A Case Study Using Deep Learning to Identify North American Arthropods in Photographs

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Bioinformatics and Computational Biology

in the

College of Graduate Studies

University of Idaho

by

Alexander J. Mckeeken

Approved by:

Major Professor: Marek Borowiec, Ph.D.

Committee Members: Audrey Fu, Ph.D.; Paul Frandsen, Ph.D.; Terrence Soule, Ph.D.

Department Administrator: Paul Hohenlohe, Ph.D.

May 2022

Abstract

Identification of arthropods is important in academic and medical applications such as species-species interaction studies and identification for medical diagnosis. Deep learning is a tool that can be used to solve these problems quickly and accurately. For this study, a deep learning model was developed that has the capability of identifying North American arthropods to the genus level and compared multiple methods to increase the performance of this model. These methods include changing the neural network architecture, class balancing, and changing the image input size. The full deep learning model using InceptionResNetV2 obtained top 1 accuracies of 80% and top 5 accuracies of 92%. Comparatively, it was found that changing the neural network to EfficientNetB7 in a subset of the full model achieved a top 1 accuracy of 90%. It was also found class balancing in certain circumstances increased recall and that increasing image input size had a logarithmic effect on performance.

Acknowledgements

I would like to express my gratitude to my major professor, Marek Borowiec, who guided me throughout this project and was always available for any questions I had. I would also like to thank the Bioinformatics and Computational Biology students and faculty, friends and family who supported me and offered deep insight into the study.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Correct species identification is integral to downstream biological applications such as species-species interactions (Åkesson et al., 2021) and phylogenetic problems which support research in conservation and sustainability. However, current identification methods often are time consuming and are susceptible to bias or inaccuracies. This is because most current approaches require an expert capable of identifying the specimen. For many taxonomic groups there are few or no experts capable of identification down to species level. For example, a paper on identification in medical laboratory settings discusses how arthropod identification is dependent on the expertise of the laboratory and can be challenging because of the variety of specimens and the variance in expertise of labs(Mathison & Pritt, 2014). If further identification is needed the specimen can be sent to entomologists, a local university, or museum. This shows how time-consuming identification can be and how many places a specimen can go before an identification is made. This also shows that there is need for arthropod identification in areas other than research and potentially lead to faster diagnosis and treatments for arthropod related illness.

This problem may be solved through deep learning, a method which utilizes large amounts of data to train a network or machine learning model to solve complex problems like identification. Deep learning is a newly evolving frontier of computer science that leverages what we see in biological neural networks. Biological neural networks are the system neurons interlinked together. Neurons receiving a will reach a charge threshold which stimulates the neuron to "fire" a signal to other connected neurons. Similarly, an artificial neural network communicates by altering the signal from the input data which then leads to an output signal to linked neurons or nodes. By applying these biological processes to computer science, a machine can mimic learning and using an artificial neural network and enough relevant data can be used to train a model to solve complex problems. While deep learning has been looked at as a method for identification of arthropods most studies have only used deep learning for small number of taxa or singular families (Boer & Vos, 2018; Ding & Taylor, 2016; Hansen et al., 2020). A notable exception to this is Seek by iNaturalist (Ueda, 2019; Van Horn et al., 2018). While iNaturalist's model tries to identify much more than just arthropods this post does quantify the accuracy of insects at only 65% for species level which is the lowest accuracy of all the groups it trained. This shows there is a lot more work needed before there is accurate large-scale identification of arthropods. Therefore, **the objective of this study is to determine if deep learning is a realistic method for identifying arthropods to the taxonomic level of genus, and to determine the most optimal machine learning methods that may be used for this specific goal**. To determine the performance of the model a full model is first trained on all North American

arthropod images obtained. This full model is then subset into a smaller dataset where different optimization methods are performed and compared together. To measure performance, accuracy metrics for the full model are calculated and then for the subset models along with accuracy a confusion matrix, accuracy over time, loss over time, precision, recall, and f1-score are also obtained according to standard metrics to measure performance for multi-class classification deep learning models. While deep learning should not be the only way to identify arthropods, if we can verify that this method is robust and efficient and optimize it for this specific goal, we could significantly impact how arthropod identification is done in the future.

