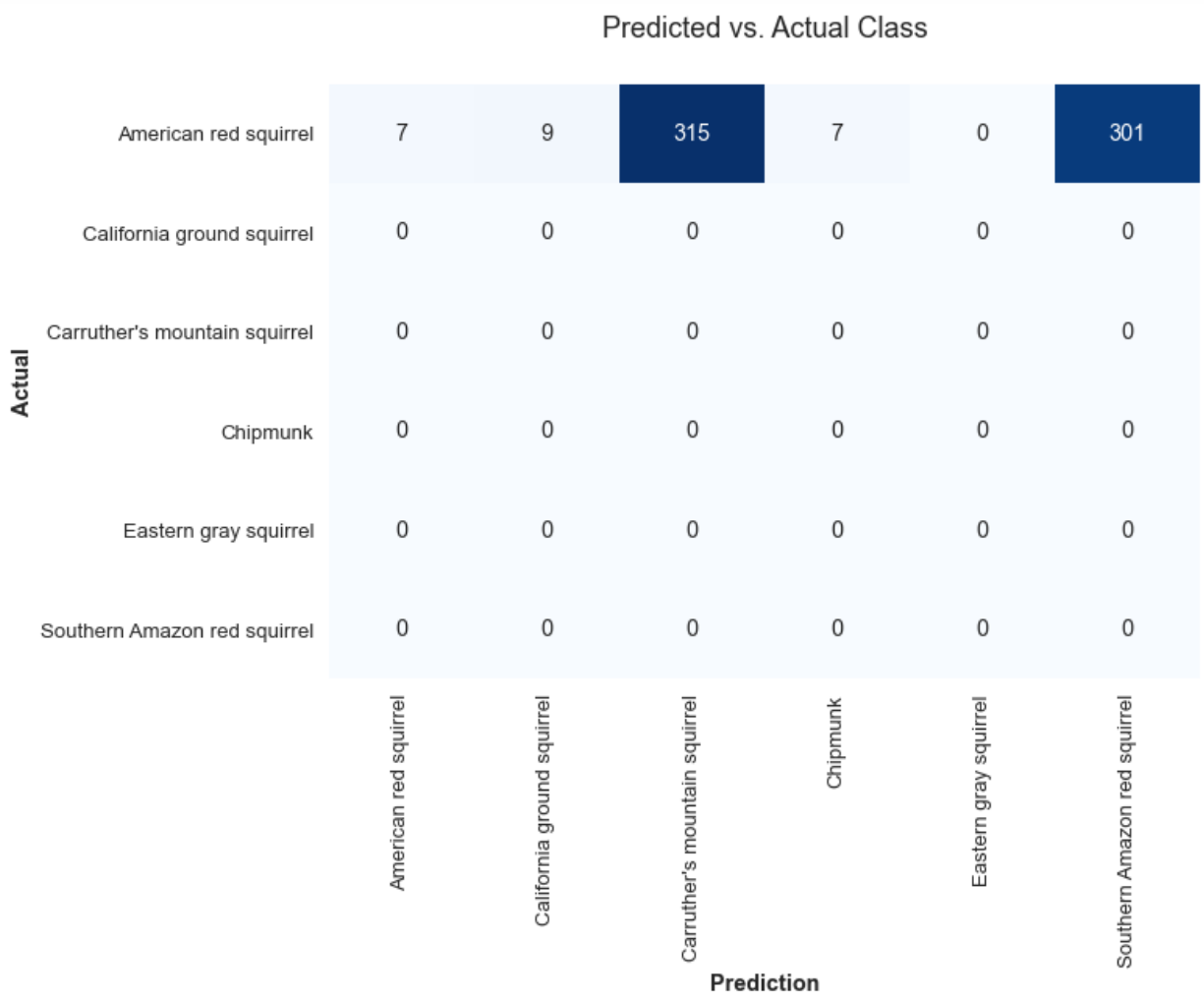


Figure 15: Sciuridae out-of-sample images confusion matrix

Given that colour is the most distinctive difference between the two species, another possible explanation for the Carruthers's Mountain Squirrel misclassifications is the lack of colour in some of the out-of-sample images, as seen in the bottom right image in Figure 16. However, Figure 17, which shows correctly classified images, also shows a black and white image which was correctly identified by the Sciuridae model, so this also fails to explain the poor performance.

As discussed in section 2.2, Tabak *et al*. (2020) tested their model on three different out-of-sample datasets. They reported impressive results on the Canada dataset (91% accuracy) but poor results on the Missouri and Caltech datasets (36% and 56%) respectively. In comparison, the Canidae and Felinae models developed here achieved 66% and 53% accuracy on the Missouri dataset. Again though, the smaller number of classes for the species-level models compared with the generic model used by Tabak *et al*. must be taken into account when considering whether this is a true improvement, along with the extra step (and any associated misclassifications) involved in first classifying the images by family.

Figure 16: Misclassified out-of-sample images. From top left to bottom right: (1) Benchmark Model. Predicted: Amazon Red Squirrel. Actual: American Red Squirrel. (2) Felinae Model. Predicted: Cat. Actual: Ocelot. (3) Canidae Model. Predicted: Jackal. Actual: Fox. (4) Sciuridae Model. Predicted: Carruthers's Mountain Squirrel. Actual: American Red Squirrel.

Figure 17: Correctly classified out-of-sample images. From top left to bottom right: (1) Benchmark Model: Ocelot. (2) Canidae Model: Fox. (3) Felinae Model: Ocelot. (4) Sciuridae Model: American Red Squirrel.

5. CONCLUSIONS

While on the surface the three species-level models appear to show a good improvement over the benchmark model, the reality is perhaps not so straightforward. In order to get to the species-level results, any images must first be passed through the family-level classifier. Taking this classifier's accuracy into account, the true accuracy of the entire system is likely lower than that of the benchmark model. As a result, it is tempting to conclude that family specific neural networks do not outperform generic CNN animal classifiers.

Similarly, the out-of-sample results present a mixed picture. The results for the Canidae and Felinae classifiers suggest that this model structure is an improvement when compared with the results of Tabak *et al*. (2020) on the same Missouri dataset. However, the results of the Sciuridae model were

extremely poor, with the model completely unusable on any out-of-sample data. This suggests that, at least for some families of animal, family specific models do not generalise to new locations any more than generic models do.

However, when considering these conclusions, the particular characteristics of the study should be acknowledged, especially regarding the limitations on computing power. Not only did this restrict the neural network architecture that could be employed, and limit the amount of hyperparameter tuning that could be performed, it also severely limited the number of images that could be used for training. While Schneider *et al*. (2020) proposed a figure of 1,000 images per class to achieve strong results, they also acknowledged that such a figure may not generalise to all datasets. When looking closely at that particular study, it can be seen that the majority of the classes analysed were quite distinct. In such a case, 1,000 images may well be sufficient for the model to learn the differences between, for example, a Grizzly Bear and a small bird. However, it is likely the case that here, such a small number of images was simply insufficient for the model to learn the minute differences between species as similar as the Amazon Red Squirrel and the American Red Squirrel. Indeed, Norouzzadeh *et al*. (2018) used over 3.2 million images to classify just 48 different species, an average of over 66,000 images per species. It would, therefore, be interesting to repeat the study with greater computing resources and a larger training dataset, comparable in size to that of Norouzzadeh *et al*. (2018). This may well result in an improvement in the performance of the final species-level classifiers.

6. FUTURE WORK

As just discussed, repeating the study with a significantly larger dataset would be an interesting approach for future study, as would experimenting with larger, more complex neural networks.

Another potential area to explore is the use of an object detector to first identify and remove any empty images (as per Norouzzadeh *et al*. 2018). This could be used before the images are passed to the family-level classifier, allowing this model to focus on learning the features of the animals present, rather than learning any details of empty images. The downside to this approach is the extra computational step involved in developing and applying the object detector.

To address the generalisation problem, a combination approach could be used by building classifiers which are both location-specific and family-specific. This would reduce the number of classes that any single model was required to learn, and may help to eliminate misclassifications between similar looking species. For example, the Sciuridae model developed for this study had a problem distinguishing

between the American Red Squirrel and the Amazon Red Squirrel. However, if the model was designed solely to be used in a Camera Trap located in Missouri, there would be no need for this model to be able to classify an Amazon Red Squirrel. The problem with this approach is that alluded to in section 2.2, of having to develop numerous models for each location. However, as with the object detector approach, if extremely high accuracy is the most important factor, then such models may well be worth investigating.

Word Count: 9,343

REFERENCES

Anton, V. *et al.* (2018) 'Monitoring the mammalian fauna of urban areas using remote cameras and citizen science', *Journal of Urban Ecology*, 4(1). doi: 10.1093/jue/juy002.

Beery, S., van Horn, G. and Perona, P. (2018) 'Recognition in Terra Incognita', *arXiv:1807.04975 [cs, q-bio]*. Available at: http://arxiv.org/abs/1807.04975 (Accessed: 22 May 2021).

'Camera traps and science - how did we get here?' (2014) *NatureSpy*, 31 March. Available at: https://www.naturespy.org/2014/03/camera-traps-science/ (Accessed: 4 August 2021).

Cloud Computing, Hosting Services, and APIs (no date) *Google Cloud*. Available at: https://cloud.google.com/gcp (Accessed: 20 June 2021).

Cloud Storage for Work and Home – Google Drive (no date). Available at: https://www.google.com/intl/en-GB/drive/ (Accessed: 31 August 2021).

Exploring dropouts (no date) *Coursera*. Available at: https://www.coursera.org/learn/convolutional-neural-networks-tensorflow/home/welcome (Accessed: 6 August 2021).

Falzon, G. *et al.* (2019) 'ClassifyMe: A Field-Scouting Software for the Identification of Wildlife in Camera Trap Images', *Animals*, 10(1), p. 58. doi: 10.3390/ani10010058.

Google Colaboratory (no date). Available at: https://colab.research.google.com/notebooks/intro.ipynb (Accessed: 31 August 2021).

Greenberg, S. (2020) 'Automated Image Recognition for Wildlife Camera Traps: Making it Work for You', *Technical report, Prism University of Calgary's Digital Repository http://hdl.handle.net/1880/112416*. doi: 10.11575/PRISM/38103.

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition [Book] (2019). Available at: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ (Accessed: 5 August 2021).

Hinton, G. (no date) 'rmsprop: Divide the gradient by a running average of its recent magnitude'. Available at: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (Accessed: 6 August 2021).

ImageNet (2021). Available at: https://www.image-net.org/ (Accessed: 23 May 2021).

Island Conservation (no date) *Island Conservation*. Available at: https://www.islandconservation.org (Accessed: 4 August 2021).

Keras documentation: Keras Applications (no date). Available at: https://keras.io/api/applications/ (Accessed: 4 August 2021).

Keras documentation: MobileNet and MobileNetV2 (no date). Available at: https://keras.io/api/applications/mobilenet/ (Accessed: 28 August 2021).

Keras documentation: Transfer learning & fine-tuning (2020). Available at: https://keras.io/guides/transfer_learning/ (Accessed: 5 August 2021).

Kingma, D. P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 [cs]*. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 31 August 2021).

Lecun, Y. *et al.* (1998) 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE*, 86(11), pp. 2278–2324. doi: 10.1109/5.726791.

LILA BC (Labeled Image Library of Alexandria: Biology and Conservation) (no date) *LILA BC*. Available at: http://lila.science/ (Accessed: 21 June 2021).

Nielsen, M. A. (2015) 'Neural Networks and Deep Learning'. Available at: http://neuralnetworksanddeeplearning.com (Accessed: 5 August 2021).

Norouzzadeh, M. S. *et al.* (2018) 'Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning', *Proceedings of the National Academy of Sciences*, 115(25), pp. E5716–E5725. doi: 10.1073/pnas.1719367115.

Norouzzadeh, M. S. *et al.* (2019) 'A deep active learning system for species identification and counting in camera trap images', *arXiv:1910.09716 [cs, eess, stat]*. Available at: http://arxiv.org/abs/1910.09716 (Accessed: 23 May 2021).

Overview of neuron structure and function (article) | Khan Academy (no date). Available at: https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function (Accessed: 5 August 2021).

Pokhrel, S. (2019) *Generate More Training Data When You Don't Have Enough*, *Medium*. Available at: https://towardsdatascience.com/generate-more-training-data-when-you-dont-have-enough-db7d8dcde90e (Accessed: 5 August 2021).

Rosebrock, A. (2016) ‘A simple neural network with Python and Keras’, *PyImageSearch*, 26 September. Available at: https://www.pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-keras/ (Accessed: 5 August 2021).

Rosebrock, A. (2019) ‘Keras: Starting, stopping, and resuming training’, *PyImageSearch*, 23 September. Available at: https://www.pyimagesearch.com/2019/09/23/keras-starting-stopping-and-resuming-training/ (Accessed: 28 August 2021).

Sandler, M. *et al.* (2019) ‘MobileNetV2: Inverted Residuals and Linear Bottlenecks’, *arXiv:1801.04381 [cs]*. Available at: http://arxiv.org/abs/1801.04381 (Accessed: 8 August 2021).

Saving Wildlife and Wild Places - WCS.org (no date). Available at: https://www.wcs.org/ (Accessed: 4 August 2021).

Schneider, S. *et al.* (2020) ‘Three critical factors affecting automated image species recognition performance for camera traps’, *Ecology and Evolution*, 10(7), pp. 3503–3517. doi: 10.1002/ece3.6147.

Shahinfar, S., Meek, P. and Falzon, G. (2020) ‘“How many images do I need?” Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring’, *Ecological Informatics*, 57, p. 101085. doi: 10.1016/j.ecoinf.2020.101085.

Splitting into train, dev and test sets (no date). Available at: https://cs230.stanford.edu/blog/split/ (Accessed: 5 August 2021).

Swanson, A. *et al.* (2015) ‘Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna’, *Scientific Data*, 2(1), p. 150026. doi: 10.1038/sdata.2015.26.

Szegedy, C. *et al.* (2015) ‘Rethinking the Inception Architecture for Computer Vision’, *arXiv:1512.00567 [cs]*. Available at: http://arxiv.org/abs/1512.00567 (Accessed: 4 August 2021).

Szegedy, C. *et al.* (2016) ‘Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning’, *arXiv:1602.07261 [cs]*. Available at: http://arxiv.org/abs/1602.07261 (Accessed: 4 August 2021).

Tabak, M. A. *et al.* (2019) ‘Machine learning to classify animal species in camera trap images: Applications in ecology’, *Methods in Ecology and Evolution*, 10(4), pp. 585–590. doi: 10.1111/2041-210X.13120.

Tabak, M. A. *et al.* (2020) 'Improving the accessibility and transferability of machine learning algorithms for identification of animals in camera trap images: MLWIC2', *Ecology and Evolution*, 10(19), pp. 10374–10383. doi: 10.1002/ece3.6692.

tf.keras.applications.mobilenet_v2.preprocess_input (no date) *TensorFlow*. Available at: https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet_v2/preprocess_input (Accessed: 28 August 2021).

Wearn, O. and Glover-Kapfer, P. (2017) *Camera-trapping for conservation: a guide to best-practices*. doi: 10.13140/RG.2.2.23409.17767.

Whytock, R. C. *et al.* (2021) 'Robust ecological analysis of camera trap data labelled by a machine learning model', *Methods in Ecology and Evolution*. Edited by C. Alberto Silva, pp. 2041–210X.13576. doi: 10.1111/2041-210X.13576.

'Wikipedia, the free encyclopedia' (2021) *Wikipedia, the free encyclopedia*. Available at: https://en.wikipedia.org/w/index.php?title=Main_Page&oldid=1004593520 (Accessed: 4 August 2021).

Willi, M. *et al.* (2019) 'Identifying animal species in camera trap images using deep learning and citizen science', *Methods in Ecology and Evolution*. Edited by O. Gaggiotti, 10(1), pp. 80–91. doi: 10.1111/2041-210X.13099.

Yousif, H., Kays, R. and He, Z. (2019) 'Dynamic programming selection of object proposals for sequence-level animal species classification in the wild.', *IEEE Transactions on Circuits and Systems for Video Technology*.

Zhang, Z. *et al.* (2016) 'Animal Detection From Highly Cluttered Natural Scenes Using Spatiotemporal Object Region Proposals and Patch Verification', *IEEE Transactions on Multimedia*. doi: 10.1109/TMM.2016.2594138.

Zooniverse (no date). Available at: https://www.zooniverse.org/organizations/meredithspalmer/snapshot-safari (Accessed: 4 August 2021).

APPENDIX

i. Dataset Details

- *Benchmark Model*

Species	Training Images	Validation Images	Test Images	Out-of-Sample Images
African wild ass	800	100	100	0
Alpaca	800	100	100	0
American red squirrel	800	100	100	639
Baboon	799	100	100	0
Blesbok	800	100	100	0
Bobcat	800	100	100	0
Brocket deer	800	100	100	0
Buffalo	800	100	100	0
Bushpig	800	100	100	0
California ground squirrel	800	100	100	0
Cape bushbuck	800	100	100	0
Carruther's mountain squirrel	800	100	100	0
Cat	796	99	99	0
Cattle	784	98	98	0
Cheetah	798	100	100	0
Chipmunk	800	100	100	0
Coati	724	90	91	0
Cougar	800	100	100	0
Coyote	799	99	101	0
Crax (Curassow)	800	100	100	0
Culpeo	800	100	100	0
Dik-dik	800	100	100	0
Dog	780	97	98	0
Dromedary	800	100	100	0
East African oryx	800	100	100	0
Eastern gray squirrel	800	100	100	0
Eland	798	99	101	0
Elk	800	100	100	0
Empty	800	100	100	0
Fox	800	100	100	501
Gambian pouched rat	800	100	100	0
Gemsbok	797	99	101	0
Goat	791	98	100	0

Grant's gazelle	799	100	100	0
Great argus	800	100	100	0
Grey-winged trumpeter	799	100	100	0
Handsome spurfowl	800	100	100	0
Hare	800	100	100	0
Hartebeest	800	100	100	0
Horse	800	100	100	0
Jackal	797	99	101	0
Jaguar	800	100	100	0
Kudu	797	99	101	0
L'Hoest's monkey	800	100	100	0
Leopard	800	100	100	0
Macaque	800	100	100	0
Marten	800	100	100	0
Mitu (Curassow)	800	100	100	0
Moose	800	100	100	0
Mule deer	800	100	100	0
Muntjac	800	100	100	0
Nesomys	800	100	100	0
Ocelot	800	100	100	539
Pale-winged trumpeter	800	100	100	0
Penelope (Guan)	800	100	100	0
Pig	783	98	98	0
Rabbit	792	99	100	0
Raccoon	799	99	101	0
Red deer	800	100	100	0
Red-legged partridge	800	100	100	0
Reedbuck	800	100	100	0
Serval	800	100	100	0
Sheep	800	100	100	0
Short-eared dog	800	100	100	0
Southern Amazon red squirrel	800	100	100	0
Springbok	796	99	100	0
Squirrel	800	100	100	0
Steenbok	800	100	100	0
Suni	800	100	100	0
Tayra	800	100	100	0
Topi	799	99	101	0
Turkey	727	90	92	0
Vervet monkey	797	99	101	0
Warthog	796	99	99	0
Waterbuck	794	99	100	0
White-tailed deer	800	100	100	0

Wild boar	800	100	100	0
Wildebeest	788	98	99	0
Zebra	793	99	100	0

- *Family Model*

Family	**Training Images**	**Validation Images**	**Test Images**
Alcelaphinae	3187	398	399
Antilopinae	3995	499	500
Bovinae	3980	497	498
Camelidae	1600	200	200
Canidae	4767	595	597
Capreolinae	3200	400	400
Caprinae	1591	198	200
Cercopithecidae	3196	399	401
Cervinae	2400	300	300
Cracidae	2409	301	302
Equidae	2393	299	300
Felinae	4793	599	600
Guloninae	1600	200	200
Hippotraginae	1597	199	201
Leporidae	1592	199	200
Nesomyidae	1600	200	200
None	800	100	100
Pantherinae	1600	200	200
Phasianidae	3127	390	392
Procyonidae	1523	190	191
Psophiidae	1600	200	200
Reduncinae	1594	199	200
Sciuridae	5600	700	700
Suidae	3178	397	398

- *Canidae Model*

Species	Training Images	Validation Images	Test Images	Out-of-Sample Images
Coyote	898	101	100	0
Culpeo	900	100	99	0
Dog	877	98	100	0
Fox	900	100	100	501
Jackal	896	101	100	0
Short-eared dog	900	100	98	0

- *Felinae Model*

Species	Training Images	Validation Images	Test Images	Out-of-Sample Images
Bobcat	900	100	100	0
Cat	895	99	100	0
Cheetah	898	100	99	0
Cougar	900	100	100	0
Ocelot	900	100	99	539
Serval	900	100	100	0