Chapter 2: Literature Review

Machine learning is a subset of artificial intelligence that creates algorithms using data to learn to solve problems. There are two main types of machine learning paradigms: supervised and unsupervised. Supervised machine learning systems utilize curating labeled data into a homogenous dataset to train on and allow for more control on how and the machine "learns" from the dataset. Labeled data means that the data is connected to a label which usually denotes what the correct prediction is supposed to be. Unsupervised machine learning is a paradigm that allows the algorithm to try to learn from a dataset without labeling the data. This results in a model that may work out its own method to tackle the problem but gives up more control on how the model learns. Most unsupervised methods are used for clustering and feature reduction (Mahesh, 2019). While many of these machine learning models use deep learning to achieve these goals some machine learning techniques opt-out of using deep learning. An example of this is random-forest which uses decision trees instead of utilizing neural networks and therefore doesn't fit into the category of deep learning (Mahesh, 2019). An advantage of this type of machine learning method is that non-deep learning methods tend to be less parameter rich and easier to implement. While deep learning usually is much more parameter-rich and tend to be more robust for more complicated problems and are more likely to be used for large-scale classification problems.

Deep learning utilizes deep networks of interconnected nodes (or neurons) to act as an artificial version of the human brain's neural network and have been invaluable in solving some of our hardest problems. These deep learning networks are called artificial neural networks. The structure of a neural network is usually comprised of layers. Layers are groups of neurons which take in information from a previous layer or the input data directly and send this information to another layer or act as the final prediction. The input layer is the layer that gets its information from the data directly and the output layer consists of the final predictions for the model. All other layers between these two are known as hidden layers. These layers connect to other layers through the individual nodes.

Each connection between a node has a weight and each node has a bias. A weight is a value used to control the strength of a connection. This value determines how much one node's signal will affect another one being signaled. A bias is a value that is associated with each node. This will be added after the weight calculation is found and affects how far the activation function is shifted. The activation function is the weighted sum between a node or nodes within a single layer of the neural network that are derived from the calculations between the weights and biases.

There are many different types of artificial neural networks used in different applications and specific problems. Some of the main neural network architectures are multilayer perceptron, convolutional, and recurrent neural networks. The multilayer perceptron (MLP) is one of the smallest neural networks usually consisting of an input layer, hidden layer, and output layer. These networks are usually used for simple datasets like tabular data, classification prediction, and regression prediction problems. Convolutional neural networks (CNNs) are much larger than MLP networks and consist of tens to thousands of interconnected hidden layers. Along with this increase in layers, convolutional neural networks use a small filter containing a grid pattern which runs across the data to interpret and learn feature and spatial information. The grid-like pattern and feature rich quality of image data makes CNNs well suited for solving image related problems like classification. Recurrent neural networks are similar to CNNs with their large networks of interconnected neurons. However, RNNs excel at data where each part of the sample is related to each other. This is because RNNs take the output from a particular layer and uses this as input to the same layer which can store information in the hidden layers as a pseudo-memory. RNNs are one of the most widely used networks because of their ability to solve problems related to language recognition, text summarization, and music composition because these types of data are highly contextual based on previous output. While all these networks are used for different types of data and in different contexts building the model from these neural networks usually follows the same three steps: training, validation, and testing. The training step allows the network to access and "learn" on the data. This usually consists of 60–80% of the whole dataset while the validation and testing sets are usually split into 10–20% each. These are split to reduce overfitting. Overfitting is a common problem in deep learning that can be thought of as a lack of ability for the model to generalize. It comes from the model learning specific features from the datasets used in training that are not generalizable, and thus creating artificially high accuracies or low losses during training but poorer performance with data outside of the data used during training. During training, each sample is converted to values based on the input data, and these values are fed into the artificial neural network. The product of the weight and value calculated in the previously connected node is found for each connection and then the bias is applied at each node. This is repeated until the last layer which is representative of the final prediction. While a single pass through the initial neural network usually does not result in accurate predictions, after each iteration the neural network is allowed to change. Iterating back through the layers the weight value between the nodes and the biases at each node of the neural network are computed with respect to the gradient of the loss function by using the chain rule. Backpropagation is this change in the weights and biases as a response to the input data and prediction (Hecht-Nielsen, 1992). The validation step runs at the end of each training epoch, a single pass through the training dataset, and compares the prediction from the

model to the real prediction to determine loss and accuracy. The training loss and accuracy is not used to determine how well our model is performing because it is not a good indication of how the model will do with novel data and if training was used to determine if the model improved after each training epoch, then there would be a higher change the model would overfit. The last step is testing which is an estimator for real-world performance of the model. The testing step is run after model development is done and the dataset used for it contains data that is never seen during training. It is run the same way as validation where it calculates loss and accuracy to minimize the likelihood of overfitting going undetected by analyzing a group of samples that should be novel to the model. After this is done the model is trained and validated and can be used to make predictions on similar data.

There are many ways to measure the performance of a deep learning model. The most common methods are accuracy, loss, precision, and recall and ROC curves. Accuracy measures the total accurate predictions over the total number of predictions. While this metric can indicate the general performance of a model at a glance, it is relatively simplistic compared to other metrics. For example, loss measures how poor a prediction was for a single sample. There are many ways to calculate this with the common types being mean absolute error, mean squared error and categorical crossentropy. All of these estimate the predictions error in reference to the true prediction but mean absolute error and mean squared error are mostly used for regression problems while categorical crossentropy is used for multi-class classification problems.

In image classification there are many neural networks to choose from but the neural network with the best accuracy for many image recognition tasks was InceptionResNetV2 at the outset of this project (Szegedy et al., 2016). While this seemed to be the best option there have been many neural networks that focus on performance and reducing computational costs since then. This led to a group of neural network that tried to be as efficient as possible which were called EfficientNet with the most accurate network being EfficientNetB7 (Tan & Le, 2020).

Chapter 3: Materials and Methods

Hardware Specification

A local Exxact Valence VWS-1542881-DPN-X299 workstation was used to store data and develop the model. The data is stored locally on an Arthropod Molecular Systematics Lab computer using a 4TB Western Digital HDD and all computation is run on an Intel Core i9-7900X CPU and 4 NVIDIA RTX 2080 Ti 11GB GPUs. The operating system used is Ubuntu 16.04.

The deep learning pipeline uses a TensorFlow 2.3-GPU on a Docker image with Jupyter support (for a full list of components see Appendix A).

Introduction

During this study we constructed two scripts that are used to obtain the images from two citizen science image web-based databases. The first is BugGuide.net, a website containing North American arthropod images uploaded and identified by citizen scientists (Anonymous, 2022). The second is iNaturalist.org which, like BugGuide, is a web-based database that crowdsources images and identifications but is not limited to arthropods, allowing users to upload photographs of any living organism (*INaturalist*, n.d.). The main difference between the two is that BugGuide collects a limited number of high-quality curated images for as many classes (species) as possible, while iNaturalist collects any identifiable images uploaded to the service. As a result, iNaturalist images often reflect organism abundance or sampling effort at the expense of image quality, while BugGuide contains a smaller number of high-quality images of many North American arthropod species.

These images were stored in custom SQLite databases which allowed for quick querying of images during model development. Model construction took place in a Docker environment, which is a virtual environment that mimics a computer's operating system allowing for version control of operating system environments (Merkel, n.d.). Genus, the first major taxonomic rank above species, was used as the goal of the class predictions. The reasoning for this is that very few images of most individual arthropod species are available which would mean insufficient data for training and identification of most arthropod species, many of which are similar. Secondly, reliable identification of genera would still constitute an important contribution as there are no experts capable of quickly and reliably identifying all of >10,000 North American arthropod genera (M. Borowiec pers. comm.).

Six separate models were created in this study (Table 1) and tested against each other based on determining three different questions, does the neural network architecture affect performance,

what effects does implementing class balancing techniques affect recall, precision, and accuracy, and does input image size affect model performance and/or training times.

Image Acquisition:

The first script is for obtaining the images from BugGuide.net where the script first opens the webpage that contains the taxonomy for Arthropoda. The script then opens the top link in each page that has not been opened before and continues until it gets the last entry. This last entry is correlated with the lowest part of the taxonomic tree and its taxonomic rank is given by the first word of the last entry above the text "Nothing below this species" and the second word gives the scientific name for that group of organisms (e.g., *Orthoporus ornatus*). The script then selects the Images tab and loops through all the images until every image is downloaded and given a unique identifier. The script then moves to the second to last taxonomic entry link and the process repeats. This allows for all images on the website to be retrieved. The original images are saved to a folder structure where each folder is named based off the scientific name and taxonomic rank and mirrors the structure of the website. After this is completed, the folders are looped through again and input into the BugGuide database detailed under the Database Design section of this paper.