- *Sciuridae Model*

Species	Training Images	Validation Images	Test Images	Out-of-Sample Images
American red squirrel	900	100	100	639
California ground squirrel	900	100	100	0
Carruther's mountain squirrel	900	100	98	0
Chipmunk	900	100	100	0
Eastern gray squirrel	900	100	100	0
Southern Amazon red squirrel	900	100	97	0

ii. Code

- *Data Pre-Processing*

```
# Import necessary libraries and modules

import pandas as pd
import numpy as np
import json
import os
from random import sample
import shutil
import glob
from google.cloud import storage

# Increase the display size in order to prevent truncation later on

pd.set_option('display.max_rows', 200)

# Function to read in json metadata for the chosen datasets as data frame

def metadata_to_df(file, dataset):
    with open(file) as json_data:
        json_dict = json.load(json_data)
    df_images = pd.DataFrame(json_dict['images'])
    df_images = df_images.set_index('id')
    df_annot = pd.DataFrame(json_dict['annotations'])
    df_annot = df_annot.set_index('image_id')
    df_annot = df_annot.drop(['id'], axis=1)
    df_annot.index.name = 'id'
    df = pd.merge(df_annot, df_images,  how='left', on='id')
    df = df[['category_id', 'file_name']]
    df_cats = pd.DataFrame(json_dict['categories'])
    for col in ['count', 'species', 'genus', 'family',
            'ord', 'class', 'common name']:
        if col in df_cats.columns:
            df_cats = df_cats.drop(col, axis=1)
    df_cats = df_cats.rename(columns={'id':'category_id'})
    df = df.reset_index().merge(df_cats, how="left").set_index('id')
    df['name'].value_counts()
    df["Dataset"] = dataset
    return df

# Use function to create dataframe for each dataset

caltech_df = metadata_to_df('caltech_images_20210113.json', 'Caltech')
island_df = metadata_to_df('island_conservation.json', 'Island')
missouri_df = metadata_to_df("missouri_camera_traps_set1.json", 'Missouri')
camdeboo_df = metadata_to_df("SnapshotCamdeboo_S1_v1.0.json", 'Camdeboo')
wcs_df = metadata_to_df("wcs_camera_traps.json", "WCS")
ena24_df = metadata_to_df("ena24.json", 'ENA24')
wellington_df = metadata_to_df("wellington_camera_traps.json", 'Wellington')
karoo_df = metadata_to_df('SnapshotKaroo_S1_v1.0.json', "Karoo")
kgalagai_df = metadata_to_df('SnapshotKgalagai_S1_v1.0.json', "Kgalagai")
```

```
enonkishu_df = metadata_to_df('SnapshotEnonkishu_S1_v1.0.json', "Enonkishu")
mountain_zebra_df = metadata_to_df('SnapshotMountainZebra_S1_v1.0.json',
                                   'Mountain Zebra')
kruger_df = metadata_to_df('SnapshotKruger_S1_v1.0.json', 'Kruger')
nacti_df = metadata_to_df('nacti_metadata.json', 'NACTI')
serengeti_df = metadata_to_df('SnapshotSerengeti_S1-11_v2.1.json',
                              'Serengeti')

# Combine resultant dataframes

complete_df = pd.concat([caltech_df, island_df, missouri_df,
                         camdeboo_df, wcs_df, ena24_df, wellington_df,
                         karoo_df, kgalagai_df, enonkishu_df,
                         mountain_zebra_df, kruger_df, nacti_df,
                         serengeti_df])

# Load the corrected species/subfamilies/families information

names_df = pd.read_csv("names_species_families.csv")

# Rename column to match with complete_df

names_df = names_df.rename(columns={'Label':'name'})

# Merge data frames

complete_df = complete_df.reset_index().merge(names_df,  how='left',
on='name').set_index('id')

# Check how many images, species, subfamilies and families are contained in
dataframe

complete_df.nunique()

# Check how many species have at least 1,000 images

species_counts = complete_df['Species'].value_counts()
highest_species_counts = species_counts[species_counts >= 1000]
len(highest_species_counts)

# Drop species that are either too vague to categorise (e.g. 'Bird') or not
animals ('human', 'motorcycle', etc.)

highest_species_counts = highest_species_counts.drop(labels=['Bird', 'Car',
'Deer',
                             'Domestic animal', 'Human', 'Motorcycle',
'Petrel',
                             'Rat', 'Rodent', 'Unknown'])

# Create new dataframe of the species that will be classified

subset_df =
complete_df[complete_df['Species'].isin(highest_species_counts.index)]

# Create separate dataframes for each dataset. These will be used to select
the images
```

```
kgalagai_df = subset_df[subset_df["Dataset"] == "Kgalagai"]
caltech_df = subset_df[subset_df["Dataset"] == "Caltech"]
island_df = subset_df[subset_df["Dataset"] == "Island"]
missouri_df = subset_df[subset_df["Dataset"] == "Missouri"]
camdeboo_df = subset_df[subset_df["Dataset"] == "Camdeboo"]
wcs_df = subset_df[subset_df["Dataset"] == "WCS"]
ena24_df = subset_df[subset_df["Dataset"] == "ENA24"]
wellington_df = subset_df[subset_df["Dataset"] == "Wellington"]
karoo_df = subset_df[subset_df["Dataset"] == "Karoo"]
enonkishu_df = subset_df[subset_df["Dataset"] == "Enonkishu"]
mountain_zebra_df = subset_df[subset_df["Dataset"] == "Mountain Zebra"]
kruger_df = subset_df[subset_df["Dataset"] == "Kruger"]
nacti_df = subset_df[subset_df["Dataset"] == "NACTI"]
serengeti_df = subset_df[subset_df["Dataset"] == "Serengeti"]

# Create a dataframe showing number of species images in each dataset

species_per_dataset = subset_df.groupby(['Species', 'Dataset']).size()
species_per_dataset = pd.DataFrame(species_per_dataset)
species_per_dataset.reset_index(inplace=True)
species_per_dataset.columns = ['Species', 'Dataset', 'Count']

# Remove Missouri from dataframe because this will be used as out-of-sample
data

species_per_dataset = species_per_dataset[~(species_per_dataset['Dataset'] ==
'Missouri')]

# Add a column which will indicate how many images to use

species_per_dataset['Selection'] = species_per_dataset['Count']

# Change the selection value to maximum of 1,000 divided by the number of
datasets containing that species

list_of_species = list(species_per_dataset['Species'].unique())
for species in list_of_species:
    length = len(species_per_dataset[species_per_dataset['Species'] ==
species])
    index =
species_per_dataset[species_per_dataset['Species']==species].index
    for n in index:
        if species_per_dataset['Selection'][n] <= int(1000/length):
            continue
        else:
            species_per_dataset.loc[n,'Selection'] = int(1000/length)

# Add a column showing the total images selected so far per species, plus
column for images left to choose

grouped = species_per_dataset.groupby(['Species']).sum()
for species in grouped.index:
    species_per_dataset.loc[species_per_dataset['Species'] == species,
'Total'] =  grouped.loc[species,'Selection']
species_per_dataset['Remaining'] = species_per_dataset['Count'] -
species_per_dataset['Selection']
```

```
# Get new list based on species where less than 1000 images have been
selected
new_list_of_species = list(species_per_dataset[species_per_dataset["Total"]
!= 1000]['Species'].unique())
# Repeat above process to select more images
for species in new_list_of_species:
    length = sum(species_per_dataset[species_per_dataset["Remaining"] !=
0]['Species'] == species)
    index = species_per_dataset[species_per_dataset["Remaining"] !=
0][species_per_dataset[species_per_dataset["Remaining"] != 0]['Species'] ==
species].index
    for n in index:
        species_per_dataset.loc[n,'Selection'] += min(int((1000 -
species_per_dataset.loc[n, 'Total'])/length),
species_per_dataset.loc[n,'Remaining'])
for species in list_of_species:
    species_per_dataset.loc[species_per_dataset['Species'] == species,
'Total'] =  sum(species_per_dataset.loc[species_per_dataset['Species'] ==
species, 'Selection'])
    species_per_dataset.loc[species_per_dataset["Species"] == species,
"Remaining"] = species_per_dataset.loc[species_per_dataset["Species"] ==
species, "Count"] - species_per_dataset.loc[species_per_dataset["Species"] ==
species,"Selection"]

# Repeat entire process
new_list_of_species = list(species_per_dataset[species_per_dataset["Total"]
!= 1000]['Species'].unique())
for species in new_list_of_species:
    length = sum(species_per_dataset[species_per_dataset["Remaining"] !=
0]['Species'] == species)
    index = species_per_dataset[species_per_dataset["Remaining"] !=
0][species_per_dataset[species_per_dataset["Remaining"] != 0]['Species'] ==
species].index
    for n in index:
        species_per_dataset.loc[n,'Selection'] += min(int((1000 -
species_per_dataset.loc[n, 'Total'])/length),
species_per_dataset.loc[n,'Remaining'])
for species in list_of_species:
    species_per_dataset.loc[species_per_dataset['Species'] == species,
'Total'] =  sum(species_per_dataset.loc[species_per_dataset['Species'] ==
species, 'Selection'])
    species_per_dataset.loc[species_per_dataset["Species"] == species,
"Remaining"] = species_per_dataset.loc[species_per_dataset["Species"] ==
species, "Count"] - species_per_dataset.loc[species_per_dataset["Species"] ==
species,"Selection"]

# Make manual adjustments to final few counts in order to get total to 1,000.

species_per_dataset.loc[19,'Selection'] = 148
species_per_dataset.loc[20,'Selection'] = 148
species_per_dataset.loc[21,'Selection'] = 148
species_per_dataset.loc[23,'Selection'] = 148
species_per_dataset.loc[36,'Selection'] = 244
species_per_dataset.loc[46,'Selection'] = 479
species_per_dataset.loc[54,'Selection'] = 251
species_per_dataset.loc[69,'Selection'] = 334
species_per_dataset.loc[74,'Selection'] = 428
```

```
species_per_dataset.loc[90,'Selection'] = 355
species_per_dataset.loc[100,'Selection'] = 162
species_per_dataset.loc[101,'Selection'] = 162
species_per_dataset.loc[102,'Selection'] = 162
species_per_dataset.loc[104,'Selection'] = 162
species_per_dataset.loc[108,'Selection'] = 84
species_per_dataset.loc[109,'Selection'] = 84
species_per_dataset.loc[110,'Selection'] = 84
species_per_dataset.loc[111,'Selection'] = 84
species_per_dataset.loc[121,'Selection'] = 216
species_per_dataset.loc[123,'Selection'] = 216
species_per_dataset.loc[130,'Selection'] = 216
species_per_dataset.loc[133,'Selection'] = 306
species_per_dataset.loc[134,'Selection'] = 306
species_per_dataset.loc[142,'Selection'] = 334
species_per_dataset.loc[159,'Selection'] = 202
species_per_dataset.loc[160,'Selection'] = 202
species_per_dataset.loc[161,'Selection'] = 202
species_per_dataset.loc[163,'Selection'] = 195
species_per_dataset.loc[165,'Selection'] = 195
species_per_dataset.loc[167,'Selection'] = 195
species_per_dataset.loc[186,'Selection'] = 178
species_per_dataset.loc[188,'Selection'] = 178
species_per_dataset.loc[191,'Selection'] = 178
species_per_dataset.loc[192,'Selection'] = 178
species_per_dataset.loc[195,'Selection'] = 230
species_per_dataset.loc[230,'Selection'] = 238
species_per_dataset.loc[253,'Selection'] = 427
species_per_dataset.loc[255,'Selection'] = 166
species_per_dataset.loc[256,'Selection'] = 166
species_per_dataset.loc[293,'Selection'] = 334
species_per_dataset.loc[312,'Selection'] = 334
species_per_dataset.loc[315,'Selection'] = 306
species_per_dataset.loc[323,'Selection'] = 267
species_per_dataset.loc[326,'Selection'] = 267
species_per_dataset.loc[330,'Selection'] = 377
species_per_dataset.loc[346,'Selection'] = 143
species_per_dataset.loc[347,'Selection'] = 143
species_per_dataset.loc[348,'Selection'] = 143
species_per_dataset.loc[349,'Selection'] = 143
species_per_dataset.loc[350,'Selection'] = 143
species_per_dataset.loc[351,'Selection'] = 143

# Check that all totals are now 1,000

species_per_dataset.groupby(['Species']).sum()

# Second check to confirm we have exactly 1,000 of all species:

species_per_dataset.groupby(['Species']).sum()['Selection'].unique()

# Function to select random images based on the quantities defined in the
species_per_dataset dataframe

def choose_random_images(dataset, dataframe):
    df = pd.DataFrame(columns = dataframe.columns)
    for species in species_per_dataset['Species'].unique():
```

```
        if species in dataframe['Species'].unique():
            qty = int(species_per_dataset[(species_per_dataset['Species'] ==
species)
                & (species_per_dataset['Dataset'] == dataset)]['Selection'])
            temp_df = (dataframe[dataframe['Species'] ==
species].sample(qty))
            df = pd.concat([df, temp_df])
    return df

# Create new dataframes containing only information on the images to be used
in the project

df_list = [kgalagai_df, caltech_df, island_df, camdeboo_df, wcs_df, ena24_df,
wellington_df, karoo_df, enonkishu_df, mountain_zebra_df, kruger_df,
nacti_df, serengeti_df]
name_list = ['Kgalagai', 'Caltech', 'Island', 'Camdeboo', 'WCS', 'ENA 24',
'Wellington', 'Karoo', 'Enonkishu', 'Mountain Zebra', 'Kruger', 'NACTI',
'Serengeti']
subset_list = []
for i in range(0, len(df_list)):
    subset_list.append(choose_random_images(name_list[i], df_list[i]))
kgalagai_subset_df, caltech_subset_df, island_subset_df, camdeboo_subset_df,
wcs_subset_df, ena24_subset_df, \
wellington_subset_df, karoo_subset_df, enonkishu_subset_df,
mountain_zebra_subset_df, kruger_subset_df, \
nacti_subset_df, serengeti_subset_df = subset_list

# Merge new dataframes

image_selection_df = pd.concat([kgalagai_subset_df, caltech_subset_df,
                                island_subset_df, camdeboo_subset_df,
wcs_subset_df,
                                ena24_subset_df, wellington_subset_df,
karoo_subset_df,
                                enonkishu_subset_df, mountain_zebra_subset_df,
                                kruger_subset_df, nacti_subset_df,
serengeti_subset_df])

# Check length matches expected number (112,000)

len(image_selection_df.index)

# Function to pull out the file names to be downloaded (using AzCopy)

def images_to_download(dataframe):
    x = list(dataframe['file_name'])
    x = ';'.join(x)
    return x

# Get lists of images to be downloaded

kgalagai_images = images_to_download(kgalagai_subset_df)
caltech_images = images_to_download(caltech_subset_df)
island_images = images_to_download(island_subset_df)
camdeboo_images = images_to_download(camdeboo_subset_df)
wcs_images = images_to_download(wcs_dsubset_f)
ena24_images = images_to_download(ena24_subset_df)
```

```
wellington_images = images_to_download(wellington_subset_df)
karoo_images = images_to_download(karoo_subset_df)
enonkishu_images = images_to_download(enonkishu_subset_df)
mountain_zebra_images = images_to_download(mountain_zebra_subset_df)
kruger_images = images_to_download(kruger_subset_df)
nacti_images = images_to_download(nacti_subset_df)
serengeti_images = images_to_download(serengeti_subset_df)

# Dataframe contains some duplicates (two or more entries for same image).
Need to drop these:

image_selection_df = image_selection_df.drop_duplicates(subset='file_name')

len(image_selection_df.index)

image_selection_df['Species'].value_counts()

# Function to create training, validation and test datasets (with random
choices)

def choose_splits(dataframe):
    train_df = pd.DataFrame(columns = dataframe.columns)
    validation_df = pd.DataFrame(columns = dataframe.columns)
    for species in species_list:
        temp_df = (dataframe[dataframe['Species'] ==
species].sample(int(image_selection_df['Species'].value_counts()[species]*.8)
))
        train_df = pd.concat([train_df, temp_df])
    remainder_df = pd.concat([dataframe,
train_df]).drop_duplicates(keep=False)
    for species in species_list:
        temp_df = (remainder_df[remainder_df['Species'] ==
species].sample(int(image_selection_df['Species'].value_counts()[species]*.1)
))
        validation_df = pd.concat([validation_df, temp_df])
    test_df = pd.concat([remainder_df,
validation_df]).drop_duplicates(keep=False)
    return train_df, validation_df, test_df

# Create folders for training, validation and test datasets on local disk

path = "F:\\"
training_path = os.path.join(path, "training")
validation_path = os.path.join(path, "validation")
test_path = os.path.join(path, "test")
path_list = [training_path, validation_path, test_path]
for path in path_list:
    if not os.path.exists(path):
        os.makedirs(path)

# List of species to be used

species_list = list(image_selection_df['Species'].unique())

# Create folders in each of the training, validation and test folders
# One folder for each species in species list
```

```
train_path = "F:\\training"
validation_path = "F:\\validation"
test_path = "F:\\test"

for species in species_list:
    os.makedirs(os.path.join(train_path, species))
    os.makedirs(os.path.join(validation_path, species))
    os.makedirs(os.path.join(test_path, species))

# Dictionary mapping datasets to their current location on the local drive

path_dict = {'Serengeti': 'F:\Serengeti\snapshotserengeti-unzipped',
             'WCS':'F:\WCS\wcs-unzipped',
'Enonkishu':'F:\Enonkishu\ENO_public',
             'Camdeboo':'F:\Camdeboo\CDB_public', 'Mountain
Zebra':'F:\Mountain Zebra\MTZ_public',
               'Kgalagai':'F:\Kgalagai\KGA_public',
'Kruger':'F:\Kruger\KRU_public',
             'ENA24':'F:\ENA24\images', 'Island':'F:\Island\public',
             'Wellington':'F:\Wellington\images',
'Caltech':'F:\Caltech\cct_images',
               'Karoo':'F:\Karoo\KAR_public', "NACTI":r"F:\NACTI\nacti-
unzipped"}

# On inspection, it is clear that files belonging to the WCS dataset often
share the same name. The initial
# download placed them in different folders (Part 0, Part 1, etc). However,
they will potentially be placed
# in the same folder when they are arranged into training, validation and
test sets. Therefore, it is necessary
# to modify the file names in order to avoid overwriting existing files and
mixing up labels.

# Begin by adding a column with a unique number for each file (as a string)

image_selection_df['unique_id'] = np.arange(image_selection_df.shape[0])
image_selection_df['unique_id'] = image_selection_df['unique_id'].astype(str)

# Add another column to dataframe, this time consisting of the file name
without "jpg"

modification = []
for file in image_selection_df['file_name']:
        # USe reverse find to work backwards to find start of 'jpg'
        index = file.rfind(".")
        modification.append(file[:index])
image_selection_df['file_name_modified'] = modification

# Add the unique number onto the modified file name, plus '.jpg'

image_selection_df.loc[image_selection_df["Dataset"] == 'WCS',
    "file_name_modified"] =
image_selection_df.loc[image_selection_df["Dataset"] == 'WCS',

"file_name_modified"]+"_"+image_selection_df.loc[image_selection_df["Dataset"
] == 'WCS',
                    "unique_id"]+".jpg"
```

```
# Create dataframe of just the WCS dataset images

wcs_new = image_selection_df[image_selection_df['Dataset'] == 'WCS']

# Find the files (using the original file name) and rename using the modified
filename

for file in wcs_new['file_name']:
    old_name = os.path.join(path_dict['WCS'], file)
    new_name = os.path.join(path_dict['WCS'], wcs_new[wcs_new['file_name'] ==
file]['file_name_modified'].iloc[0])
    try:
        os.rename(old_name, new_name)
    except:
        continue

# Can now correct the file name column in the main dataframe

image_selection_df.loc[image_selection_df['Dataset'] == 'WCS',
        'file_name'] = image_selection_df.loc[image_selection_df['Dataset']
== 'WCS', 'file_name_modified']

# Use the earlier defined function to create the training, validation and
test splits

train_df, validation_df, test_df = choose_splits(image_selection_df)

# Function to copy files from their downloaded location into the correct
folder in the
# appropriate dataset (training, validation or test)

def file_copier(dataframe, path):
    for species in training_species:
        temp_df = dataframe[dataframe['Species'] == species]
        for dataset in temp_df['Dataset'].unique():
            temp_df2 = temp_df[temp_df['Dataset'] == dataset]
            for file in temp_df2['file_name']:
                file_to_locate = os.path.join(path_dict[dataset], file)
                new_location = os.path.join(path, species)
                try:
                    shutil.copy2(file_to_locate, new_location)
                except:
                    continue

file_copier(train_df, "F:\\training")

file_copier(validation_df, "F:\\validation")

file_copier(test_df, "F:\\test")

# Can now move on to creating the family-level datasets

# Check for and locate any entries without a family value

image_selection_df['Species'][image_selection_df['Family'].isna()].unique()
```

```
# Get index of missing value

image_selection_df['Family'][image_selection_df['Species'] ==
'Porcupine'][image_selection_df['Family'][image_selection_df['Species'] ==
'Porcupine'].isna()]

# Correct missing family value

image_selection_df.loc[55942, 'Family'] = 'Hystricidae'

# Set subfamily and faily to 'None' for empty images

image_selection_df.loc[image_selection_df['Species'] == 'Empty',
['Subfamily', 'Family']] = 'None'

# Get list of family values for entries that do not have a subfamily
classification

families_to_use =
list(image_selection_df['Family'][image_selection_df['Subfamily'].isna()].uni
que())

# Add column for one classification (either subfamily or family), intially
filled iwth subfamily values

image_selection_df['Subfamily/Family'] = image_selection_df['Subfamily']

# Change the subfamily/family entry to family classification when subfamily
doesn't exist

for family in families_to_use:
    image_selection_df.loc[image_selection_df['Family'] == family,
['Subfamily/Family']] = family

# Check unique subfamily/family types

image_selection_df['Subfamily/Family'].unique()

# Correct typing error in Reduncinae (extra space at end of word)

image_selection_df.loc[image_selection_df['Subfamily/Family'] == 'Reduncinae
', ['Subfamily/Family']] = 'Reduncinae'

# Save final image selection dataframe

image_selection_df.to_csv("image_selection_df.csv")

# Create folder to the family-level datasets

if not os.path.exists("F:\\family_data"):
    os.makedirs("F:\\family_data")

# Create training, validation and test folders

path = "F:\\family_data"
training_path = os.path.join(path, "training_family")
validation_path = os.path.join(path, "validation_family")
```

```
test_path = os.path.join(path, "test_family")
path_list = [training_path, validation_path, test_path]
for path in path_list:
    if not os.path.exists(path):
        os.makedirs(path)

# Create subfolders in each main folder for every subfamily/family
classification

families = list(image_selection_df['Subfamily/Family'].unique())
train_path = "F:\\family_data\\training_family"
validation_path = "F:\\family_data\\validation_family"
test_path = "F:\\family_data\\test_family"

for family in families:
    os.makedirs(os.path.join(train_path, family))
    os.makedirs(os.path.join(validation_path, family))
    os.makedirs(os.path.join(test_path, family))

# Function to create random training, validation and test splits

def choose_splits_family(dataframe):
    train_df = pd.DataFrame(columns = dataframe.columns)
    validation_df = pd.DataFrame(columns = dataframe.columns)
    for family in families:
        temp_df = (dataframe[dataframe['Subfamily/Family'] ==
family].sample(int(dataframe['Subfamily/Family'].value_counts()[family]*.8)))
        train_df = pd.concat([train_df, temp_df])
    remainder_df = pd.concat([dataframe,
train_df]).drop_duplicates(keep=False)
    for family in families:
        temp_df = (remainder_df[remainder_df['Subfamily/Family'] ==
family].sample(int(dataframe['Subfamily/Family'].value_counts()[family]*.1)))
        validation_df = pd.concat([validation_df, temp_df])
    test_df = pd.concat([remainder_df,
validation_df]).drop_duplicates(keep=False)
    return train_df, validation_df, test_df

# Create the splits

family_train_df, family_validation_df, family_test_df =
choose_splits_family(image_selection_df)

# Function to copy files from original download location to approrpiate
family-level folder

def file_copier_family(dataframe, path):
    for family in families:
        temp_df = dataframe[dataframe['Subfamily/Family'] == family]
        for dataset in temp_df['Dataset'].unique():
            temp_df2 = temp_df[temp_df['Dataset'] == dataset]
            for file in temp_df2['file_name']:
                file_to_locate = os.path.join(path_dict[dataset], file)
                new_location = os.path.join(path, family)
                try:
                    shutil.copy2(file_to_locate, new_location)
                except:
```

```
                    continue

file_copier_family(family_validation_df,
"F:\\family_data\\validation_family")

file_copier_family(family_test_df, "F:\\family_data\\test_family")

file_copier_family(family_train_df, "F:\\family_data\\training_family")

# Import credentials for GCP bucket

os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="x-pathway-318914-
d2e75ae928d4.json"

# Define GCP bucket

client=storage.Client()
bucket = client.get_bucket("dsm500_bucket_europe_west2_a")

# Create function to upload images from local disk to appropriate folder in
GCP bucket

def upload_local_directory_to_gcs(upload_list, location, bucket):
    for item in upload_list:
        local_path = "F://" + location + "/" + item
        gcs_path = location + "/" + item
        assert os.path.isdir(local_path)
        for local_file in glob.glob(local_path + '/**'):
            if not os.path.isfile(local_file):
                upload_local_directory_to_gcs(local_file, bucket, gcs_path +
"/" + os.path.basename(local_file))
            else:
                remote_path = gcs_path + "/" + local_file[1 +
len(local_path):]
                blob = bucket.blob(remote_path)
                blob.upload_from_filename(local_file)

upload_local_directory_to_gcs(species_list, "training", bucket)

upload_local_directory_to_gcs(species_list, "validation", bucket)

upload_local_directory_to_gcs(species_list, "test", bucket)

# Create list of subfamily/family classifications to use in uploading
function

family_list = list(image_selection_df['Subfamily/Family'].unique())

upload_local_directory_to_gcs(family_list, "family_data/training_family",
bucket)

upload_local_directory_to_gcs(family_list, "family_data/validation_family",
bucket)

upload_local_directory_to_gcs(family_list, "family_data/test_family", bucket)
```

```
# Initial model training attempt required too much computational power.
Therefore, decision was made to
# delete any subfamilies/families with only one species

# Get list of families/subfamilies with less than/equal to 1,000 images
(indicating presence of only one species)

families_to_delete =
list((image_selection_df.groupby('Subfamily/Family').size()[image_selection_d
f.groupby('Subfamily/Family').size() <=1000]).index)

# Use families list to get species images to delete
species_to_delete = []
for family in families_to_delete:

species_to_delete.append(image_selection_df[image_selection_df['Subfamily/Fam
ily'] == family]['Species'].unique()[0])

# Next, need to get extra unused images to be used for the species_level
classifier test sets

# Create dataframe of unused images, then filter it to contain only Canidae,
Felinae or Sciruidae images

used_files = list(image_selection_df['file_name'])
unused_images_df = subset_df[~subset_df['id'].isin(used_files)]
unused_images_df = unused_images_df.drop_duplicates(subset='id')
# Use 'felidae' in the or statement in order to filter by family column
unused_images_df = unused_images_df.loc[(unused_images_df['Family'] ==
'Felidae') | (unused_images_df['Family'] == 'Sciuridae') |
(unused_images_df['Family'] == 'Canidae')]
# Remove pantherinae in order to be left with felinae
unused_images_df = unused_images_df[~(unused_images_df['Subfamily'] ==
'Pantherinae')]

# Repeat process performed above to get proprotoinate number of images from
each different dataset

species_per_dataset = unused_images_df.groupby(['Species', 'Dataset']).size()
species_per_dataset = pd.DataFrame(species_per_dataset)
species_per_dataset.reset_index(inplace=True)
species_per_dataset.columns = ['Species', 'Dataset', 'Count']
species_per_dataset = species_per_dataset[~(species_per_dataset['Dataset'] ==
'Missouri')]
species_per_dataset['Selection'] = species_per_dataset['Count']
list_of_species = list(species_per_dataset['Species'].unique())
for species in list_of_species:
    length = len(species_per_dataset[species_per_dataset['Species'] ==
species])
    index =
species_per_dataset[species_per_dataset['Species']==species].index
    for n in index:
        if species_per_dataset['Selection'][n] <= int(100/length):
            continue
        else:
            species_per_dataset.loc[n,'Selection'] = int(100/length)
grouped = species_per_dataset.groupby(['Species']).sum()
```

```
for species in grouped.index:
    species_per_dataset.loc[species_per_dataset['Species'] == species,
'Total'] =  grouped.loc[species,'Selection']

# Manual correction of one row

species_per_dataset.loc[38,'Selection'] = 50+20

# Confirm selectin includes 100 images for each species

species_per_dataset.groupby(['Species']).sum()

# Create new dataframes of unused images by dataset

kgalagai_new_df = unused_images_df[unused_images_df["Dataset"] == "Kgalagai"]
caltech_new_df = unused_images_df[unused_images_df["Dataset"] == "Caltech"]
island_new_df = unused_images_df[unused_images_df["Dataset"] == "Island"]
missouri_new_df = unused_images_df[unused_images_df["Dataset"] == "Missouri"]
camdeboo_new_df = unused_images_df[unused_images_df["Dataset"] == "Camdeboo"]
wcs_new_df = unused_images_df[unused_images_df["Dataset"] == "WCS"]
ena24_new_df = unused_images_df[unused_images_df["Dataset"] == "ENA24"]
wellington_new_df = unused_images_df[unused_images_df["Dataset"] ==
"Wellington"]
karoo_new_df = unused_images_df[unused_images_df["Dataset"] == "Karoo"]
enonkishu_new_df = unused_images_df[unused_images_df["Dataset"] ==
"Enonkishu"]
mountain_zebra_new_df = unused_images_df[unused_images_df["Dataset"] ==
"Mountain Zebra"]
kruger_new_df = unused_images_df[unused_images_df["Dataset"] == "Kruger"]
nacti_new_df = unused_images_df[unused_images_df["Dataset"] == "NACTI"]
serengeti_new_df = unused_images_df[unused_images_df["Dataset"] ==
"Serengeti"]

# Use new dataframes and image selection function to get dataframe of new
test images

kgalagai_new_test = choose_random_images('Kgalagai', kgalagai_new_df)
caltech_new_test = choose_random_images('Caltech', caltech_new_df)
island_new_test = choose_random_images('Island', island_new_df)
camdeboo_new_test = choose_random_images('Camdeboo', camdeboo_new_df)
wcs_new_test = choose_random_images('WCS', wcs_new_df)
ena24_new_test = choose_random_images('ENA24', ena24_new_df)
wellington_new_test = choose_random_images('Wellington', wellington_new_df)
karoo_new_test = choose_random_images('Karoo', karoo_new_df)
enonkishu_new_test = choose_random_images('Enonkishu', enonkishu_new_df)
mountain_new_test = choose_random_images('Mountain Zebra',
mountain_zebra_new_df)
kruger_new_test = choose_random_images('Kruger', kruger_new_df)
nacti_new_test = choose_random_images('NACTI', nacti_new_df)
serengeti_new_test = choose_random_images('Serengeti', serengeti_new_df)

# Combbine into single dataframe

new_image_selection_df = pd.concat([kgalagai_new_test, caltech_new_test,
                            island_new_test, camdeboo_new_test, wcs_new_test,
                            ena24_new_test, wellington_new_test,
                            karoo_new_test, enonkishu_new_test,
```

```
                            mountain_new_test, kruger_new_test,
                            nacti_new_test, serengeti_new_test])

# Use eariler function to get download information for use with AzCopy

kgalagai_new_images = images_to_download('Kgalagai', kgalagai_new_test)
caltech_new_images = images_to_download('Caltech', caltech_new_test)
island_new_images = images_to_download('Island', island_new_test)
camdeboo_new_images = images_to_download('Camdeboo', camdeboo_new_test)
wcs_new_images = images_to_download('WCS', wcs_new_test)
ena24_new_images = images_to_download('ENA24', ena24_new_test)
wellington_new_images = images_to_download('Wellington', wellington_new_test)
karoo_new_images = images_to_download('Karoo', karoo_new_test)
enonkishu_new_images = images_to_download('Enonkishu', enonkishu_new_test)
mountain_zebra_new_images = images_to_download('Mountain Zebra',
mountain_new_test)
kruger_new_images = images_to_download('Kruger', kruger_new_test)
nacti_new_images = images_to_download('NACTI', nacti_new_test)
serengeti_new_images = images_to_download('Serengeti', serengeti_new_test)

# Define new mapping and species list for copying files to correct location

path_dict_new = {'Serengeti':
'F:\Final_Test_Data\Serengeti\snapshotserengeti-unzipped',
             'WCS':'F:\Final_Test_Data\WCS\wcs-unzipped',
             'Camdeboo':'F:\Final_Test_Data\Camdeboo\CDB_public',
                'Mountain Zebra':'F:\Final_Test_Data\Mountain
Zebra\MTZ_public',
              'Kgalagai':'F:\Final_Test_Data\Kgalagai\KGA_public',
                'Kruger':'F:\Final_Test_Data\Kruger\KRU_public',
             'ENA24':'F:\Final_Test_Data\ENA24\images',
                'Island':'F:\Final_Test_Data\Island\public',
             'Wellington':'F:\Final_Test_Data\Wellington\images',
                'Caltech':'F:\Final_Test_Data\Caltech\cct_images',
               'Karoo':'F:\Final_Test_Data\Karoo\KAR_public',
                "NACTI":r"F:\Final_Test_Data\NACTI\nacti-unzipped"}

new_species_list = list(new_image_selection_df['Species'].unique())

# Modified file copier function

def file_copier(dataframe, path):
    for species in new_species_list:
        temp_df = dataframe[dataframe['Species'] == species]
        for dataset in temp_df['Dataset'].unique():
            temp_df2 = temp_df[temp_df['Dataset'] == dataset]
            for file in temp_df2['file_name']:
                file_to_locate = os.path.join(path_dict_new[dataset], file)
                new_location = os.path.join(path, species)
                try:
                    shutil.copy2(file_to_locate, new_location)
                except:
                    continue

# Copy files

file_copier(new_image_selection_df, "F:\\Final_Test_Data\Test")
```

```
# Create lists for uploading to CGP

felinae_list =
list(new_image_selection_df[new_image_selection_df['Subfamily'] ==
'Felinae']['Species'].unique())
canidae_list = list(new_image_selection_df[new_image_selection_df['Family']
== 'Canidae']['Species'].unique())
sciuridae_list = list(new_image_selection_df[new_image_selection_df['Family']
== 'Sciuridae']['Species'].unique())

upload_local_directory_to_gcs(felinae_list, "felinae/test", bucket)

upload_local_directory_to_gcs(canidae_list, "canidae/test", bucket)

upload_local_directory_to_gcs(sciuridae_list, "sciuridae/test", bucket)

# Now need to upload out-of-sample data using Missouri dataset. Entire
dataset was previously downloaded in bulk,
# so simply a case of selecting approrpiate images and uploading to GCP

# Create dataframe of appropriate images (canidae, felinae, sciuridae)

missouri_df = unused_images_df[unused_images_df['Dataset']=='Missouri']

def upload_local_directory_to_gcs(upload_list, location, bucket):
    for item in upload_list:
        local_path = "F://" + location + "/" + item
        gcs_path = location + "/" + item
        assert os.path.isdir(local_path)
        for local_file in glob.glob(local_path + '/**'):
            if not os.path.isfile(local_file):
                upload_local_directory_to_gcs(local_file, bucket, gcs_path +
"/" + os.path.basename(local_file))
            else:
                remote_path = gcs_path + "/" + local_file[1 +
len(local_path):]
                blob = bucket.blob(remote_path)
                blob.upload_from_filename(local_file)

# Simplified uploaded function to account for fact that only one species
present per class (so list not used)

def upload_local_directory_to_gcs2(item, location, bucket):
    local_path = "F://" + location + "/" + item
    gcs_path = location + "/" + item
    assert os.path.isdir(local_path)
    for local_file in glob.glob(local_path + '/**'):
        if not os.path.isfile(local_file):
            upload_local_directory_to_gcs(local_file, bucket, gcs_path + "/"
+ os.path.basename(local_file))
        else:
            remote_path = gcs_path + "/" + local_file[1 + len(local_path):]
            blob = bucket.blob(remote_path)
            blob.upload_from_filename(local_file)
```

```
# Locate in species list and upload the specific species present in missouri
dataset

upload_local_directory_to_gcs2(canidae_list[2], "canidae/out_of_sample",
bucket)

upload_local_directory_to_gcs2(sciuridae_list[1], "sciuridae/out_of_sample",
bucket)

upload_local_directory_to_gcs2(felinae_list[3], "felinae/out_of_sample",
bucket)
```