The second script was used to obtain iNaturalist images through Global Biodiversity Information Facility (GBIF) which includes all the available image data from the iNaturalist website that falls into creative commons licenses for public use (Occdownload Gbif.Org, 2020). The format of this image data includes taxonomic information and the image's associated hyperlink. There are limitations to the number of images which may be extracted from the website every day so not all images from iNaturalist have been retrieved. Like the BugGuide webscraper, all images are stored in an associated SQLite database.

Database Design:

An individualized SQLite database to store the image data from iNaturalist and BugGuide were constructed.

The BugGuide database was constructed where each table in the database contained a taxonomic level up to phylum with the additional table containing the image data. This included all levels present on the BugGuide site which are subspecies, species, genus, subtribe, tribe, supertribe, subfamily, family, superfamily, suborder, order, superorder, subclass, class, subphylum, phylum, kingdom. The database then references these tables by starting with the image data table and then referencing the subspecies name and index in the subspecies table. The subspecies table then

references the species name that the subspecies is associated with taxonomically and its corresponding index position. This pattern continues throughout the database until it reaches the singular kingdom entry in the kingdom table. Although only tables up to phylum are needed, the kingdom table is available for future expansion of the model to account for different phyla. The exception to this structure is in the image data, subspecies, and species entries where everything above the entry is referenced until genus. This is because everything from genus and above have unique names while species and subspecies do not. This results in errors when searching for images based on subspecies or species and so this workaround was added.

During the image acquisition process for iNaturalist images each image had associated taxonomic positions but, BugGuide had a much more specific taxonomic ranking scheme. Because of this the iNaturalist database only referenced the genus, species, and subspecies information for each entry reducing querying times and the size of the database. The extracted data from the iNaturalist database then could use the BugGuide database as a reference to determine the complete and more specific taxonomic hierarchy of each image.

Docker as Virtual Environment:

Docker is an application that emulates working environments without affecting local computer program versions. This allows for reproducibility of applications by packaging them with everything that is needed for them to run without having issues with conflicting versions of programs among other common problems. We opted to use Docker because TensorFlow (the main application used for training and testing our models) has a Docker image with most of the applications we need to start training and has the required GPU applications for doing multiple GPU training. The Docker image we used is the tensorflow/tensorflow (https://hub.docker.com/r/tensorflow/tensorflow) which includes the latest GPU and Jupyter support. TensorFlow 2.3, Keras 2.4 and Python 3.5.2 were used for the model development. Along with these core programs we also utilized many Python modules. These modules are pandas (pandas development team, 2020; McKinney, 2010), NumPy (Harris, et al., 2020), matplotlib (Hunter, 2007), Pillow (Clark, 2015), and scikit-learn (Pedregosa, et al., 2011).

Training and Validation:

Image Sanitization:

The datasets used were not sanitized meaning that images were not manually removed if they were not optimal or accurate. This is because this process would be time

consuming, and BugGuide and iNaturalist have systems in place to maximize the quality of the images.

iNaturalist has a verification system which allows the image, or observation as described by iNaturalist, to be added to a category called research grade observation. This is achieved if 2/3 of identifiers agree on a taxon assignment, the taxon is species level or more specific (if the community doesn't vote that this observation does not need more IDs), the community does not vote for the observation to need more IDs, and it will not be counted if the smallest place that contains this observation does not include at least 10 other observations or >80% of the observations in that area are marked as not wild/naturalized.

BugGuide has a more simplistic method of determining accuracy of taxon using a category called ID requests where users can vote for an image to be added to a certain taxon. If image is subjectively judged to be low quality or cannot be identified, it is moved to category called frass and then deleted after 30 days.