- *Modelling*

```
# Prevent TensorFlow from allocating all GPU memory

import tensorflow as tf
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)

# Import necessary libraries and modules

import pandas as pd
import numpy as np
import os
from PIL import ImageFile
from tensorflow.keras import layers
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import json
from tensorflow.keras.callbacks import BaseLogger
from tensorflow.keras.models import load_model
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import Callback

# Handle truncated images

ImageFile.LOAD_TRUNCATED_IMAGES = True

# Define classes for benchmark model

classes = os.listdir('MOUNT_DIRECTORY/training')

# Define data augmentation for training and validation sets (no augmentation
for validation)

trainAug = ImageDataGenerator(
    rotation_range=25,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest")

valAug = ImageDataGenerator()

# Define training and validation directories
```

```
train_dir = "MOUNT_DIRECTORY/training"
validation_dir = "MOUNT_DIRECTORY/validation"

# Use flow_from_directory to create training and validation generators, using
batches of images

trainGen = trainAug.flow_from_directory(
    train_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=True,
    batch_size=128)

valGen = valAug.flow_from_directory(
    validation_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

# Define callback to save model training progress after specified number of
epochs

class EpochCheckpoint(Callback):
    def __init__(self, outputPath, every=5, startAt=0):
        super(Callback, self).__init__()
        self.outputPath = outputPath
        self.every = every
        self.intEpoch = startAt

    def on_epoch_end(self, epoch, logs={}):
        if (self.intEpoch + 1) % self.every == 0:
            p = os.path.sep.join([self.outputPath,
"epoch_{}".format(self.intEpoch + 1)])
            self.model.save(p, overwrite = True)
            self.intEpoch += 1

# Define callback to save and plot metrics during training

class TrainingMonitor(BaseLogger):
    def __init__(self, figPath, jsonPath=None, startAt=0):
        # store the output path for the figure, the path to the JSON
        # serialized file, and the starting epoch
        super(TrainingMonitor, self).__init__()
        self.figPath = figPath
        self.jsonPath = jsonPath
        self.startAt = startAt

    def on_train_begin(self, logs={}):
        # initialize the history dictionary
        self.H = {}

        # if the JSON history path exists, load the training history
        if self.jsonPath is not None:
```

```
            if os.path.exists(self.jsonPath):
                self.H = json.loads(open(self.jsonPath).read())

                # check to see if a starting epoch was supplied
                if self.startAt > 0:
                    # loop over the entries in the history log and
                    # trim any entries that are past the starting
                    # epoch
                    for k in self.H.keys():
                        self.H[k] = self.H[k][:self.startAt]

    def on_epoch_end(self, epoch, logs={}):
        # loop over the logs and update the loss, accuracy, etc.
        # for the entire training process
        for (k, v) in logs.items():
            l = self.H.get(k, [])
            l.append(float(v))
            self.H[k] = l

        # check to see if the training history should be serialized
        # to file
        if self.jsonPath is not None:
            f = open(self.jsonPath, "w")
            f.write(json.dumps(self.H))
            f.close()

        # ensure at least two epochs have passed before plotting
        # (epoch starts at zero)
        if len(self.H["loss"]) > 1:
            # plot the training loss and accuracy
            N = np.arange(0, len(self.H["loss"]))
            plt.style.use("ggplot")
            plt.figure()
            plt.plot(N, self.H["loss"], label="train_loss")
            plt.plot(N, self.H["val_loss"], label="val_loss")
            plt.plot(N, self.H["accuracy"], label="train_acc")
            plt.plot(N, self.H["val_accuracy"], label="val_acc")
            plt.title("Training Loss and Accuracy [Epoch {}]".format(
                len(self.H["loss"])))
            plt.xlabel("Epoch #")
            plt.ylabel("Loss/Accuracy")
            plt.legend()

            # save the figure
            plt.savefig(self.figPath)
            plt.close()

# Define preprocessing layer, followed by loading MobileNetV2 architecture
with weights learned from ImageNet dataset
# Set include_top to false to remove final layers

i = tf.keras.layers.Input([None, None, 3], dtype = tf.uint8)
x = tf.cast(i, tf.float32)
x = tf.keras.applications.mobilenet_v2.preprocess_input(x)
core = tf.keras.applications.MobileNetV2(weights="imagenet",
include_top=False,
input_tensor=Input(shape=(224, 224, 3)))
```

```
# Freeze all layers
for layer in core.layers:
    layer.trainable = False
# Unfreeze final five layers
for layer in core.layers[150:]:
    layer.trainable = True

# Add four additional layers to model, specifying output layer size using
classes earlier defined

x = core(x)
baseModel = Model(inputs=[i], outputs=[x])

y = baseModel.output
y = layers.Flatten()(y)
y = layers.Dense(1024, activation='relu')(y)
y = layers.Dropout(0.2)(y)
y = layers.Dense(len(classes), activation="softmax")(y)

model = Model(inputs=baseModel.input, outputs=y)

# Set existing model to None to prevent loading any model and resuming
training

existing_model = None

# Compile model with predetermined hyperparameters

if existing_model is None:
    opt = Adam(learning_rate=1e-4)
    model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy", "top_k_categorical_accuracy"])
# Provide the option of resuming from an existing model in case training is
interrupted
else:
    model = load_model(existing_model)
    K.set_value(model.optimizer.learning_rate, 1e-5)

# Define checkpoints and starting epoch to be used by callback when saving
models and performance

checkpoints = "MOUNT_DIRECTORY/output/checkpoints"
start_epoch = 0

# Define location for callbacks to save performance

plotPath = os.path.sep.join(["MOUNT_DIRECTORY/output/", "benchmark.png"])
jsonPath = os.path.sep.join(["MOUNT_DIRECTORY/output/", "benchmark.json"])

# Define callbacks so as to save model after every epoch

callbacks = [
    EpochCheckpoint(checkpoints, every=1,
        startAt=start_epoch),
    TrainingMonitor(plotPath,
        jsonPath=jsonPath,
        startAt=start_epoch)]
```

```
# Train benchmark model

history = model.fit(
    trainGen,
    validation_data=valGen,
    epochs=50,
  callbacks=callbacks)

# Define classes for family model

family_classes = os.listdir('MOUNT_DIRECTORY/family_data/training_family')

# Clear existing training and validatoin generators

trainGen.reset()
valGen.reset()

# Define new generators for the family model

train_dir = 'MOUNT_DIRECTORY/family_data/training_family'
validation_dir = "MOUNT_DIRECTORY/family_data/validation_family"

trainGen = trainAug.flow_from_directory(
    train_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=True,
    batch_size=128)

valGen = valAug.flow_from_directory(
    validation_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

# Repeat model building process for family model, using family classes for
output

i = tf.keras.layers.Input([None, None, 3], dtype = tf.uint8)
x = tf.cast(i, tf.float32)
x = tf.keras.applications.mobilenet_v2.preprocess_input(x)
core = tf.keras.applications.MobileNetV2(weights="imagenet",
include_top=False,
input_tensor=Input(shape=(224, 224, 3)))
for layer in core.layers:
    layer.trainable = False
for layer in core.layers[150:]:
    layer.trainable = True

x = core(x)
baseModel = Model(inputs=[i], outputs=[x])

y = baseModel.output
```

```
y = layers.Flatten()(y)
y = layers.Dense(1024, activation='relu')(y)
y = layers.Dropout(0.2)(y)
y = layers.Dense(len(family_classes), activation="softmax")(y)

model = Model(inputs=baseModel.input, outputs=y)

existing_model = None

if existing_model is None:
    opt = Adam(learning_rate=1e-4, decay=1e-4 / 50)
    model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy", "top_k_categorical_accuracy"])
else:
    model = load_model(existing_model)
    K.set_value(model.optimizer.learning_rate, 1e-5)

checkpoints = "MOUNT_DIRECTORY/family_data/output/checkpoints"
start_epoch = 0

plotPath = os.path.sep.join(["MOUNT_DIRECTORY/family_data/output/",
"family.png"])
jsonPath = os.path.sep.join(["MOUNT_DIRECTORY/family_data/output/",
"family.json"])

# Train family model

history = model.fit(
    trainGen,
    validation_data=valGen,
    epochs=50,
  callbacks=callbacks)

# Load benchmark model from best epoch

benchmark_model = load_model('MOUNT_DIRECTORY/output/checkpoints/epoch_48')

# Define test generator (again without data augmentation) to evaluate model
performance

test_dir = 'MOUNT_DIRECTORY/test'

testGen = valAug.flow_from_directory(
    test_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

# Evaluate bechmark model and save scores to csv file

benchmark_score = benchmark_model.evaluate(testGen)
print("Evaluation finished")
benchmark_score_df = pd.DataFrame({'Loss':benchmark_score[0], 'Top-1-
Accuracy':benchmark_score[1],
```

```
                                    'Top-5-Accuracy':benchmark_score[2]},
index=['Performance'])
benchmark_score_df.to_csv('MOUNT_DIRECTORY/output/benchmark_score.csv')
benchmark_predIdxs = benchmark_model.predict(x=testGen)
print("Predictions finished")
benchmark_predIdxs = np.argmax(benchmark_predIdxs, axis=1)
# Use scikit-learn classificatoin report to create and save csv file of
precision, recall and F1-score
benchmark_report = classification_report(testGen.classes, benchmark_predIdxs,

target_names=testGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(benchmark_report).transpose()
df.to_csv('MOUNT_DIRECTORY/output/benchmark_evaluation.csv')

# Create (and save) dataframe of incorrect predictions along with file names

benchmark_mistakes = pd.DataFrame.from_dict({"Prediction":
benchmark_predIdxs, "Actual": testGen.classes})
benchmark_mistakes.to_csv('MOUNT_DIRECTORY/output/benchmark_mistakes.csv')

# Repeat process with out-of-sample data

oos_dir = 'MOUNT_DIRECTORY/out_of_sample'

oosGen = valAug.flow_from_directory(
    oos_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

benchmark_oos_score = benchmark_model.evaluate(oosGen)
print("Evaluation finished")
benchmark_oos_score_df = pd.DataFrame({'Loss':benchmark_oos_score[0], 'Top-1-
Accuracy':benchmark_oos_score[1],
                                       'Top-5-
Accuracy':benchmark_oos_score[2]}, index=['Performance'])
benchmark_oos_score_df.to_csv('MOUNT_DIRECTORY/output/benchmark_oos_score.csv
')
benchmark_oos_predIdxs = benchmark_model.predict(x=oosGen)
print("Predictions finished")
benchmark_oos_predIdxs = np.argmax(benchmark_oos_predIdxs, axis=1)
# Necessary to define labels, otherwise dataframe dimensions do not match
benchmark_oos_report = classification_report(oosGen.classes,
benchmark_oos_predIdxs,
                                  labels = list(range(0,79)),
target_names=oosGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(benchmark_oos_report).transpose()
df.to_csv('MOUNT_DIRECTORY/output/benchmark_oos_evaluation.csv')

# Save out-of-sample mistakes

benchmark_oos_mistakes = pd.DataFrame.from_dict({"Prediction":
benchmark_oos_predIdxs, "Actual": oosGen.classes})
benchmark_oos_mistakes.to_csv('MOUNT_DIRECTORY/output/benchmark_oos_mistakes.
csv')
```

```
# Repeat process with family model

family_model =
load_model('MOUNT_DIRECTORY/family_data/output/checkpoints/epoch_45')

testGen.reset()

test_dir = 'MOUNT_DIRECTORY/family_data/test_family'

testGen = valAug.flow_from_directory(
    test_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

family_score = family_model.evaluate(testGen)
print("Evaluation finished")
score_df = pd.DataFrame({'Loss':family_score[0], 'Top-1-
Accuracy':family_score[1], 'Top-5-Accuracy':family_score[2]},
                        index=['Performance'])
score_df.to_csv('MOUNT_DIRECTORY/family_data/output/score.csv')
family_predIdxs = family_model.predict(x=testGen)
print("Predictions finished")
family_predIdxs = np.argmax(family_predIdxs, axis=1)
report = classification_report(testGen.classes, family_predIdxs,
target_names=testGen.class_indices.keys(),
                               output_dict=True)
df = pd.DataFrame(report).transpose()
df.to_csv('MOUNT_DIRECTORY/family_data/output/evaluation.csv')

# Define felinae classes in order to build felinae model

felinae_classes = os.listdir('MOUNT_DIRECTORY/felinae/training')

# Repeat process to create generators for felinae model

trainGen.reset()
valGen.reset()

train_dir = 'MOUNT_DIRECTORY/felinae/training'
validation_dir = "MOUNT_DIRECTORY/felinae/validation"

trainGen = trainAug.flow_from_directory(
    train_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=True,
    batch_size=128)

valGen = valAug.flow_from_directory(
    validation_dir,
    class_mode="categorical",
    target_size=(224, 224),
```

```
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

# Load family model (instead of pretrained MobileNetV2)

model = load_model('MOUNT_DIRECTORY/family_data/output/checkpoints/epoch_45')

# Replace last layer using felinae classes

felinae_model= Model(inputs=model.input, outputs=model.layers[-2].output)
y = felinae_model.output
y = layers.Dense(len(felinae_classes), activation="softmax")(y)
felinae_model = Model(inputs=felinae_model.input, outputs=y)

# Repeat training process

existing_model = None

if existing_model is None:
    opt = Adam(learning_rate=1e-4, decay=1e-4 / 50)
    felinae_model.compile(loss="categorical_crossentropy", optimizer=opt,
                          metrics=["accuracy", "top_k_categorical_accuracy"])
else:
    model = load_model(existing_model)
    K.set_value(model.optimizer.learning_rate, 1e-5)

checkpoints = "MOUNT_DIRECTORY/felinae/output/checkpoints"
start_epoch = 0

plotPath = os.path.sep.join(["MOUNT_DIRECTORY/felinae/output/",
"felinae.png"])
jsonPath = os.path.sep.join(["MOUNT_DIRECTORY/felinae/output/",
"felinae.json"])

callbacks = [
    EpochCheckpoint(checkpoints, every=1,
        startAt=start_epoch),
    TrainingMonitor(plotPath,
        jsonPath=jsonPath,
        startAt=start_epoch)]

history = felinae_model.fit(
    trainGen,
    validation_data=valGen,
    epochs=50,
    callbacks=callbacks)

# Repeat process for canidae and sciuridae models

canidae_classes = os.listdir('MOUNT_DIRECTORY/canidae/training')

trainGen.reset()
valGen.reset()

train_dir = 'MOUNT_DIRECTORY/canidae/training'
validation_dir = "MOUNT_DIRECTORY/canidae/validation"
```

```
trainGen = trainAug.flow_from_directory(
    train_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=True,
    batch_size=128)

valGen = valAug.flow_from_directory(
    validation_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

model = load_model('MOUNT_DIRECTORY/family_data/output/checkpoints/epoch_45')
canidae_model= Model(inputs=model.input, outputs=model.layers[-2].output)
y = canidae_model.output
y = layers.Dense(len(canidae_classes), activation="softmax")(y)
canidae_model = Model(inputs=canidae_model.input, outputs=y)

existing_model = None

if existing_model is None:
    opt = Adam(learning_rate=1e-4, decay=1e-4 / 50)
    canidae_model.compile(loss="categorical_crossentropy", optimizer=opt,
                          metrics=["accuracy", "top_k_categorical_accuracy"])
else:
    canidae_model = load_model(existing_model)
    K.set_value(canidae_model.optimizer.learning_rate, 1e-5)

checkpoints = "MOUNT_DIRECTORY/canidae/output/checkpoints"
start_epoch = 0

plotPath = os.path.sep.join(["MOUNT_DIRECTORY/canidae/output/",
"canidae.png"])
jsonPath = os.path.sep.join(["MOUNT_DIRECTORY/canidae/output/",
"canidae.json"])

callbacks = [
    EpochCheckpoint(checkpoints, every=1,
        startAt=start_epoch),
    TrainingMonitor(plotPath,
        jsonPath=jsonPath,
        startAt=start_epoch)]

history = canidae_model.fit(
    trainGen,
    validation_data=valGen,
    epochs=50,
    callbacks=callbacks)

sciuridae_classes = os.listdir('MOUNT_DIRECTORY/sciuridae/training')

trainGen.reset()
```

```
valGen.reset()

train_dir = 'MOUNT_DIRECTORY/sciuridae/training'
validation_dir = "MOUNT_DIRECTORY/sciuridae/validation"

trainGen = trainAug.flow_from_directory(
    train_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=True,
    batch_size=128)

valGen = valAug.flow_from_directory(
    validation_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

model = load_model('MOUNT_DIRECTORY/family_data/output/checkpoints/epoch_45')

sciuridae_model= Model(inputs=model.input, outputs=model.layers[-2].output)
y = sciuridae_model.output
y = layers.Dense(len(sciuridae_classes), activation="softmax")(y)
sciuridae_model = Model(inputs=sciuridae_model.input, outputs=y)

existing_model = None

if existing_model is None:
    opt = Adam(learning_rate=1e-4, decay=1e-4 / 50)
    sciuridae_model.compile(loss="categorical_crossentropy", optimizer=opt,
                            metrics=["accuracy",
"top_k_categorical_accuracy"])
else:
    sciuridae_model = load_model(existing_model)
    K.set_value(sciuridae_model.optimizer.learning_rate, 1e-5)

checkpoints = "MOUNT_DIRECTORY/sciuridae/output/checkpoints"
start_epoch = 0

plotPath = os.path.sep.join(["MOUNT_DIRECTORY/sciuridae/output/",
"sciuridae.png"])
jsonPath = os.path.sep.join(["MOUNT_DIRECTORY/sciuridae/output/",
"sciuridae.json"])

callbacks = [
    EpochCheckpoint(checkpoints, every=1,
        startAt=start_epoch),
    TrainingMonitor(plotPath,
        jsonPath=jsonPath,
        startAt=start_epoch)]

history = sciuridae_model.fit(
    trainGen,
    validation_data=valGen,
```

```
    epochs=50,
    callbacks=callbacks)

# Load felinae model from best epoch, then repeat model evaluation process

felinae_model =
load_model('MOUNT_DIRECTORY/felinae/output/checkpoints/epoch_40')

testGen.reset()

felinae_test_dir = 'MOUNT_DIRECTORY/felinae/test'

felinae_testGen = valAug.flow_from_directory(
    felinae_test_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

felinae_score = felinae_model.evaluate(felinae_testGen)
print("Evaluation finished")
felinae_score_df = pd.DataFrame({'Loss':felinae_score[0], 'Top-1-
Accuracy':felinae_score[1],
                                 'Top-5-Accuracy':felinae_score[2]},
index=['Performance'])
felinae_score_df.to_csv('MOUNT_DIRECTORY/felinae/output/felinae_score.csv')
felinae_predIdxs = felinae_model.predict(x=felinae_testGen)
print("Predictions finished")
felinae_predIdxs = np.argmax(felinae_predIdxs, axis=1)
felinae_report = classification_report(felinae_testGen.classes,
felinae_predIdxs,

target_names=felinae_testGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(felinae_report).transpose()
df.to_csv('MOUNT_DIRECTORY/felinae/output/felinae_evaluation.csv')

felinae_mistakes = pd.DataFrame.from_dict({"Prediction": felinae_predIdxs,
"Actual": felinae_testGen.classes})
felinae_mistakes.to_csv('MOUNT_DIRECTORY/felinae/output/felinae_mistakes.csv'
)

felinae_oos_dir = 'MOUNT_DIRECTORY/felinae/out_of_sample'

felinae_oosGen = valAug.flow_from_directory(
    felinae_oos_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

felinae_oos_score = felinae_model.evaluate(felinae_oosGen)
print("Evaluation finished")
felinae_oos_score_df = pd.DataFrame({'Loss':felinae_oos_score[0], 'Top-1-
Accuracy':felinae_oos_score[1],
```

```
                                        'Top-5-Accuracy':felinae_oos_score[2]},
index=['Performance'])
felinae_oos_score_df.to_csv('MOUNT_DIRECTORY/felinae/output/felinae_oos_score
.csv')
felinae_oos_predIdxs = felinae_model.predict(x=felinae_oosGen)
print("Predictions finished")
felinae_oos_predIdxs = np.argmax(felinae_oos_predIdxs, axis=1)
felinae_oos_report = classification_report(felinae_oosGen.classes,
felinae_oos_predIdxs,
                            labels = (0,1,2,3,4,5),
target_names=felinae_oosGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(felinae_oos_report).transpose()
df.to_csv('MOUNT_DIRECTORY/felinae/output/felinae_oos_evaluation.csv')

felinae_oos_mistakes = pd.DataFrame.from_dict({"Prediction":
felinae_oos_predIdxs, "Actual": felinae_oosGen.classes})
felinae_oos_mistakes.to_csv('MOUNT_DIRECTORY/felinae/output/felinae_oos_mista
kes.csv')

# Repeat for Canidae and Sciuridae models

canidae_model =
load_model('MOUNT_DIRECTORY/canidae/output/checkpoints/epoch_43')

testGen.reset()

canidae_test_dir = 'MOUNT_DIRECTORY/canidae/test'

canidae_testGen = valAug.flow_from_directory(
    canidae_test_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

canidae_score = canidae_model.evaluate(canidae_testGen)
print("Evaluation finished")
canidae_score_df = pd.DataFrame({'Loss':canidae_score[0], 'Top-1-
Accuracy':canidae_score[1],
                                    'Top-5-Accuracy':canidae_score[2]},
index=['Performance'])
canidae_score_df.to_csv('MOUNT_DIRECTORY/canidae/output/canidae_score.csv')
canidae_predIdxs = canidae_model.predict(x=canidae_testGen)
print("Predictions finished")
canidae_predIdxs = np.argmax(canidae_predIdxs, axis=1)
canidae_report = classification_report(canidae_testGen.classes,
canidae_predIdxs,

target_names=canidae_testGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(canidae_report).transpose()
df.to_csv('MOUNT_DIRECTORY/canidae/output/canidae_evaluation.csv')

canidae_mistakes = pd.DataFrame.from_dict({"Prediction": canidae_predIdxs,
"Actual": canidae_testGen.classes})
canidae_mistakes.to_csv('MOUNT_DIRECTORY/canidae/output/canidae_mistakes.csv'
)
```

```
canidae_oos_dir = 'MOUNT_DIRECTORY/canidae/out_of_sample'

canidae_oosGen = valAug.flow_from_directory(
    canidae_oos_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

canidae_oos_score = canidae_model.evaluate(canidae_oosGen)
print("Evaluation finished")
canidae_oos_score_df = pd.DataFrame({'Loss':canidae_oos_score[0], 'Top-1-
Accuracy':canidae_oos_score[1],
                                     'Top-5-Accuracy':canidae_oos_score[2]},
index=['Performance'])
canidae_oos_score_df.to_csv('MOUNT_DIRECTORY/canidae/output/canidae_oos_score
.csv')
canidae_oos_predIdxs = canidae_model.predict(x=canidae_oosGen)
print("Predictions finished")
canidae_oos_predIdxs = np.argmax(canidae_oos_predIdxs, axis=1)
canidae_oos_report = classification_report(canidae_oosGen.classes,
canidae_oos_predIdxs,
                            labels = (0,1,2,3,4,5),
target_names=canidae_oosGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(canidae_oos_report).transpose()
df.to_csv('MOUNT_DIRECTORY/canidae/output/canidae_oos_evaluation.csv')

canidae_oos_mistakes = pd.DataFrame.from_dict({"Prediction":
canidae_oos_predIdxs, "Actual": canidae_oosGen.classes})
canidae_oos_mistakes.to_csv('MOUNT_DIRECTORY/canidae/output/canidae_oos_mista
kes.csv')

sciuridae_model =
load_model('MOUNT_DIRECTORY/sciuridae/output/checkpoints/epoch_35')

testGen.reset()

sciuridae_test_dir = 'MOUNT_DIRECTORY/sciuridae/test'

sciuridae_testGen = valAug.flow_from_directory(
    sciuridae_test_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

sciuridae_score = sciuridae_model.evaluate(sciuridae_testGen)
print("Evaluation finished")
sciuridae_score_df = pd.DataFrame({'Loss':sciuridae_score[0], 'Top-1-
Accuracy':sciuridae_score[1],
                                   'Top-5-Accuracy':sciuridae_score[2]},
index=['Performance'])
sciuridae_score_df.to_csv('MOUNT_DIRECTORY/sciuridae/output/sciuridae_score.c
sv')
```

```
sciuridae_predIdxs = sciuridae_model.predict(x=sciuridae_testGen)
print("Predictions finished")
sciuridae_predIdxs = np.argmax(sciuridae_predIdxs, axis=1)
sciuridae_report = classification_report(sciuridae_testGen.classes,
sciuridae_predIdxs,

target_names=sciuridae_testGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(sciuridae_report).transpose()
df.to_csv('MOUNT_DIRECTORY/sciuridae/output/sciuridae_evaluation.csv')

sciuridae_mistakes = pd.DataFrame.from_dict({"Prediction":
sciuridae_predIdxs, "Actual": sciuridae_testGen.classes})
sciuridae_mistakes.to_csv('MOUNT_DIRECTORY/sciuridae/output/sciuridae_mistake
s.csv')

sciuridae_oos_dir = 'MOUNT_DIRECTORY/sciuridae/out_of_sample'

sciuridae_oosGen = valAug.flow_from_directory(
    sciuridae_oos_dir,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=128)

sciuridae_oos_score = sciuridae_model.evaluate(sciuridae_oosGen)
print("Evaluation finished")
sciuridae_oos_score_df = pd.DataFrame({'Loss':sciuridae_oos_score[0], 'Top-1-
Accuracy':sciuridae_oos_score[1],
                                       'Top-5-
Accuracy':sciuridae_oos_score[2]}, index=['Performance'])
sciuridae_oos_score_df.to_csv('MOUNT_DIRECTORY/sciuridae/output/sciuridae_oos
_score.csv')
sciuridae_oos_predIdxs = sciuridae_model.predict(x=sciuridae_oosGen)
print("Predictions finished")
sciuridae_oos_predIdxs = np.argmax(sciuridae_oos_predIdxs, axis=1)
sciuridae_oos_report = classification_report(sciuridae_oosGen.classes,
sciuridae_oos_predIdxs,
                        labels = (0,1,2,3,4,5),
target_names=sciuridae_oosGen.class_indices.keys(), output_dict=True)
df = pd.DataFrame(sciuridae_oos_report).transpose()
df.to_csv('MOUNT_DIRECTORY/sciuridae/output/sciuridae_oos_evaluation.csv')

sciuridae_oos_mistakes = pd.DataFrame.from_dict({"Prediction":
sciuridae_oos_predIdxs,
                                                 "Actual":
sciuridae_oosGen.classes})
sciuridae_oos_mistakes.to_csv('MOUNT_DIRECTORY/sciuridae/output/sciuridae_oos
_mistakes.csv')
```