Model Development:

The script used to train the data was built with many modern machine learning techniques in mind. To keep everything consistent all random variables in building the model have been kept consistent with a seed value of 12345. This includes the use of Python's built-in random module and os module and when calling Keras' ImageDataGenerators method. Images were then read from their SQLite Databases using the sqlite3 Python module and added to a pandas dataframe for easy data manipulation where each image was added with its associated taxonomic information and a unique image identifier name. This pandas dataframe was split into three pandas dataframes which each contain the images and their corresponding taxon information according to the training dataset, validation dataset, and testing dataset as per machine learning standards.

Full Model:

The first model constructed is what is referred to here as the full model. This consists of all images and genera collected and is split where the training dataset contains 80% of the images and the validation and testing datasets both contain 10% of the remaining 20% of the images. A table showing the number of images in each set is shown below (Table 1).

Table 1. Number of Classes and Images in Full Dataset

Datasets	Number of classes	Number of Images	Percentage of Total
Training	7,524	1,486,631	80%
Validation	7,524	161,544	10%
Testing	7,524	179,426	10%
Total	7,524	1,648,175	100%

The top 1, top 3, and top 5 accuracies were obtained for the full dataset. The top 1 accuracy represents the model's prediction correctly guessing the true label, top 3 means that the model's top 3 predictions contained the true label, and top 5 means the top 5 predictions contained the true label.

To allow for testing of the full model a website was made available to upload an image and obtain the top 10 predictions for our model. This was done through the University of Idaho's Northwest Knowledge Network (*IIDS NKN*, n.d.) and this website called Bug ID (*Bug ID - Using Machine Learning to Identify Arthropods*, n.d.).

Experimental Models:

For the training portion of this study, the full model was reduced to only include images from the taxonomic family Noctuidae (Lepidoptera; moths and butterflies) to minimize computation time for each model to train. Noctuidae contains many genera with a large variation in color, size, and morphology. The models constructed are shown below (Table 2).

Table 2. Experimental Models Constructed

Neural Network Architecture	Input Image Size	Separate Balanced Model?
InceptionResNetV2	512x512 (control)	Yes
	256x256	No
	768x768	No
EfficientNetB7	512x512	Yes

For performance evaluation confusion matrices, accuracy and loss graphs per epoch, final accuracy along with macro average precision, recall, and f1-scores were obtained for each model.

This was split into three datasets were split where the validation and testing set each held ~20% of the data and the training dataset held ~60% of the data. The final number of images in each dataset is given below (Table 3).

Table 3. Number of Classes and Images in the Experimental Models Datasets

Datasets	Number of Classes	Number of Images	Percentage of Total
Training	323	60,256	60%
Validation	323	19,894	20%
Testing	323	19,892	20%
Total	323	100,042	100%

Dataset Augmentation:

Each of the training, validation, and testing datasets are then shuffled to decrease the likelihood of the model learning relationships between positional information. All images that are found in the three datasets are then added to a folder for use with Keras' Image data generator method. This uses a pandas dataframe and a folder reference to create a Python generator that can be used to retrieve and train all the images to create a machine learning model. The Keras generator references the pandas dataframes created previously and uses categorical crossentropy as the label method, RGB color images, and shuffle all the images before storing them as a Python generator. The training generator also includes data augmentation which changes the images being added to the training generator by randomly augmenting an image in several different ways. The list of ways in which the images were augmented, description of what it does, and the value we used are listed below (Table 4).

Table 4. Image Augmentation with Description and Values Used in Training Generator

Augmentation Type	Description	Value
Rotation Range (degrees)	Image is rotated within range set	90
Width Shift Range (%)	Image is shifted along width axis	.05
Height Shift Range (%)	Image is shifted along height axis	.05

Shear Range (%)	Slants the shape of the image	.05
Zoom Range (proportion)	Zooms and magnifies the image	.2
Channel Shift Range	Shifts the color value within range	20
Horizontal Flip	Flips the image across vertical axis	True
Fill Mode	Fills in unknown pixels with known pixels in reverse order	Reflect

The augmentations listed above are all possible augmentations that can be applied to each image. It is important to note that this does not add all the possible augmentations as separate images and therefore the epoch size and time per epoch will not change but each image can have an augmentation when trained upon. The reason we use augmentations on images that are being trained on is to try to increase the chance of generalizing the image data and features to predict classes or in other words to reduce the likelihood of overfitting to the dataset. Variables constant across training variations are in Table 5.