- *Evaluation*

```
# Import necessary libraries and modules

import pandas as pd
import matplotlib.pyplot as plt
import json
import numpy as np
import seaborn as sns
import dataframe_image as dfi
from sklearn.metrics import confusion_matrix

# Define font to be used for graphics and increase dataframe rows visible in
window

plt.rcParams["font.family"] = "Times New Roman"
pd.set_option('display.max_rows', 100)

# Function to read metrics from json files

def json_reader(file):
    with open(file) as json_data:
        json_dict = json.load(json_data)
    return json_dict

# Create dataframe of metrics for each model

benchmark_training_df = pd.DataFrame.from_dict(json_reader('benchmark.json'))
family_training_df = pd.DataFrame.from_dict(json_reader('family.json'))
canidae_training_df = pd.DataFrame.from_dict(json_reader('canidae.json'))
felinae_training_df = pd.DataFrame.from_dict(json_reader('felinae.json'))
sciuridae_training_df = pd.DataFrame.from_dict(json_reader('sciuridae.json'))

# Function to clean up dataframe

def dataframe_tidy(df):
    df.index = range(1,51)
    df.index.name='Epoch'
    df.columns = ["Training Loss", "Training Accuracy", "Top 5 Training
Accuracy",
                                  "Validation Loss", "Validation Accuracy",
"Validation Top 5 Accuracy"]
    return df

benchmark_training_df = dataframe_tidy(benchmark_training_df)
family_training_df = dataframe_tidy(family_training_df)
canidae_training_df = dataframe_tidy(canidae_training_df)
felinae_training_df = dataframe_tidy(felinae_training_df)
sciuridae_training_df = dataframe_tidy(sciuridae_training_df)

# Save dataframes as images for use in report

dfi.export(benchmark_training_df, 'results_images/benchmark_training_df.png')
dfi.export(family_training_df, 'results_images/family_training_df.png')
dfi.export(canidae_training_df, 'results_images/canidae_training_df.png')
```

```
dfi.export(felinae_training_df, 'results_images/felinae_training_df.png')
dfi.export(sciuridae_training_df, 'results_images/sciuridae_training_df.png')

# Get training/validation metrics for best epoch for each model

benchmark_best_epoch =
pd.DataFrame(benchmark_training_df.loc[48,]).transpose()
family_best_epoch = pd.DataFrame(family_training_df.loc[45,]).transpose()
canidae_best_epoch = pd.DataFrame(canidae_training_df.loc[43,]).transpose()
felinae_best_epoch = pd.DataFrame(felinae_training_df.loc[40,]).transpose()
sciuridae_best_epoch =
pd.DataFrame(sciuridae_training_df.loc[35,]).transpose()

# Combine into single dataframe for visualisation

best_epoch_df = pd.concat([benchmark_best_epoch, family_best_epoch,
                           canidae_best_epoch, felinae_best_epoch,
                           sciuridae_best_epoch])

# Tidy and save dataframe

best_epoch_df.reset_index(inplace=True)
best_epoch_df.rename(columns={'index':'Epoch'}, inplace=True)
best_epoch_df.index = ['Benchmark', 'Family-Level', 'Canidae',
                       'Felinae', 'Sciuridae']
best_epoch_df.index.name = 'Classifier'
best_epoch_df = best_epoch_df.round(2)
dfi.export(best_epoch_df, 'results_images/best_epoch_df.png')

# Function to plot training/validation progress

def train_val_plot(df, best_epoch, acc_text_y, acc_arrow_y, val_text_y,
val_arrow_y, filename):
    plt.style.use("ggplot")
    fig = plt.figure(figsize=(15.5,5.5))
    N = np.arange(1, len(df)+1)
    ax1 = fig.add_subplot(1,2,1)
    ax1.plot(N, df["Training Accuracy"], label="Training Accuracy")
    ax1.plot(N, df["Validation Accuracy"], label="Validation Accuracy")
    ax1.set_title("Training and Validation Accuracy", pad=20, size=18)
    ax1.set_xlabel("Epoch", size=14)
    ax1.set_ylabel("Accuracy", size=14)
    ax1.axvline(x=best_epoch, color='k', linestyle = 'dashed', linewidth =
0.5)
    ax1.annotate('Epoch used for evaluation', xy=(best_epoch-23, acc_text_y),
size=12)
    ax1.arrow(x=best_epoch-13, y=acc_arrow_y, dx=12, dy=0, color='k',
head_width=0.02,
              width=0.005, head_length=0.4, alpha=0.5)
    ax1.legend(fontsize=12)
    ax2 = fig.add_subplot(1,2,2)
    ax2.plot(N, benchmark_training_df["Training Loss"], label="Training
Loss")
    ax2.plot(N, benchmark_training_df["Validation Loss"], label="Validation
Loss")
    ax2.set_title("Training and Validation Loss", pad=20, size=18)
    ax2.set_xlabel("Epoch", size=14)
```

```
    ax2.set_ylabel("Loss", size=14)
    ax2.axvline(x=best_epoch, color='k', linestyle = 'dashed', linewidth =
0.5)
    ax2.annotate('Epoch used for evaluation', xy=(best_epoch-23, val_text_y),
size=12)
    ax2.arrow(x=best_epoch-13, y=val_arrow_y, dx=12, dy=0, color='k',
head_width=0.1,
              width=0.025, head_length=0.4, alpha=0.5)
    ax2.legend(fontsize=12)
    plt.subplots_adjust(left=None, bottom=None, right=None, top = None,
wspace=None, hspace = 0.3)
    plt.savefig(filename)
    plt.show()

# Plot and save as image file training/validation progress

train_val_plot(benchmark_training_df, 48, 0.55, 0.53, 2.25, 2.15,
"results_images/benchmark_plot.png")

train_val_plot(family_training_df, 45, 0.55, 0.53, 2.25, 2.15,
"results_images/family_plot.png")

train_val_plot(canidae_training_df, 43, 0.55, 0.53, 2.25, 2.15,
"results_images/canidae_plot.png")

train_val_plot(felinae_training_df, 40, 0.55, 0.53, 2.25, 2.15,
"results_images/felinae_plot.png")

train_val_plot(sciuridae_training_df, 35, 0.925, 0.915, 2.25, 2.15,
"results_images/sciuridae_plot.png")

# Function to create dataframe showing test performance by model

def test_scores(file_list):
    df = pd.read_csv(file_list[0])
    for file in file_list[1:]:
        test_scores_df = pd.concat([df,pd.read_csv(file)])
        df = test_scores_df
    test_scores_df.drop(columns='Unnamed: 0', inplace=True)
    test_scores_df.index = ['Benchmark', 'Family-Level', 'Canidae',
                            'Felinae', 'Sciuridae']
    test_scores_df.index.name = 'Classifier'
    test_scores_df = test_scores_df.round(2)
    return test_scores_df

test_scores_df = test_scores(["./Results/output_benchmark_score.csv",
"./Results/family_data_output_score.csv",

"./Results/canidae_output_canidae_score.csv","./Results/felinae_output_felina
e_score.csv",
                  "./Results/sciuridae_output_sciuridae_score.csv"])
dfi.export(test_scores_df, 'results_images/test_scores_df.png')

#Function to plot test performance

def test_plot(df, filename):
    df_melted = df.drop(columns='Loss')
```

```
    df_melted = df_melted.reset_index()
    df_melted = pd.melt(df_melted, id_vars='Classifier')
    df_melted.columns = ['Classifier', 'Metric', 'Accuracy']
    fig = plt.figure(figsize=(12,7))
    ax = sns.barplot(y='Classifier', x='Accuracy', hue='Metric',
data=df_melted, orient='h', palette='deep')
    ax.set_title('Test Accuracy', pad=20, size=18)
    ax.set_xlim(0, 1.3)
    ax.set_xlabel('Accuracy', size=14, weight='bold')
    ax.set_ylabel('Classifier', size=14, weight='bold')
    for container in ax.containers:
        ax.bar_label(container, padding=5, fmt='%.2f', size=11)
    ax.legend(fontsize=12)
    plt.savefig(filename)
    plt.show()

test_plot(test_scores_df, "results_images/test_scores_plot.png")

# Function to create two dataframes: (1) precision, recall and F1 score by
species (2) weighted average
# across all species

def test_evaluation(file, index_label, name):
    df = pd.read_csv(file)
    df.columns=[index_label, "Precision", "Recall", "F1-Score", "No. Images"]
    df.set_index(index_label, inplace=True)
    df = df.round(2)
    df2 = df.tail(1)
    df2.index=[name]
    df2.index.name = 'Classifier'
    df.drop(df.tail(3).index,inplace = True)
    return df, df2

benchmark_test_evaluation_df, benchmark_avg =
test_evaluation("./Results/output_benchmark_evaluation.csv",
                                                                'Species',
'Benchmark')
dfi.export(benchmark_test_evaluation_df,
'results_images/benchmark_test_evaluation_df.png')
family_test_evaluation_df, family_avg =
test_evaluation("./Results/family_data_output_evaluation.csv",
                                                        'Family', 'Family')
dfi.export(family_test_evaluation_df,
'results_images/family_test_evaluation_df.png')
canidae_test_evaluation_df, canidae_avg =
test_evaluation("./Results/canidae_output_canidae_evaluation.csv",
                                                            'Species',
'Canidae')
dfi.export(canidae_test_evaluation_df,
'results_images/canidae_test_evaluation_df.png')
felinae_test_evaluation_df, felinae_avg =
test_evaluation("./Results/felinae_output_felinae_evaluation.csv",
                                                            'Species',
'Felinae')
dfi.export(felinae_test_evaluation_df,
'results_images/felinae_test_evaluation_df.png')
```

```
sciuridae_test_evaluation_df, sciuridae_avg =
test_evaluation("./Results/sciuridae_output_sciuridae_evaluation.csv",
                                                                'Species',
'Sciuridae')
dfi.export(sciuridae_test_evaluation_df,
'results_images/sciuridae_test_evaluation_df.png')
weighted_average_df = pd.concat([benchmark_avg, family_avg,
                                 canidae_avg, felinae_avg, sciuridae_avg])
dfi.export(weighted_average_df, 'results_images/weighted_average_df.png')

# Function to plot weighted average metrics

def weighted_average_plot(df, filename):
    df_melted = df.drop(columns='No. Images')
    df_melted = df_melted.reset_index()
    df_melted = pd.melt(df_melted, id_vars='Classifier')
    df_melted.columns = ['Classifier', 'Metric', 'Value']
    fig = plt.figure(figsize=(12,7))
    ax = sns.barplot(y='Classifier', x='Value', hue='Metric', data=df_melted,
orient='h', palette='deep')
    ax.set_title('Test Evaluation', pad=20, size=18)
    ax.set_xlim(0, 1.2)
    ax.set_xlabel('Value', size=14, weight='bold')
    ax.set_ylabel('Classifier', size=14, weight='bold')
    for container in ax.containers:
        ax.bar_label(container, padding=5, fmt='%.2f', size=11)
    ax.legend(fontsize=12)
    plt.savefig(filename)
    plt.show()

weighted_average_plot(weighted_average_df,
"results_images/weighted_average_plot.png")

# Function to create mappings between numeric labels and equivalent species
name

def create_mappings(df):
    mapping={}
    for i in range(0,len(df)):
        mapping[i] = df.index[i]
    return mapping

# Function to create:
# (1) dataframe of incorrect classifications
# (2) dataframe showing filenames of incorrect classifications
# (3) dataframe showing filenames of correct classifications
# (4) confusion matrix for predictions vs. truth

def test_mistakes(file, mapping):
    df = pd.read_csv(file, index_col=0)
    # reorder columns
    df = df[["Actual", "Prediction", "File"]]
    # Change integers to labels
    df['Prediction']=df['Prediction'].map(mapping)
    df['Actual']=df['Actual'].map(mapping)
    # Create separate dataframe of misclassified images
    misclassified_images = df[df["Prediction"] != df["Actual"]]
```

```
    # Create separate dataframe of correct classifications
    correct_classifications = df[df["Prediction"] == df["Actual"]]
    # Summarise mislcassifications and create confusion matrix
    df.drop(columns='File', inplace=True)
    conf_mat = confusion_matrix(df['Actual'], df['Prediction'])
    df = df.groupby(['Actual', 'Prediction']).size().reset_index()
    df.rename(columns={0:'Count'}, inplace=True)
    return df, misclassified_images, correct_classifications, conf_mat

# Function to create and save labelled confusion matrix

def conf_matrix(matrix, labels, filename):
    sns.set(font_scale=1.2)
    plt.figure(figsize = (10,7))
    cf = sns.heatmap(matrix, cmap='Blues', annot=True, cbar=False,
            xticklabels = labels,  yticklabels = labels, fmt='g')
    cf.set_xticklabels(cf.get_xticklabels(), rotation=90)
    cf.set_yticklabels(cf.get_yticklabels(), rotation=0)
    cf.set_title('Predicted vs. Actual Class', pad=30, size=18)
    plt.xlabel("Prediction", weight='bold')
    plt.ylabel("Actual", weight='bold')
    plt.savefig(filename)

benchmark_mapping = create_mappings(benchmark_test_evaluation_df)
benchmark_mistakes_df, benchmark_mistake_files, benchmark_correct,
benchmark_conf_mat = test_mistakes("./Results/output_benchmark_mistakes.csv",

benchmark_mapping)
conf_matrix(benchmark_conf_mat, list(benchmark_test_evaluation_df.index),
"results_images/benchmark_conf_mat.png")

canidae_mapping = create_mappings(canidae_test_evaluation_df)
canidae_mistakes_df, canidae_mistake_files, canidae_correct, canidae_conf_mat
= test_mistakes("./Results/canidae_output_canidae_mistakes.csv",

canidae_mapping)
dfi.export(canidae_mistakes_df, 'results_images/canidae_mistakes_df.png')
conf_matrix(canidae_conf_mat, list(canidae_test_evaluation_df.index),
"results_images/canidae_conf_mat.png")

felinae_mapping = create_mappings(felinae_test_evaluation_df)
felinae_mistakes_df, felinae_mistake_files, felinae_correct, felinae_conf_mat
= test_mistakes("./Results/felinae_output_felinae_mistakes.csv",

felinae_mapping)
dfi.export(felinae_mistakes_df, 'results_images/felinae_mistakes_df.png')
conf_matrix(felinae_conf_mat, list(felinae_test_evaluation_df.index),
"results_images/felinae_conf_mat.png")

sciuridae_mapping = create_mappings(sciuridae_test_evaluation_df)
sciuridae_mistakes_df, sciuridae_mistake_files, sciuridae_correct,
sciuridae_conf_mat =
test_mistakes("./Results/sciuridae_output_sciuridae_mistakes.csv",

sciuridae_mapping)
dfi.export(sciuridae_mistakes_df, 'results_images/sciuridae_mistakes_df.png')
```

```
conf_matrix(sciuridae_conf_mat, list(sciuridae_test_evaluation_df.index),
"results_images/sciuridae_conf_mat.png")

# Repeat entire test process for out-of-sample datasets


def oos_scores(file_list):
    df = pd.read_csv(file_list[0])
    for file in file_list[1:]:
        oos_scores_df = pd.concat([df,pd.read_csv(file)])
        df = oos_scores_df
    oos_scores_df.drop(columns='Unnamed: 0', inplace=True)
    oos_scores_df.index = ['Benchmark', 'Canidae',
                           'Felinae', 'Sciuridae']
    oos_scores_df.index.name = 'Classifier'
    return oos_scores_df

oos_scores_df = oos_scores(["./Results/output_benchmark_oos_score.csv",
                  "./Results/canidae_output_canidae_oos_score.csv",
                  "./Results/felinae_output_felinae_oos_score.csv",
                  "./Results/sciuridae_output_sciuridae_oos_score.csv"])
dfi.export(oos_scores_df, 'results_images/oos_scores_df.png')

# Function to plot out-of-sample scores

def oos_plot(df, filename):
    df_melted = df.drop(columns='Loss')
    df_melted = df_melted.reset_index()
    df_melted = pd.melt(df_melted, id_vars='Classifier')
    df_melted.columns = ['Classifier', 'Metric', 'Accuracy']
    fig = plt.figure(figsize=(12,7))
    ax = sns.barplot(y='Classifier', x='Accuracy', hue='Metric',
data=df_melted, orient='h', palette='deep')
    ax.set_title('Out-of-Sample Accuracy', pad=20, size=18)
    ax.set_xlim(0, 1.3)
    ax.set_xlabel('Accuracy', size=14, weight='bold')
    ax.set_ylabel('Classifier', size=14, weight='bold')
    for container in ax.containers:
        ax.bar_label(container, padding=5, fmt='%.2f', size=11)
    ax.legend(fontsize=12)
    plt.savefig(filename)
    plt.show()

oos_plot(oos_scores_df, "results_images/oos_scores_plot.png")

# Use test function to create equivalent dataframes for out-of-sample data

benchmark_oos_evaluation_df, benchmark_oos_avg =
test_evaluation("./Results/output_benchmark_oos_evaluation.csv",
                                                          'Species',
'Benchmark')
canidae_oos_evaluation_df, canidae_oos_avg =
test_evaluation("./Results/canidae_output_canidae_oos_evaluation.csv",
                                                      'Species',
'Canidae')
felinae_oos_evaluation_df, felinae_oos_avg =
test_evaluation("./Results/felinae_output_felinae_oos_evaluation.csv",
```

```
                                                        'Species',
'Felinae')
sciuridae_oos_evaluation_df, sciuridae_oos_avg =
test_evaluation("./Results/sciuridae_output_sciuridae_oos_evaluation.csv",
                                                            'Species',
'Sciuridae')

dfi.export(benchmark_oos_evaluation_df,
'results_images/benchmark_oos_evaluation_df.png')
dfi.export(canidae_oos_evaluation_df,
'results_images/canidae_oos_evaluation_df.png')
dfi.export(felinae_oos_evaluation_df,
'results_images/felinae_oos_evaluation_df.png')
dfi.export(sciuridae_oos_evaluation_df,
'results_images/sciuridae_oos_evaluation_df.png')

weighted_average_oos_df = pd.concat([benchmark_oos_avg, canidae_oos_avg,
                                     felinae_oos_avg, sciuridae_oos_avg])
dfi.export(weighted_average_oos_df,
'results_images/weighted_average_oos_df.png')

def weighted_average_oos_plot(df, filename):
    df_melted = df.drop(columns='No. Images')
    df_melted = df_melted.reset_index()
    df_melted = pd.melt(df_melted, id_vars='Classifier')
    df_melted.columns = ['Classifier', 'Metric', 'Value']
    fig = plt.figure(figsize=(12,7))
    ax = sns.barplot(y='Classifier', x='Value', hue='Metric', data=df_melted,
orient='h', palette='deep')
    ax.set_title('Out-of-Sample Evaluation', pad=20, size=18)
    ax.set_xlim(0, 1.3)
    ax.set_xlabel('Value', size=14, weight='bold')
    ax.set_ylabel('Classifier', size=14, weight='bold')
    for container in ax.containers:
        ax.bar_label(container, padding=5, fmt='%.2f', size=11)
    ax.legend(fontsize=12)
    plt.savefig(filename)
    plt.show()

weighted_average_oos_plot(weighted_average_oos_df,
"results_images/weighted_average_oos_plot.png")

def oos_mistakes(file, mapping):
    df = pd.read_csv(file, index_col=0)
    # reorder columns
    df = df[["Actual", "Prediction", "File"]]
    # Create separate dataframe of misclassified images
    misclassified_images = df[df["Prediction"] != df["Actual"]]
    # Change integers to labels

misclassified_images['Prediction']=misclassified_images['Prediction'].map(map
ping)

misclassified_images['Actual']=misclassified_images['Actual'].map(mapping)
    # Create separate dataframe of correct classifications
    correct_classifications = df[df["Prediction"] == df["Actual"]]
    # Change integers to labels
```

```
correct_classifications['Prediction']=correct_classifications['Prediction'].m
ap(mapping)

correct_classifications['Actual']=correct_classifications['Actual'].map(mappi
ng)
    # Summarise misclassifications and create confusion matrix
    df.drop(columns='File', inplace=True)
    conf_mat = confusion_matrix(df['Actual'], df['Prediction'])
    mapping_list = list(mapping.keys())
    predictions_list = df['Prediction'].unique()
    # correct size of confusion matrix to account for values not predicted
    missing_list = [item for item in mapping_list if item not in
predictions_list]
    for val in missing_list:
        conf_mat = np.insert(conf_mat, val, np.zeros((1,conf_mat.shape[0])),
0)
        conf_mat = np.insert(conf_mat, val, np.zeros((1,1)), 1)
    df = df.groupby(['Actual', 'Prediction']).size().reset_index()
    df.rename(columns={0:'Count'}, inplace=True)
    df['Prediction']=df['Prediction'].map(mapping)
    df['Actual']=df['Actual'].map(mapping)
    return df, misclassified_images, correct_classifications, conf_mat

benchmark_oos_mapping = create_mappings(benchmark_oos_evaluation_df)
canidae_oos_mapping = create_mappings(canidae_oos_evaluation_df)
felinae_oos_mapping = create_mappings(felinae_oos_evaluation_df)
sciuridae_oos_mapping = create_mappings(sciuridae_oos_evaluation_df)

benchmark_oos_mistakes_df, benchmark_oos_mistake_files,
benchmark_oos_correct, benchmark_oos_conf_mat =
oos_mistakes("./Results/output_benchmark_oos_mistakes.csv",

benchmark_oos_mapping)
canidae_oos_mistakes_df, canidae_oos_mistake_files, canidae_oos_correct,
canidae_oos_conf_mat =
oos_mistakes("./Results/canidae_output_canidae_oos_mistakes.csv",

canidae_oos_mapping)
felinae_oos_mistakes_df, felinae_oos_mistake_files, felinae_oos_correct,
felinae_oos_conf_mat =
oos_mistakes("./Results/felinae_output_felinae_oos_mistakes.csv",

felinae_oos_mapping)
sciuridae_oos_mistakes_df, sciuridae_oos_mistake_files,
sciuridae_oos_correct, sciuridae_oos_conf_mat =
oos_mistakes("./Results/sciuridae_output_sciuridae_oos_mistakes.csv",

sciuridae_oos_mapping)

dfi.export(benchmark_oos_mistakes_df,
'results_images/benchmark_oos_mistakes_df.png')
conf_matrix(benchmark_oos_conf_mat, list(benchmark_test_evaluation_df.index),
            "results_images/benchmark_oos_conf_mat.png")
dfi.export(canidae_oos_mistakes_df,
'results_images/canidae_oos_mistakes_df.png')
conf_matrix(canidae_oos_conf_mat, list(canidae_test_evaluation_df.index),
```

```
            "results_images/canidae_oos_conf_mat.png")
dfi.export(felinae_oos_mistakes_df,
'results_images/felinae_oos_mistakes_df.png')
conf_matrix(felinae_oos_conf_mat, list(felinae_test_evaluation_df.index),
            "results_images/felinae_oos_conf_mat.png")
dfi.export(sciuridae_oos_mistakes_df,
'results_images/sciuridae_oos_mistakes_df.png')
conf_matrix(sciuridae_oos_conf_mat, list(sciuridae_test_evaluation_df.index),
            "results_images/sciuridae_oos_conf_mat.png")

# Function to choose random images from dataframe

def random_files(df_list):
    df = df_list[0].sample(1)
    for dataframe in df_list[1:]:
        new_df = pd.concat([df,dataframe.sample(1)])
        df = new_df
    return new_df

# Use function to create dataframe of misclassified files

misclassified_files = random_files([benchmark_mistake_files,
benchmark_oos_mistake_files, canidae_mistake_files,
            canidae_oos_mistake_files, felinae_mistake_files,
felinae_oos_mistake_files,
            sciuridae_mistake_files, sciuridae_oos_mistake_files])

# Change display settings so entire dataframe is visible

misclassified_files.style.set_properties(subset=['File'], **{'width-min':
'300px'})

# Repeat with correct classifications

correct_files = random_files([benchmark_correct, benchmark_oos_correct,
canidae_correct,
            canidae_oos_correct, felinae_correct, felinae_oos_correct,
            sciuridae_correct, sciuridae_oos_correct])

correct_files.style.set_properties(subset=['File'], **{'width-min': '300px'})
```

- *AzCopy Download Example*

```
PS C:\Users\jrlee> cd azcopy
PS C:\Users\jrlee\azcopy> azcopy cp "https://lilablobssc.blob.core.windows.net/wellington-unzipped/images?st=2020-01-01T
00%3A00%3A00Z&se=2034-01-01T00%3A00%3A00Z&sp=rl&sv=2019-07-07&sr=c&sig=ohyGrUrcFn0qz3D5fSquQqlGeZ6IY%2BYQFc1iL72z5lQ%3D"
 "F:\Wellington" --include-path
```

- *CloudShell mounting bucket*

```
jrlee_85@tensorflow-2-3-20210807-160127:~$ cd /home/jupyter/
jrlee_85@tensorflow-2-3-20210807-160127:/home/jupyter$ sudo -s su jupyter
jupyter@tensorflow-2-3-20210807-160127:~$ mkdir MOUNT_DIRECTORY
jupyter@tensorflow-2-3-20210807-160127:~$ /usr/bin/gcsfuse --implicit-dirs dsm500_bucket_europe_west2_a MOUNT_DIRECTORY
```

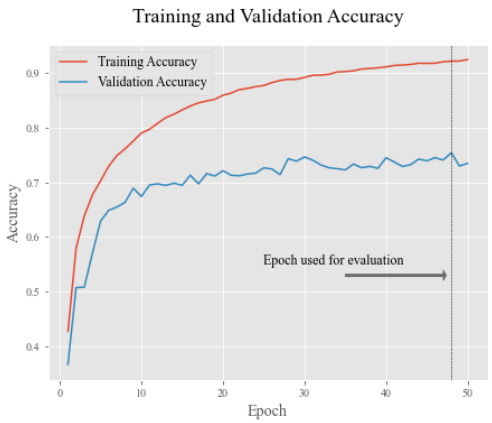
- *Gsutil downloading example images*

```
C:\Users\jrlee\Correct>gsutil cp "gs://dsm500_bucket_europe_west2_a/sciuridae/out_of_sample/American red squirrel/SEQ85150_IMG_0003.JPG" .
Copying gs://dsm500_bucket_europe_west2_a/sciuridae/out_of_sample/American red squirrel/SEQ85150_IMG_0003.JPG...
/ [1 files][203.3 KiB/203.3 KiB]
Operation completed over 1 objects/203.3 KiB.
```

iii. Results

- *Benchmark Model Training*

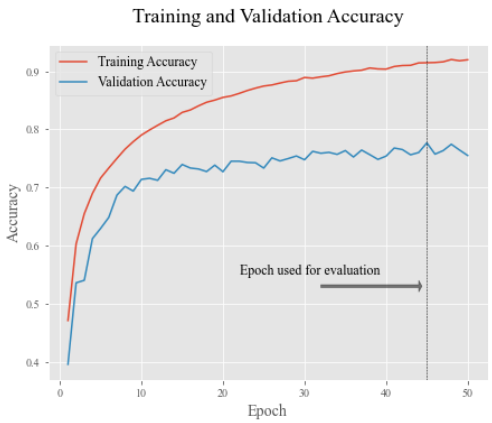
Epoch	Training Loss	Training Accuracy	Top 5 Training Accuracy	Validation Loss	Validation Accuracy	Validation Top 5 Accuracy
1	2.233504	0.427863	0.719287	3.635324	0.367537	0.688606
2	1.489553	0.579988	0.861098	2.379183	0.507829	0.814258
3	1.251493	0.638600	0.896141	2.294865	0.508339	0.817059
4	1.104557	0.677060	0.918200	1.844198	0.570719	0.859834
5	0.999506	0.702966	0.932456	1.446183	0.629281	0.890134
6	0.905253	0.729681	0.943056	1.382157	0.649013	0.896499
7	0.832208	0.748959	0.950955	1.317328	0.654869	0.903119
8	0.776699	0.761355	0.958790	1.331935	0.663526	0.901719
9	0.726998	0.775484	0.963558	1.181291	0.689370	0.918014
10	0.678395	0.790026	0.967658	1.230029	0.674220	0.914704
11	0.647952	0.796971	0.971107	1.157037	0.695226	0.918778
12	0.615067	0.808064	0.973062	1.151507	0.697263	0.916104
13	0.580353	0.818378	0.976622	1.158041	0.694462	0.919032
14	0.553129	0.824545	0.979165	1.148735	0.698409	0.920687
15	0.530162	0.832412	0.979705	1.201346	0.694589	0.915850
16	0.506339	0.839484	0.981914	1.121355	0.713049	0.923488
17	0.487430	0.845205	0.983424	1.228056	0.697263	0.915977
18	0.471752	0.848781	0.985156	1.119981	0.715850	0.923743
19	0.458501	0.851705	0.985474	1.142425	0.711649	0.924125
20	0.438517	0.859238	0.986825	1.116205	0.721197	0.923870
21	0.424118	0.863053	0.988239	1.151035	0.712922	0.925143
22	0.407913	0.869537	0.988764	1.189015	0.711903	0.920815
23	0.396808	0.871635	0.989400	1.140598	0.715213	0.927053
24	0.389708	0.874829	0.990226	1.182377	0.716741	0.925652
25	0.376541	0.877054	0.991259	1.138491	0.726671	0.929854
26	0.363391	0.882235	0.991307	1.189055	0.724379	0.927180
27	0.352482	0.886192	0.992070	1.196564	0.714067	0.919287
28	0.347827	0.888243	0.991958	1.110202	0.743348	0.933673
29	0.343664	0.888417	0.992912	1.115894	0.738638	0.928071
30	0.328499	0.891914	0.993103	1.097155	0.746531	0.933800
31	0.323323	0.895633	0.993166	1.113462	0.740547	0.930363
32	0.318925	0.896014	0.993738	1.152122	0.731890	0.929726
33	0.314836	0.897460	0.993627	1.196907	0.726671	0.927562
34	0.298912	0.901720	0.994215	1.243175	0.725016	0.928835
35	0.295012	0.902514	0.994835	1.215419	0.722724	0.927307
36	0.289537	0.903897	0.995121	1.182678	0.733418	0.932272
37	0.284654	0.907203	0.994803	1.180721	0.726798	0.924761
38	0.281415	0.908220	0.995185	1.221042	0.729217	0.926034
39	0.275745	0.909348	0.995200	1.232888	0.725652	0.922597
40	0.268761	0.911096	0.996090	1.089027	0.745003	0.934055
41	0.264776	0.913655	0.996011	1.168791	0.737237	0.931381
42	0.261371	0.914402	0.995979	1.219211	0.729090	0.928199
43	0.258336	0.915546	0.996074	1.202633	0.732145	0.928835
44	0.252201	0.917708	0.996281	1.179471	0.742584	0.929854
45	0.247489	0.917374	0.996472	1.252920	0.739274	0.927180
46	0.250866	0.917724	0.995677	1.191165	0.745130	0.934055
47	0.242153	0.920346	0.996472	1.277209	0.740929	0.931127
48	0.237383	0.921649	0.997044	1.152496	0.754042	0.938129
49	0.237643	0.921681	0.996631	1.269941	0.729854	0.926671
50	0.233107	0.924510	0.996949	1.242765	0.734819	0.930363

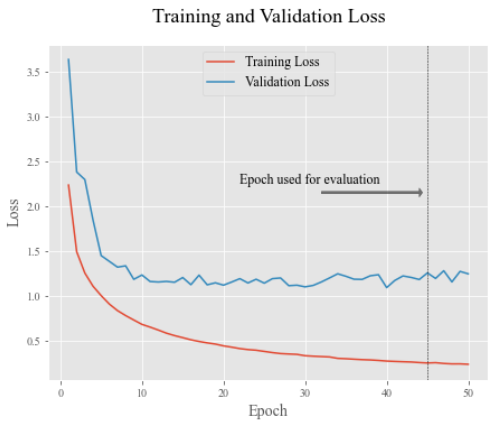

Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Epoch used for evaluation
Accuracy
Epoch


Training and Validation Loss
Training Loss
Validation Loss
Epoch used for evaluation
Loss
Epoch

- *Family Model Training*

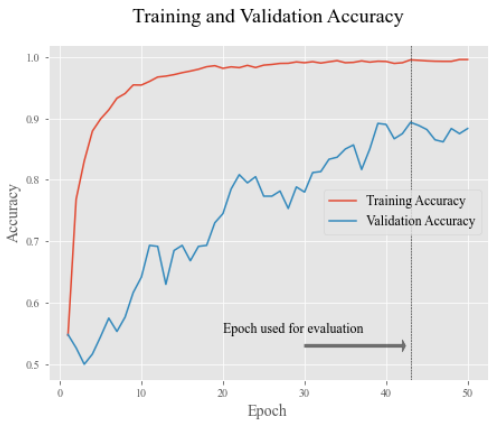
Epoch	Training Loss	Training Accuracy	Top 5 Training Accuracy	Validation Loss	Validation Accuracy	Validation Top 5 Accuracy
1	1.788266	0.470948	0.822796	2.694490	0.395470	0.818679
2	1.252757	0.603001	0.912002	1.817263	0.535819	0.885609
3	1.074255	0.654858	0.934792	1.645940	0.540145	0.892862
4	0.965130	0.689775	0.946505	1.321178	0.611910	0.921746
5	0.874852	0.716331	0.956677	1.271453	0.629469	0.923909
6	0.816988	0.733543	0.962573	1.212957	0.648429	0.933070
7	0.762188	0.750135	0.967515	1.044444	0.686856	0.943377
8	0.717203	0.766075	0.971266	0.955443	0.701870	0.947830
9	0.671407	0.778949	0.975144	1.019153	0.693854	0.946431
10	0.638950	0.790328	0.977544	0.949465	0.713959	0.948212
11	0.605339	0.799100	0.980293	0.967467	0.715867	0.951266
12	0.579289	0.807174	0.982121	1.025264	0.712432	0.944777
13	0.553975	0.814930	0.983996	0.927639	0.730627	0.954956
14	0.538060	0.819809	0.984600	0.961530	0.724520	0.953684
15	0.508994	0.829344	0.986634	0.896561	0.739662	0.952157
16	0.494444	0.833572	0.987445	0.940213	0.733554	0.953175
17	0.478115	0.840612	0.988525	0.929343	0.731900	0.953938
18	0.457724	0.846874	0.990035	0.971485	0.727446	0.951393
19	0.446418	0.850561	0.989670	0.912926	0.738516	0.954447
20	0.431010	0.855202	0.990750	1.008724	0.727192	0.950248
21	0.419731	0.857745	0.991895	0.913341	0.745260	0.953684
22	0.411445	0.862163	0.992403	0.916745	0.745260	0.954829
23	0.393874	0.867089	0.992070	0.942566	0.743097	0.955847
24	0.384236	0.871221	0.992991	0.992163	0.742588	0.952920
25	0.370937	0.874797	0.993150	0.986018	0.733427	0.949485
26	0.363051	0.876657	0.993579	0.938319	0.751113	0.958392
27	0.358931	0.879978	0.993834	0.956443	0.745769	0.956101
28	0.347490	0.882966	0.994310	0.926486	0.749841	0.955465
29	0.340153	0.883904	0.995248	0.932184	0.754294	0.957755
30	0.330307	0.889546	0.995073	0.985264	0.747805	0.954702
31	0.325341	0.888370	0.995645	0.907102	0.762311	0.959028
32	0.318989	0.890642	0.995439	0.935777	0.759002	0.957883
33	0.313694	0.892438	0.995852	0.921665	0.760529	0.959028
34	0.301845	0.896253	0.996202	0.968620	0.757094	0.959155
35	0.298079	0.899065	0.995836	0.906899	0.763583	0.960936
36	0.293954	0.900782	0.996186	0.968471	0.752513	0.957628
37	0.288269	0.902069	0.996774	0.934448	0.764474	0.959919
38	0.281004	0.905883	0.996631	0.984606	0.756585	0.956738
39	0.280183	0.904294	0.996678	1.036664	0.748441	0.953556
40	0.277468	0.903929	0.996821	0.944315	0.754040	0.954829
41	0.265480	0.908490	0.997123	0.938441	0.768037	0.957374
42	0.263149	0.910143	0.997171	0.974771	0.765365	0.958137
43	0.259634	0.910492	0.997012	0.979625	0.756330	0.955592
44	0.249902	0.914640	0.997505	0.955285	0.760275	0.960173
45	0.250539	0.914958	0.997251	0.936836	0.776816	0.959282
46	0.247516	0.915324	0.997759	0.979918	0.757475	0.956992
47	0.246250	0.916516	0.997664	0.968886	0.763583	0.957628
48	0.235834	0.920489	0.998013	0.938817	0.774526	0.959028
49	0.241400	0.918343	0.997505	0.988654	0.764601	0.961064
50	0.232067	0.920044	0.997934	1.041352	0.755185	0.955592

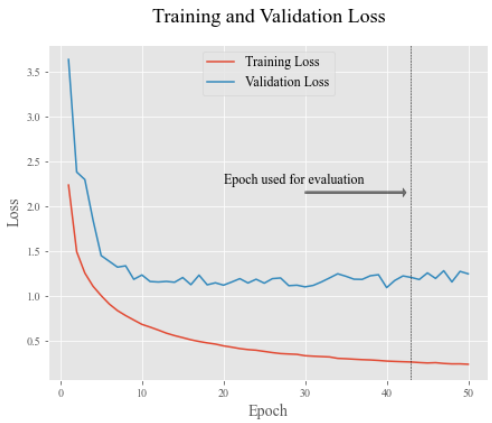

Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Epoch used for evaluation
Accuracy
Epoch
0.4
0.5
0.6
0.7
0.8
0.9
0
10
20
30
40
50


Training and Validation Loss
Training Loss
Validation Loss
Epoch used for evaluation
Loss
Epoch
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
10
20
30
40
50

- *Canidae Model Training*

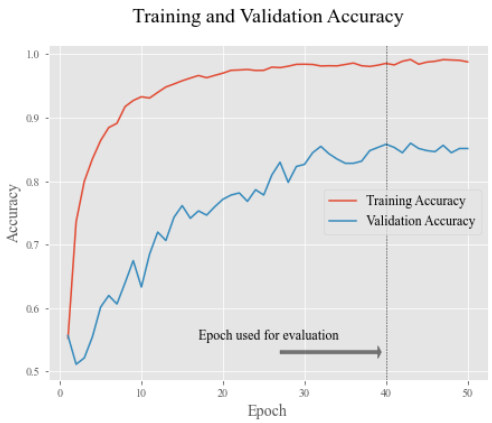
Epoch	Training Loss	Training Accuracy	Top 5 Training Accuracy	Validation Loss	Validation Accuracy	Validation Top 5 Accuracy
1	1.342556	0.547012	0.964811	1.767446	0.548333	0.985000
2	0.644680	0.768386	0.994973	1.975365	0.526667	0.986667
3	0.464424	0.830758	0.998511	2.294888	0.500000	0.990000
4	0.346886	0.879166	0.999069	2.173277	0.516667	0.990000
5	0.289487	0.898902	0.999255	1.882316	0.545000	0.991667
6	0.247029	0.913610	0.999814	1.802207	0.575000	0.993333
7	0.187208	0.932601	0.999814	1.957799	0.553333	0.995000
8	0.166660	0.940235	1.000000	1.857807	0.576667	0.996667
9	0.129439	0.954198	1.000000	1.464817	0.616667	0.998333
10	0.131983	0.954012	0.999814	1.369547	0.641667	0.996667
11	0.112040	0.959970	1.000000	1.256221	0.693333	0.996667
12	0.091080	0.967045	1.000000	1.195395	0.691667	1.000000
13	0.092038	0.968535	1.000000	1.681384	0.630000	0.998333
14	0.087055	0.970955	1.000000	1.645377	0.685000	0.995000
15	0.075614	0.974120	1.000000	1.589309	0.693333	0.996667
16	0.064881	0.976727	1.000000	1.661694	0.668333	0.998333
17	0.062167	0.979706	1.000000	1.566858	0.691667	1.000000
18	0.046913	0.983802	1.000000	1.527508	0.693333	1.000000
19	0.042727	0.985478	1.000000	1.239456	0.730000	1.000000
20	0.052699	0.981195	1.000000	1.193152	0.745000	1.000000
21	0.047575	0.983616	1.000000	1.039239	0.785000	0.996667
22	0.045240	0.982312	1.000000	0.773126	0.808333	1.000000
23	0.039311	0.986036	1.000000	0.774555	0.795000	1.000000
24	0.051797	0.982499	1.000000	0.725328	0.805000	1.000000
25	0.041431	0.986408	1.000000	0.989942	0.773333	0.998333
26	0.035266	0.987526	1.000000	0.981088	0.773333	0.996667
27	0.033124	0.989201	1.000000	1.008296	0.781667	0.998333
28	0.031797	0.989387	1.000000	1.180721	0.753333	1.000000
29	0.028423	0.991622	1.000000	0.984562	0.788333	1.000000
30	0.028017	0.990318	1.000000	1.188648	0.780000	1.000000
31	0.025849	0.991994	1.000000	0.797393	0.811667	1.000000
32	0.029074	0.989760	1.000000	0.891371	0.813333	1.000000
33	0.028418	0.991808	1.000000	0.752615	0.833333	1.000000
34	0.019374	0.993670	1.000000	0.789146	0.836667	1.000000
35	0.029400	0.990318	1.000000	0.669308	0.850000	1.000000
36	0.025523	0.990877	1.000000	0.555925	0.856667	1.000000
37	0.021858	0.993297	1.000000	0.762234	0.816667	1.000000
38	0.029598	0.991249	1.000000	0.578497	0.850000	1.000000
39	0.021783	0.992739	1.000000	0.423337	0.891667	1.000000
40	0.024150	0.992366	1.000000	0.463569	0.890000	0.998333
41	0.032962	0.989201	1.000000	0.497365	0.866667	1.000000
42	0.032065	0.990318	1.000000	0.460314	0.875000	1.000000
43	0.017090	0.994973	1.000000	0.411955	0.893333	1.000000
44	0.014826	0.994228	1.000000	0.443450	0.888333	1.000000
45	0.019240	0.993297	1.000000	0.506297	0.881667	1.000000
46	0.025633	0.992739	1.000000	0.527496	0.865000	1.000000
47	0.021615	0.992553	1.000000	0.548223	0.861667	1.000000
48	0.020384	0.992553	1.000000	0.564084	0.883333	1.000000
49	0.015969	0.995532	1.000000	0.531206	0.875000	1.000000
50	0.012163	0.995532	1.000000	0.462179	0.883333	1.000000

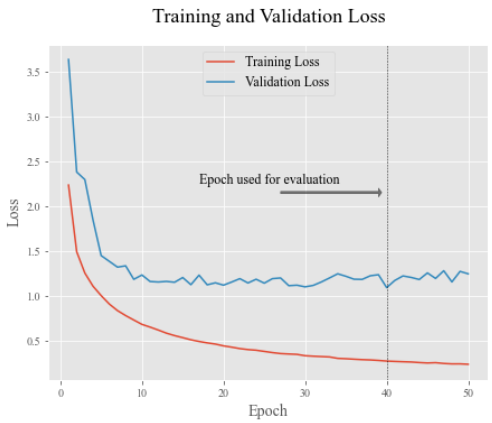

Training and Validation Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy
Epoch used for evaluation


Training and Validation Loss
Loss
Epoch
Training Loss
Validation Loss
Epoch used for evaluation

- *Felinae Model Training*

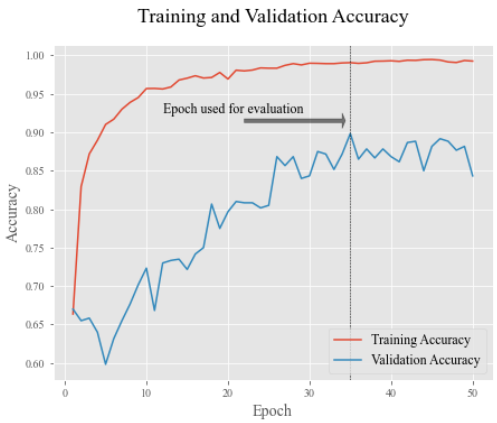
Epoch	Training Loss	Training Accuracy	Top 5 Training Accuracy	Validation Loss	Validation Accuracy	Validation Top 5 Accuracy
1	1.276604	0.552197	0.977564	1.803686	0.555927	0.979967
2	0.676927	0.735769	0.996106	2.321099	0.510851	0.956594
3	0.522721	0.799555	0.997960	2.427188	0.520868	0.966611
4	0.431400	0.834971	0.998887	2.162980	0.554257	0.974958
5	0.349708	0.863898	0.999629	1.931873	0.601002	0.984975
6	0.302106	0.884480	0.999629	1.948972	0.619366	0.993322
7	0.266235	0.891155	0.999815	2.045959	0.606010	0.988314
8	0.222969	0.917486	1.000000	1.898995	0.639399	0.986644
9	0.191700	0.927128	1.000000	1.556902	0.674457	0.991653
10	0.175073	0.933061	1.000000	1.877025	0.632721	0.988314
11	0.187372	0.931022	0.999815	1.387023	0.684474	0.993322
12	0.154862	0.939922	1.000000	1.121575	0.719533	0.993322
13	0.139447	0.948452	1.000000	1.162256	0.706177	0.989983
14	0.123801	0.953273	1.000000	0.934413	0.742905	0.994992
15	0.108936	0.958279	1.000000	0.938736	0.761269	0.991653
16	0.096003	0.962544	1.000000	1.030625	0.741235	0.994992
17	0.100309	0.966623	1.000000	1.081277	0.752922	0.996661
18	0.097349	0.963100	1.000000	1.096621	0.746244	0.996661
19	0.088176	0.966809	1.000000	1.148352	0.759599	0.991653
20	0.087072	0.970332	1.000000	1.043756	0.771285	0.994992
21	0.072235	0.974782	1.000000	0.948508	0.777963	0.993322
22	0.068568	0.975338	1.000000	1.059154	0.781302	0.991653
23	0.066562	0.976080	1.000000	0.999053	0.767947	0.991653
24	0.078715	0.974597	1.000000	1.055828	0.786310	0.994992
25	0.070642	0.974782	1.000000	1.112553	0.777963	0.993322
26	0.060881	0.979789	1.000000	0.855438	0.809683	0.993322
27	0.055272	0.979047	1.000000	0.789006	0.829716	0.998331
28	0.051794	0.981272	1.000000	0.861704	0.797997	0.996661
29	0.042864	0.984239	1.000000	0.758723	0.823038	0.998331
30	0.044898	0.984424	1.000000	0.769919	0.826377	1.000000
31	0.048530	0.984053	1.000000	0.644987	0.844741	1.000000
32	0.048030	0.981643	1.000000	0.717042	0.854758	1.000000
33	0.057155	0.982014	1.000000	0.742119	0.843072	1.000000
34	0.050795	0.981828	1.000000	0.757630	0.834725	1.000000
35	0.048240	0.984053	1.000000	0.833504	0.828047	0.996661
36	0.036377	0.986279	1.000000	0.793445	0.828047	1.000000
37	0.051773	0.982014	1.000000	0.818246	0.831386	1.000000
38	0.049391	0.980901	1.000000	0.730490	0.848080	0.998331
39	0.046833	0.982941	1.000000	0.701939	0.853088	0.998331
40	0.041857	0.985722	1.000000	0.699022	0.858097	0.998331
41	0.044976	0.983312	1.000000	0.746692	0.853088	1.000000
42	0.032207	0.989060	1.000000	0.846464	0.844741	1.000000
43	0.025546	0.991841	1.000000	0.787913	0.859766	1.000000
44	0.041955	0.984424	1.000000	0.706896	0.851419	1.000000
45	0.040206	0.987762	1.000000	0.732231	0.848080	0.998331
46	0.037387	0.989060	1.000000	0.723116	0.846411	0.998331
47	0.028027	0.991656	1.000000	0.701365	0.856427	0.998331
48	0.029594	0.991100	1.000000	0.787138	0.844741	0.998331
49	0.027168	0.990543	1.000000	0.783517	0.851419	0.996661
50	0.039980	0.988133	1.000000	0.734830	0.851419	0.998331