Table 5. Variables for Experimental Models

Variables	Value
Batch Size	16
Maximum Number of Epochs	Normal Training: 60, Fine Tuning: 120
Learning Rate	e^{-4}
MirroredStrategy	Reduction To One Device
Number of workers	20
Optimizer	Adam

The batch size is how many images are running at one time during training. Through testing I found that for our specific setup sixteen images resulted in the most optimized training where number of images running at a time is maximized and epoch completion time is minimized. I chose 60 and 120 as the maximum number of epochs that can be run for a single model for the normal training step and the fine-tuning step. These are not reached in regular model development runs, however, because of an early stopping rule implemented in our code: if validation accuracy does not increase after five epochs, the model will either move on to the fine-tuning step or, if already fine-tuning, stop training and save model. This is to both decrease training time and prevent overfitting.

Although how neural network architecture affects model performance is one of the variables that we test there are some features that are the same between all models that we ran. One of these identical features is that all the neural networks use the same set of ending layers or top layers. These top layers are added to the end of base neural network. They are composed of a Global Average Pooling 2D layer, Dropout layer, Dense Layer size 1024 with L2 regularization of .0001 and a rectified linear activation, another Dropout layer, and another Dense layer the same as before except is the size of the number of classes we are trying to classify and a softmax activation. We add these layers to further protect from overfitting and minimize the output to the number of classes that we have. Transfer learning is also used to reduce training times and increase accuracy. Transfer learning provides a framework to utilize features that pre-built models have learned to solve new but similar problems more quickly and effectively (Lu et al., 2015). Transfer learning copies pre-built models trained on a specific task and utilizes these models for model development in similar tasks. ImageNet is a frequently used dataset that many neural networks use for transfer learning in image classification because of its large variability in classes and large dataset size (Deng et al., 2009; Huh et al., 2016). For this study the initial weights and biases for InceptionResNetV2 and EfficientNet came from pre-training on ImageNet. For the first part of training the model can only alter the top layers and therefore none of the original ImageNet weights and biases are changed. This allows to use neural networks that can efficiently recognize images and general features of images which can be utilized to classify the images in this study and prevents training changing these layers which could alter what features were found to have predictive capabilities. After each epoch the model is saved to a folder for reference and TensorBoard is utilized as well to create visualizations of how the model performs over time. These visualizations are graphs of how accuracy and loss change over epochs and include both the base training step and the fine-tuning training step.

InceptionResNetV2:

The control neural network used for this study is InceptionResNetV2 (Szegedy et al., 2016). InceptionResNetV2 is composed of both the Inception neural network and the Residual neural network. Residual networks build in “jumps” in its architecture that allows nodes to connect to other nodes in a non-linear way by skipping one or multiple layers. This is analogous to the biological construct known as pyramidal cells that have this same property. The inception network uses a method of creating a wider network instead of a deeper network where there are different sized filters that each share a single node allowing

for both broad and narrow pattern recognition. Both methods in a single network allows for some of the best prediction accuracy for image datasets such as ImageNet and is why this was implemented in our original design of our model.

EfficientNetB7:

For testing how to optimize identification we looked at a modern neural network called EfficientNetB7(Tan & Le, 2020). EfficientNet was made by using deep learning to construct its network using state of the art infrastructure techniques along with maximizing accuracy and minimizing the size of the network. This led to the development of 7 different models named EfficientNetB1, EfficientNetB2…EfficientNetB7. With each subsequent model the accuracy of the model increases while also attempting to minimize the size of the model to be as efficient as possible. Each of the models were constructed with a different image size input and so it is recommended to maximize performance from the model to train on images that are close to and the same or smaller size than the input size that the network was trained on. EfficientNetB7 was trained on images with the size 600x600.

All the same features from the base model stay the same besides replacing InceptionResNetV2 with the EfficientNetB7 neural network. Also, instead of normalizing the pixel values during the Keras generator step we pass in the images without any normalization. This is because EfficientNet contains a normalization layer that does the same thing as the Keras normalization function. These are both run on our dataset and the data is saved to TensorBoard for comparison.