Training and Validation Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy
Epoch used for evaluation


Training and Validation Loss
Loss
Epoch
Training Loss
Validation Loss
Epoch used for evaluation

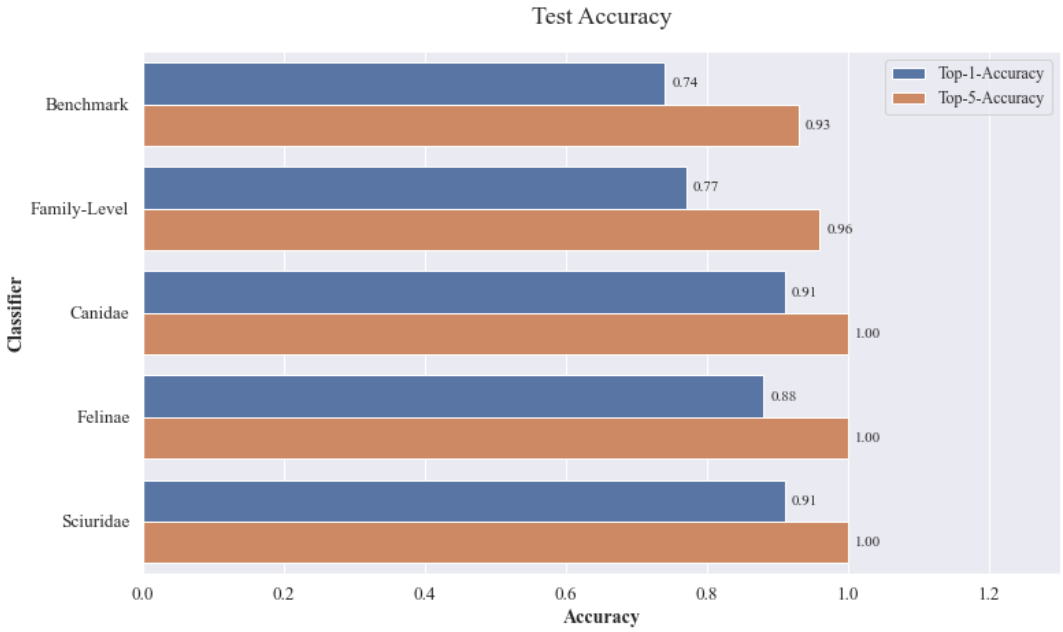
- *Sciuridae Model Training*

Epoch	Training Loss	Training Accuracy	Top 5 Training Accuracy	Validation Loss	Validation Accuracy	Validation Top 5 Accuracy
1	0.996958	0.663704	0.982593	1.334021	0.670000	0.991667
2	0.444612	0.829444	0.999259	1.826159	0.655000	0.973333
3	0.336548	0.871852	0.999815	1.998593	0.658333	0.975000
4	0.288461	0.889630	0.999815	2.425234	0.640000	0.985000
5	0.231435	0.910185	0.999815	2.468593	0.598333	0.981667
6	0.215881	0.917037	1.000000	2.458998	0.631667	0.966667
7	0.191797	0.930000	1.000000	1.800128	0.655000	0.996667
8	0.159491	0.938889	1.000000	1.564727	0.676667	0.996667
9	0.146553	0.945185	1.000000	1.611772	0.701667	0.990000
10	0.115702	0.956852	1.000000	1.248062	0.723333	0.998333
11	0.117189	0.957037	1.000000	1.484775	0.668333	0.995000
12	0.108340	0.956296	1.000000	1.118980	0.730000	0.996667
13	0.103656	0.958889	1.000000	1.131090	0.733333	0.998333
14	0.089191	0.967963	1.000000	1.136364	0.735000	0.998333
15	0.083760	0.970370	1.000000	1.197590	0.721667	0.998333
16	0.067608	0.973518	1.000000	1.183057	0.741667	1.000000
17	0.079339	0.970556	1.000000	1.154788	0.750000	1.000000
18	0.074750	0.971296	1.000000	0.847101	0.806667	1.000000
19	0.062567	0.977778	1.000000	1.126039	0.775000	1.000000
20	0.095557	0.969259	1.000000	1.129284	0.796667	0.998333
21	0.051937	0.980741	1.000000	0.999712	0.810000	0.998333
22	0.060271	0.979815	1.000000	0.892564	0.808333	0.998333
23	0.049279	0.980926	1.000000	0.848414	0.808333	0.998333
24	0.045618	0.983704	1.000000	0.936546	0.801667	0.998333
25	0.051309	0.983333	1.000000	0.805048	0.805000	0.998333
26	0.051700	0.983333	1.000000	0.535087	0.868333	1.000000
27	0.035378	0.987037	1.000000	0.544311	0.856667	1.000000
28	0.030318	0.989259	1.000000	0.545092	0.868333	1.000000
29	0.032929	0.987593	1.000000	0.621692	0.840000	0.998333
30	0.027737	0.989815	1.000000	0.654425	0.843333	0.998333
31	0.033339	0.989630	1.000000	0.462970	0.875000	0.998333
32	0.031589	0.989259	1.000000	0.552514	0.871667	1.000000
33	0.028468	0.989259	1.000000	0.618682	0.851667	1.000000
34	0.028640	0.990185	1.000000	0.455491	0.871667	1.000000
35	0.029142	0.990556	1.000000	0.377559	0.898333	1.000000
36	0.028237	0.989630	1.000000	0.523846	0.865000	1.000000
37	0.026449	0.990370	1.000000	0.541838	0.878333	1.000000
38	0.027695	0.992407	1.000000	0.593284	0.866667	1.000000
39	0.019809	0.992593	1.000000	0.526515	0.878333	1.000000
40	0.021274	0.992963	1.000000	0.550936	0.868333	1.000000
41	0.024565	0.992222	1.000000	0.581650	0.861667	1.000000
42	0.024343	0.993704	1.000000	0.397568	0.886667	1.000000
43	0.016798	0.993519	1.000000	0.449566	0.888333	1.000000
44	0.016753	0.994444	1.000000	0.740501	0.850000	1.000000
45	0.017463	0.994630	1.000000	0.458323	0.881667	1.000000
46	0.018493	0.993889	1.000000	0.458627	0.891667	1.000000
47	0.022327	0.991481	1.000000	0.539005	0.888333	1.000000
48	0.024046	0.990741	1.000000	0.550038	0.876667	1.000000
49	0.014886	0.993333	1.000000	0.574666	0.881667	1.000000
50	0.029877	0.992593	1.000000	0.763111	0.843333	1.000000

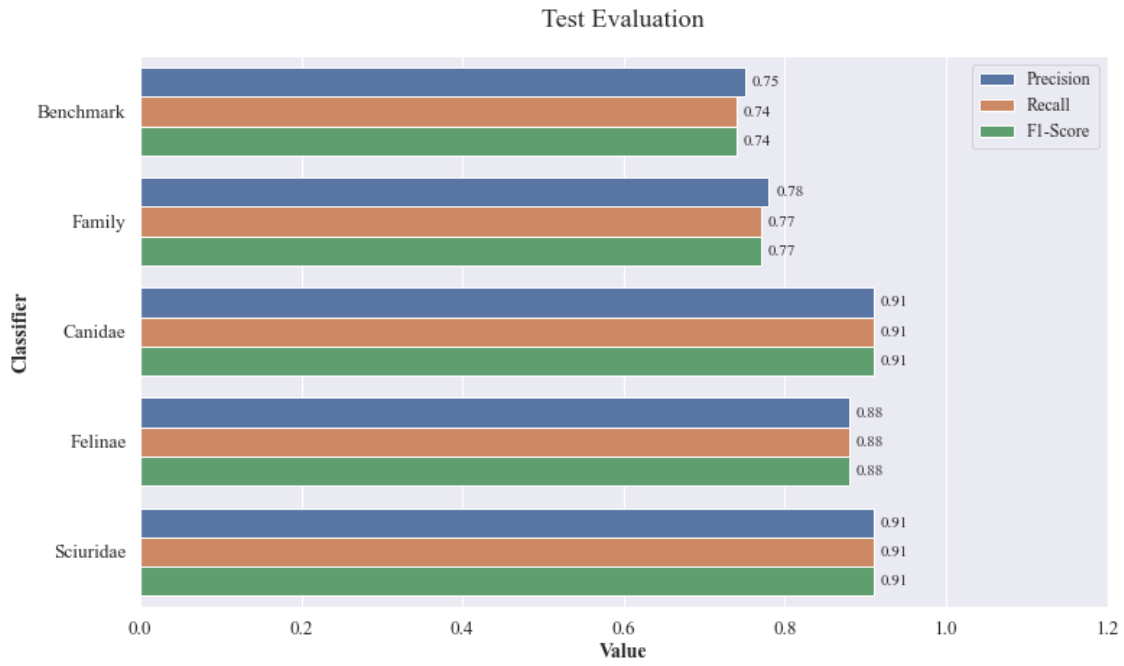


- *Model Testing*

Classifier	Loss	Top-1-Accuracy	Top-5-Accuracy
Benchmark	1.24	0.74	0.93
Family-Level	0.93	0.77	0.96
Canidae	0.60	0.91	1.00
Felinae	0.50	0.88	1.00
Sciuridae	0.42	0.91	1.00

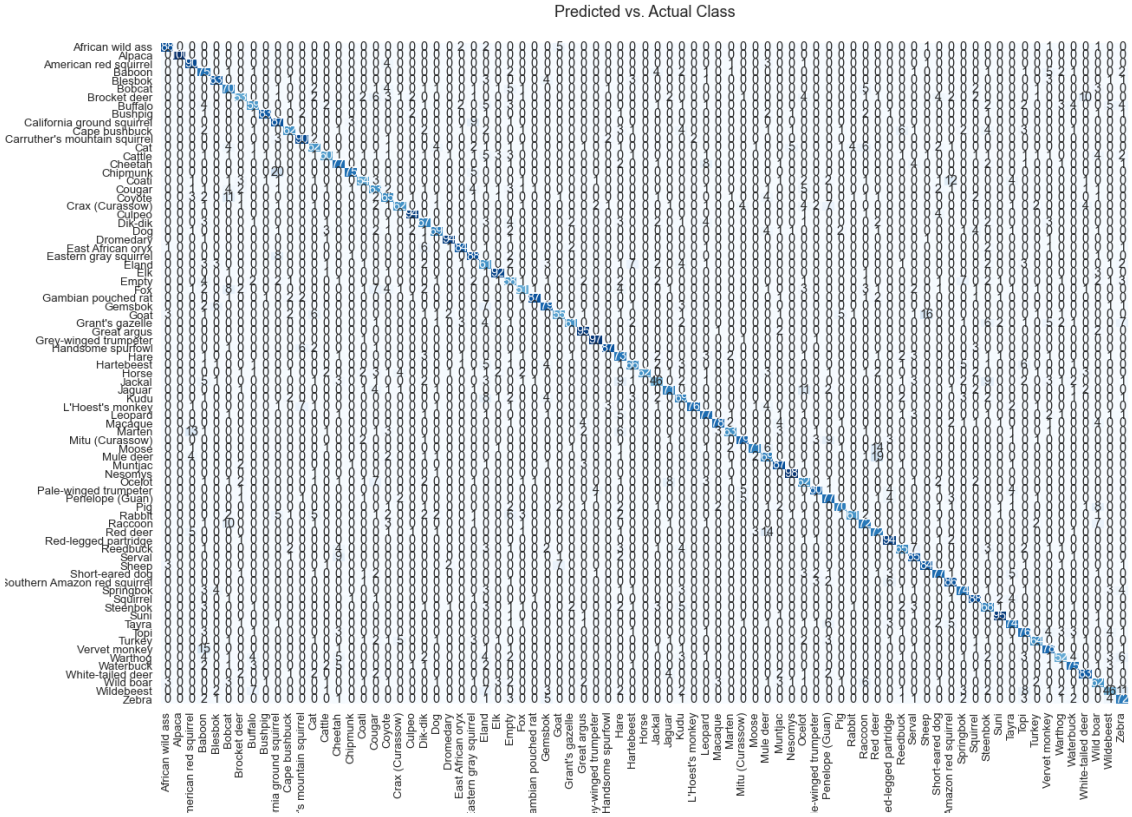


Classifier	Precision	Recall	F1-Score	No. Images
Benchmark	0.75	0.74	0.74	7882.0
Family	0.78	0.77	0.77	7879.0
Canidae	0.91	0.91	0.91	597.0
Felinae	0.88	0.88	0.88	598.0
Sciuridae	0.91	0.91	0.91	595.0



- *Benchmark Model Testing*

Species	Precision	Recall	F1-Score	No. Images
African wild ass	0.90	0.88	0.89	100.0
Alpaca	1.00	1.00	1.00	100.0
American red squirrel	0.74	0.90	0.81	100.0
Baboon	0.51	0.75	0.61	100.0
Blesbok	0.82	0.83	0.83	100.0
Bobcat	0.56	0.70	0.62	100.0
Brocket deer	0.71	0.53	0.61	100.0
Buffalo	0.69	0.59	0.64	100.0
Bushpig	0.95	0.83	0.89	100.0
California ground squirrel	0.65	0.87	0.75	100.0
Cape bushbuck	0.81	0.62	0.70	100.0
Carruther's mountain squirrel	0.83	0.90	0.87	100.0
Cat	0.67	0.63	0.65	99.0
Cattle	0.76	0.61	0.68	98.0
Cheetah	0.68	0.77	0.72	100.0
Chipmunk	0.91	0.75	0.82	100.0
Coati	0.76	0.59	0.67	91.0
Cougar	0.56	0.63	0.59	100.0
Coyote	0.62	0.64	0.63	101.0
Crax (Curassow)	0.82	0.62	0.70	100.0
Culpeo	0.90	0.94	0.92	100.0
Dik-dik	0.65	0.67	0.66	100.0
Dog	0.78	0.70	0.74	98.0
Dromedary	0.92	0.94	0.93	100.0
East African oryx	0.88	0.84	0.86	100.0
Eastern gray squirrel	0.71	0.88	0.79	100.0
Eland	0.43	0.60	0.50	101.0
Elk	0.86	0.92	0.89	100.0
Empty	0.50	0.58	0.54	100.0
Fox	0.71	0.51	0.59	100.0
Gambian pouched rat	0.95	0.87	0.91	100.0
Gemsbok	0.74	0.78	0.76	101.0
Goat	0.77	0.55	0.64	100.0
Grant's gazelle	0.88	0.61	0.72	100.0
Great argus	0.84	0.95	0.89	100.0
Grey-winged trumpeter	0.92	0.97	0.94	100.0
Handsome spurfowl	0.93	0.87	0.90	100.0
Hare	0.56	0.73	0.63	100.0
Hartebeest	0.73	0.66	0.69	100.0
Horse	1.00	0.62	0.77	100.0
Jackal	0.64	0.46	0.53	101.0
Jaguar	0.78	0.71	0.74	100.0
Kudu	0.62	0.68	0.65	101.0
L'Hoest's monkey	0.97	0.76	0.85	100.0
Leopard	0.71	0.77	0.74	100.0
Macaque	0.90	0.78	0.83	100.0
Marten	0.84	0.63	0.72	100.0
Mitu (Curassow)	0.82	0.79	0.81	100.0
Moose	0.91	0.71	0.80	100.0
Mule deer	0.60	0.69	0.64	100.0
Muntjac	0.79	0.87	0.83	100.0
Nesomys	0.91	0.98	0.94	100.0
Ocelot	0.57	0.62	0.59	100.0
Pale-winged trumpeter	0.84	0.80	0.82	100.0
Penelope (Guan)	0.68	0.77	0.72	100.0
Pig	0.84	0.71	0.77	98.0
Rabbit	0.90	0.61	0.73	100.0
Raccoon	0.68	0.71	0.70	101.0
Red deer	0.61	0.72	0.66	100.0
Red-legged partridge	0.79	0.94	0.86	100.0
Reedbuck	0.74	0.65	0.69	100.0
Serval	0.72	0.85	0.78	100.0
Sheep	0.81	0.84	0.82	100.0
Short-eared dog	0.76	0.77	0.77	100.0
Southern Amazon red squirrel	0.73	0.86	0.79	100.0
Springbok	0.73	0.74	0.73	100.0
Squirrel	0.77	0.88	0.82	100.0
Steenbok	0.58	0.68	0.63	100.0
Suni	0.90	0.95	0.92	100.0
Tayra	0.72	0.74	0.73	100.0
Topi	0.67	0.75	0.71	101.0
Turkey	0.81	0.70	0.75	92.0
Vervet monkey	0.68	0.77	0.73	101.0
Warthog	0.71	0.53	0.60	99.0
Waterbuck	0.74	0.75	0.75	100.0
White-tailed deer	0.78	0.83	0.80	100.0
Wild boar	0.58	0.62	0.60	100.0
Wildebeest	0.62	0.46	0.53	99.0
Zebra	0.58	0.72	0.64	100.0

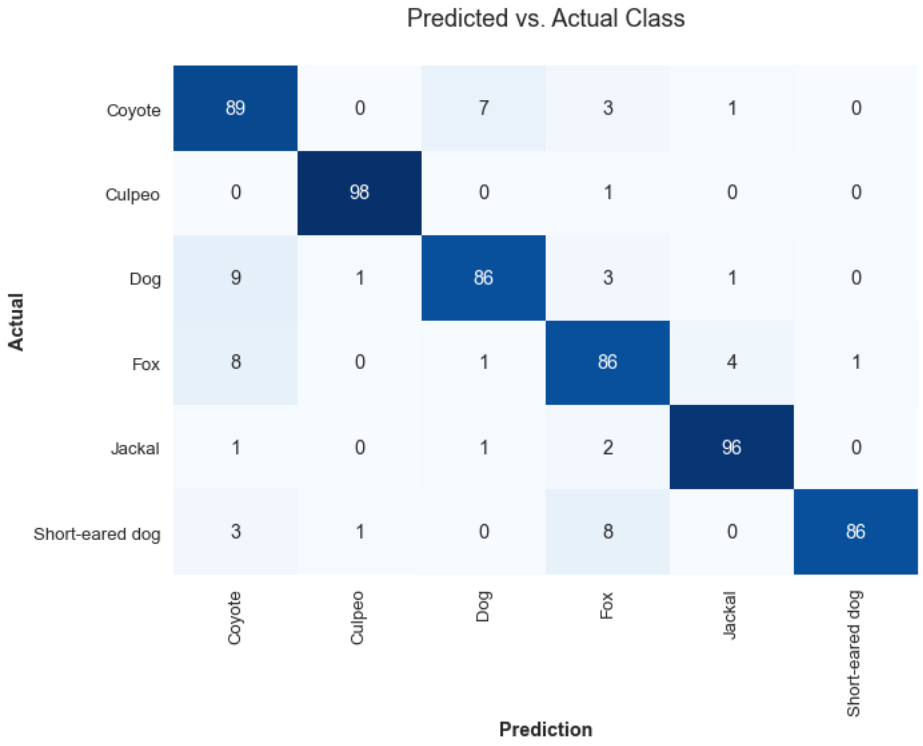

Predicted vs. Actual Class

- *Family Model Testing*

Family	Precision	Recall	F1-Score	No. Images
Alcelaphinae	0.78	0.75	0.77	399.0
Antilopinae	0.74	0.81	0.77	500.0
Bovinae	0.72	0.72	0.72	498.0
Camelidae	0.97	0.90	0.94	200.0
Canidae	0.73	0.69	0.71	597.0
Capreolinae	0.72	0.85	0.78	400.0
Caprinae	0.84	0.83	0.83	200.0
Cercopithecidae	0.85	0.76	0.80	401.0
Cervinae	0.89	0.69	0.78	300.0
Cracidae	0.88	0.80	0.84	302.0
Equidae	0.82	0.71	0.76	300.0
Felinae	0.64	0.75	0.69	600.0
Guloninae	0.73	0.81	0.77	200.0
Hippotraginae	0.81	0.77	0.79	201.0
Leporidae	0.77	0.56	0.65	200.0
Nesomyidae	0.88	0.95	0.92	200.0
None	0.61	0.55	0.58	100.0
Pantherinae	0.71	0.76	0.74	200.0
Phasianidae	0.89	0.82	0.85	392.0
Procyonidae	0.68	0.59	0.63	191.0
Psophiidae	0.85	0.86	0.85	200.0
Reduncinae	0.66	0.70	0.68	200.0
Sciuridae	0.84	0.93	0.88	700.0
Suidae	0.76	0.76	0.76	398.0

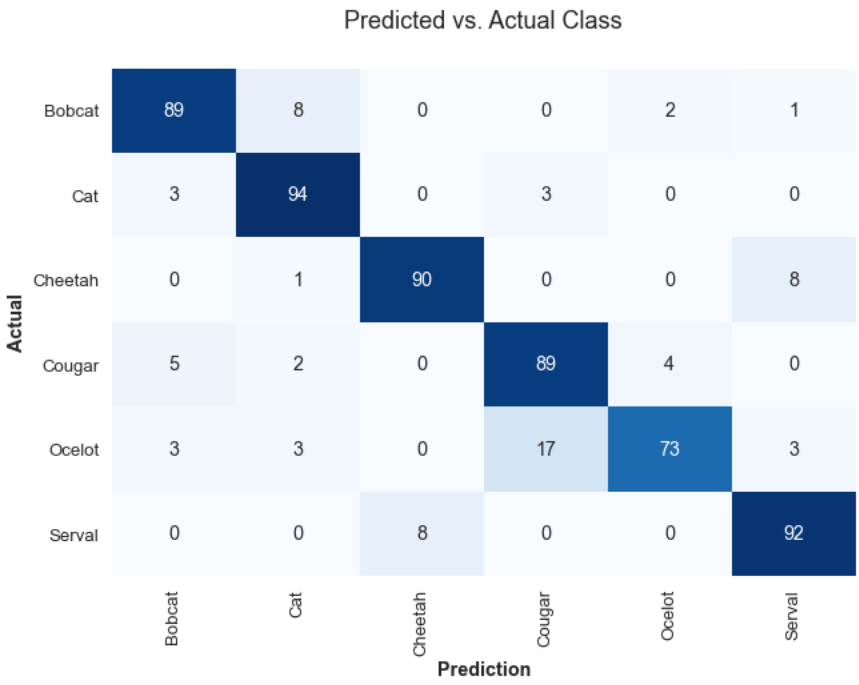
- *Canidae Model Testing*

Species	Precision	Recall	F1-Score	No. Images
Coyote	0.81	0.89	0.85	100.0
Culpeo	0.98	0.99	0.98	99.0
Dog	0.91	0.86	0.88	100.0
Fox	0.83	0.86	0.85	100.0
Jackal	0.94	0.96	0.95	100.0
Short-eared dog	0.99	0.88	0.93	98.0



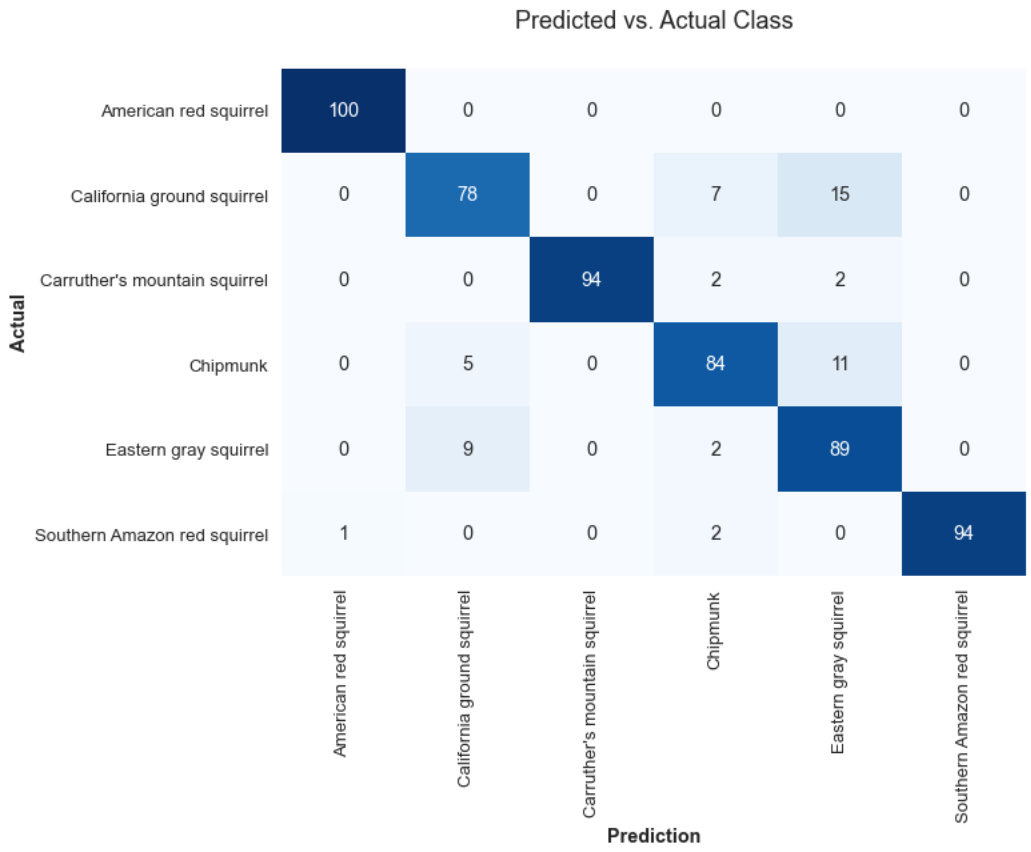
- *Felinae Model Testing*

Species	Precision	Recall	F1-Score	No. Images
Bobcat	0.89	0.89	0.89	100.0
Cat	0.87	0.94	0.90	100.0
Cheetah	0.92	0.91	0.91	99.0
Cougar	0.82	0.89	0.85	100.0
Ocelot	0.92	0.74	0.82	99.0
Serval	0.88	0.92	0.90	100.0



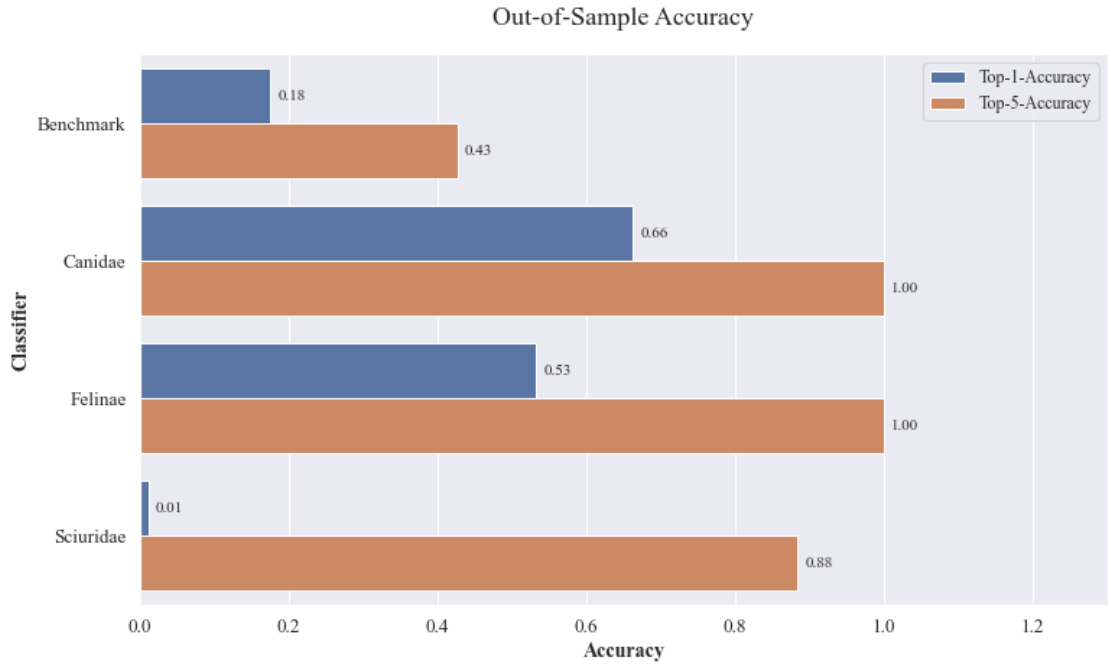
- *Sciuridae Model Testing*

Species	Precision	Recall	F1-Score	No. Images
American red squirrel	0.99	1.00	1.00	100.0
California ground squirrel	0.85	0.78	0.81	100.0
Carruther's mountain squirrel	1.00	0.96	0.98	98.0
Chipmunk	0.87	0.84	0.85	100.0
Eastern gray squirrel	0.76	0.89	0.82	100.0
Southern Amazon red squirrel	1.00	0.97	0.98	97.0

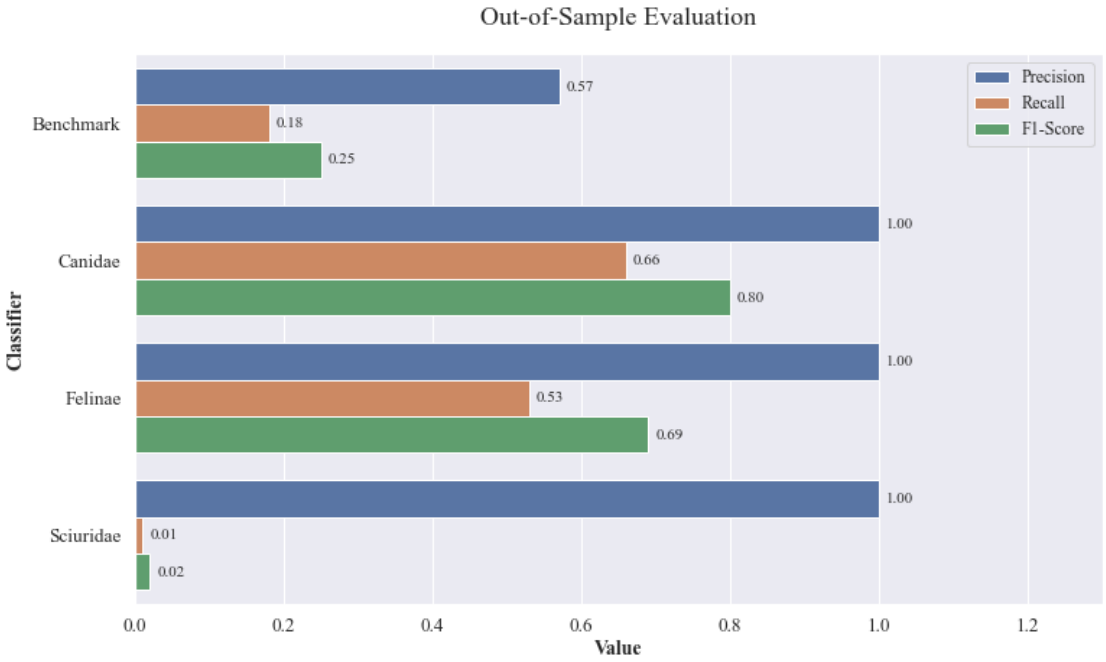


- *Out-of-Sample Model Testing*

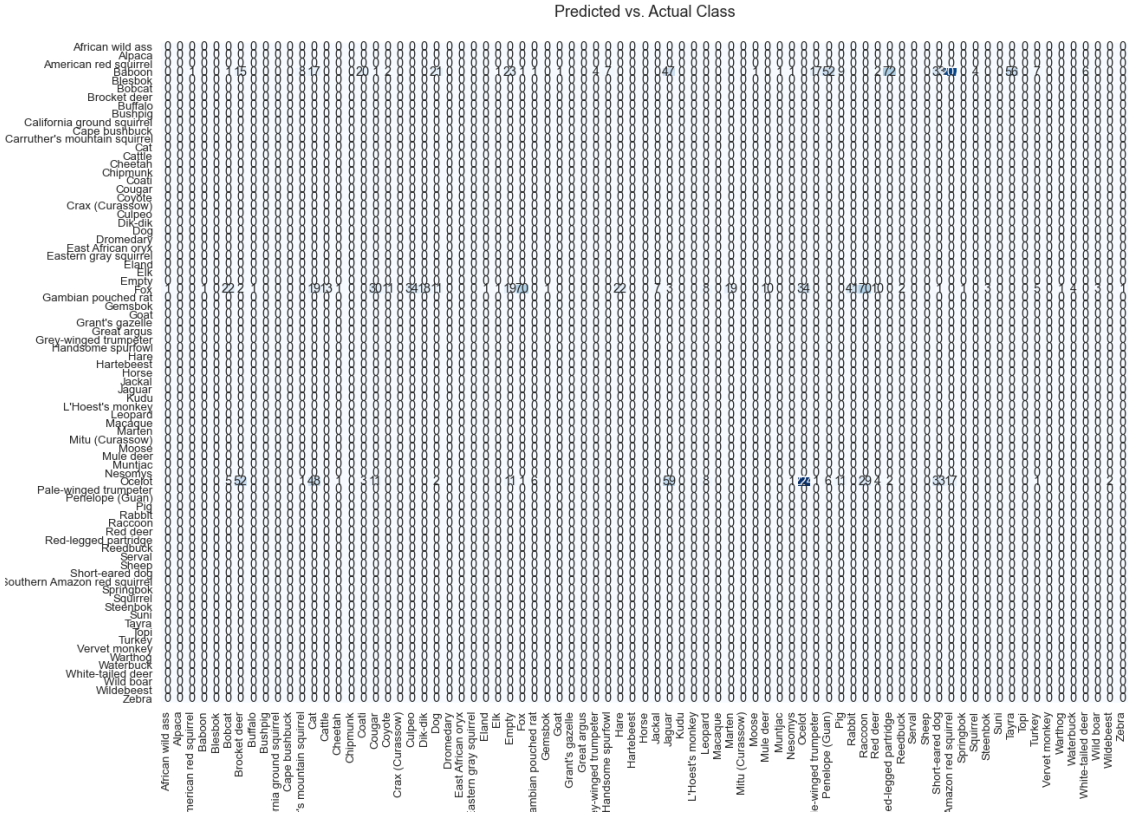
Classifier	Loss	Top-1-Accuracy	Top-5-Accuracy
Benchmark	7.914284	0.175104	0.427040
Canidae	1.547057	0.662675	1.000000
Felinae	2.277555	0.532468	1.000000
Sciuridae	14.581409	0.010955	0.884194



Classifier	Precision	Recall	F1-Score	No. Images
Benchmark	0.57	0.18	0.25	1679.0
Canidae	1.00	0.66	0.80	501.0
Felinae	1.00	0.53	0.69	539.0
Sciuridae	1.00	0.01	0.02	639.0

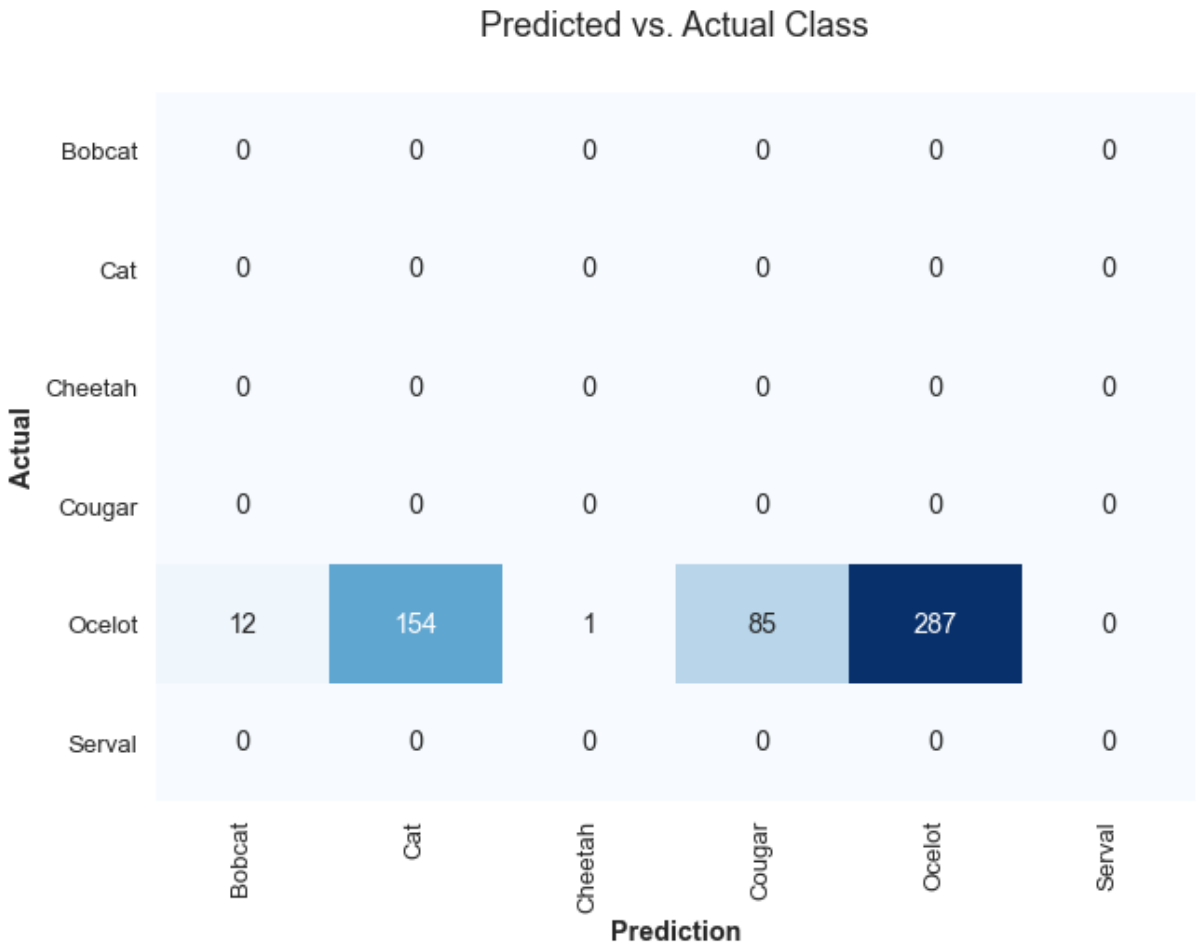

Out-of-Sample Evaluation
Precision
Recall
F1-Score
Benchmark
Canidae
Felinae
Sciuridae
0.57
0.18
0.25
1.00
0.66
0.80
1.00
0.53
0.69
1.00
0.01
0.02
Classifier
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Value

- *Benchmark Model Out-of-Sample Testing*

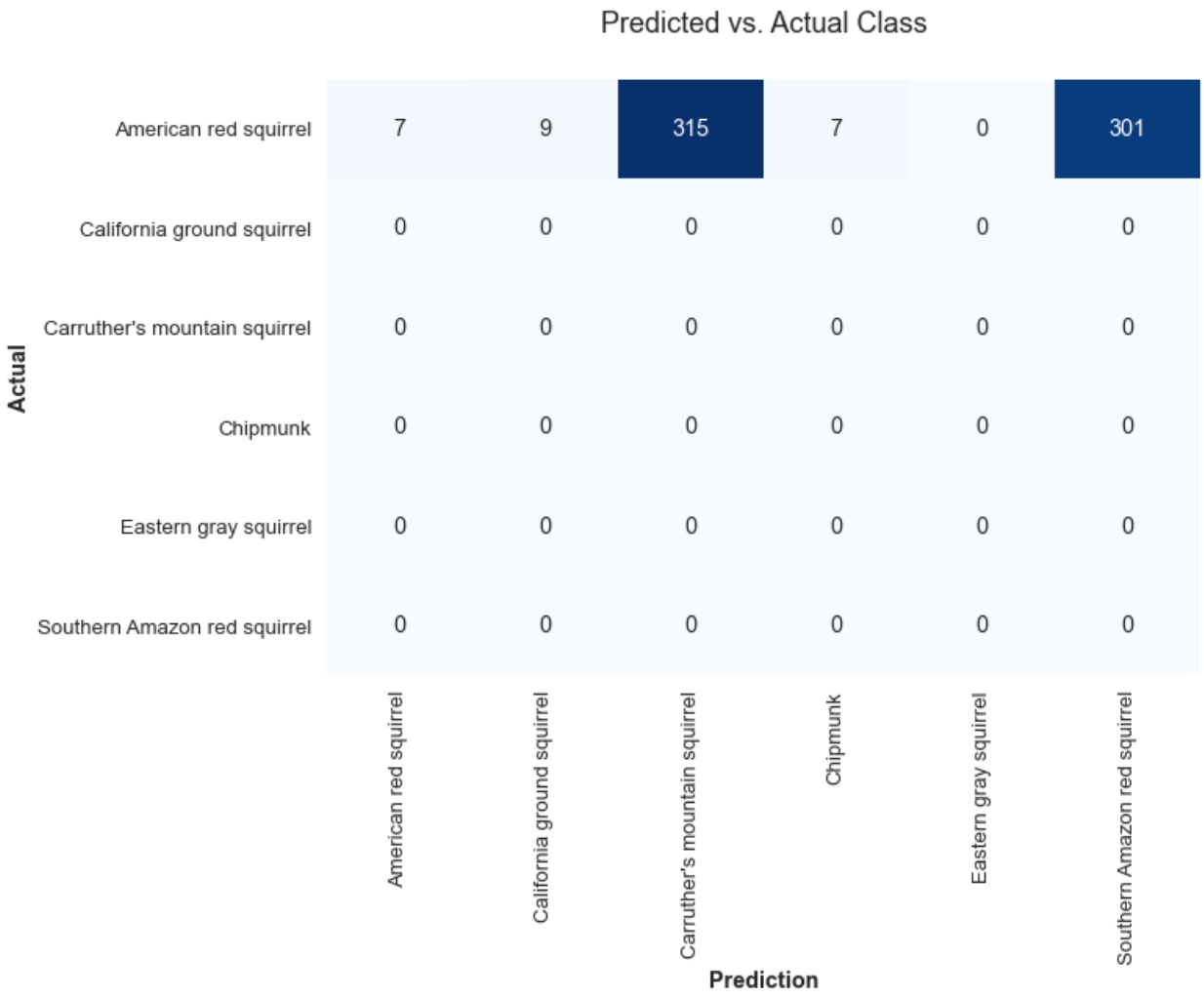


- *Canidae Model Out-of-Sample Testing*

- *Felinae Model Out-of-Sample Testing*



- *Sciuridae Model Out-of-Sample Testing*



iv. Instance Details