Class Balancing:

In this paper we also test whether using simplistic oversampling method can increase the recall on our dataset and compare the cost of that increase in recall compared to the precision. To do this we change the size of the adjustments to the weights of the neural network to normalize it to the number of images in each of the classes using a Keras class weight method. This modifies the loss function in a such a way that the smaller classes have more weight than the larger classes. While undersampling techniques remove data from the dataset which leads to loss of information and oversampling involves duplicating images which makes specific features contained in the duplicated images to be learned over the majority classes features and these features could only be found in the minority class. The oversampling method that is used in this study is an oversampling approach that may lead to minority class features more pronounced in the final model each image is still only seen once

and so identical features will not be present compared to duplication-based oversampling. To do this in our model training we use the following equation (Equation 1).

$$\frac{n_{samples}}{(n_{classes} * np.bincount(y))} \quad (1)$$

This equation divides the number of total images ($n_{samples}$) by the number of classes ($n_{classes}$) multiplied by the number of samples per that class ($np.bincount(y)$). This results in a value that when multiplied by the number of samples in that class is equal to the number of samples divided by the number of classes.

Image Sizing:

The last variable tested was image size. This involved evaluation of whether the relatively large 512x512 image size that we chose to normalize all our images to is efficient and results in better performance of the model. To do this we tested two other variations, 256x256 and 768x768. The reason these dimensions were chosen specifically is because 256x256 is a standard image size used by many machine learning models and 768x768 has square dimensions and is divisible by our smallest increment of 256. While 1024x1024 would follow the 2^n rule that is usually applied to image sizes our current setup's VRAM capacity is unable to work with images of that size. While the reasons why dimensions were chosen are based on square dimensions divisible by 256 BugGuide and iNaturalist have different limits on what size images they will take into their databases. Differences in the average image size could result in an impact in accuracy due to Keras' bilinear interpolation of images adding information to the images that aren't representative of the true features of the arthropod in the image if the images are resized from a lower resolution to a higher resolution.

Testing:

For the testing portion of this study, we use Keras' evaluate method that works similarly to the model fit method except we only run only one epoch and use our testing dataset to evaluate how well our model worked. We run this for each model and just like the other runs output the results to TensorBoard. These TensorBoard statistics are then sent to matplotlib to create graphs to compare similar runs together.

Along with the graphs, mathematical metrics are calculated using the classification report method of scikit-learn. These include macro average precision, macro average recall, macro average f1-score, and accuracy which are standard metrics to measure the performance of a model (Grandini

et al., 2020). Macro average is used because it more accurately demonstrates that the effect of the biggest classes have the same importance as small ones have while other averages can disproportionately favor the biggest classes in the measure of the metrics. Precision is defined as the number of positive class predictions that belong to the positive class. Precision is usually maximized when the purpose of the model is to identify the highest number of instances of a class correctly. Recall is defined as the number of positive class predictions out of all positive examples in the dataset. Recall is usually maximized to minimize the likelihood of false negatives and maximize the accuracy of rarer classes. F1-score is a metric that evaluates the performance of a model by combining both precision and recall into a measurable score. Accuracy is a measure of correct predictions over all predictions.

CHAPTER 4: Results

Full Model:

The full model was used to determine if using deep learning for the creation of a large-scale model was an effective way to identify North American arthropods. The full model consists of all ~1.4 million images. The top 1, top 3, and top 5 accuracy metrics were taken. The results of this are shown below (Table 6).

Table 6. Full Model Top 1, Top 3, and Top 5 Accuracy

Model	Top 1 Accuracy	Top 3 Accuracy	Top 5 Accuracy
Full Model	.80	.89	.92

For the full model 80% of the time the model correctly identified the arthropod from the image. On top of this 89% of the time the true label was in the top 3 predictions and 92% of the time the model was able to predict the true label in the top 5 results.

Experimental Models:

Control Model using InceptionResNetV2:

All experimental models were constructed using a subset of the full model which only included the taxonomic family Noctuidae (Lepidoptera; moths and butterflies). This control model uses the trained InceptionResNetV2, no weight balancing, and 512x512 images. The testing dataset was used to determine the statistics relevant to each part of the

results. Analysis was done using Keras' evaluate function. The confusion matrix below shows the results of this evaluation (Figure 1).

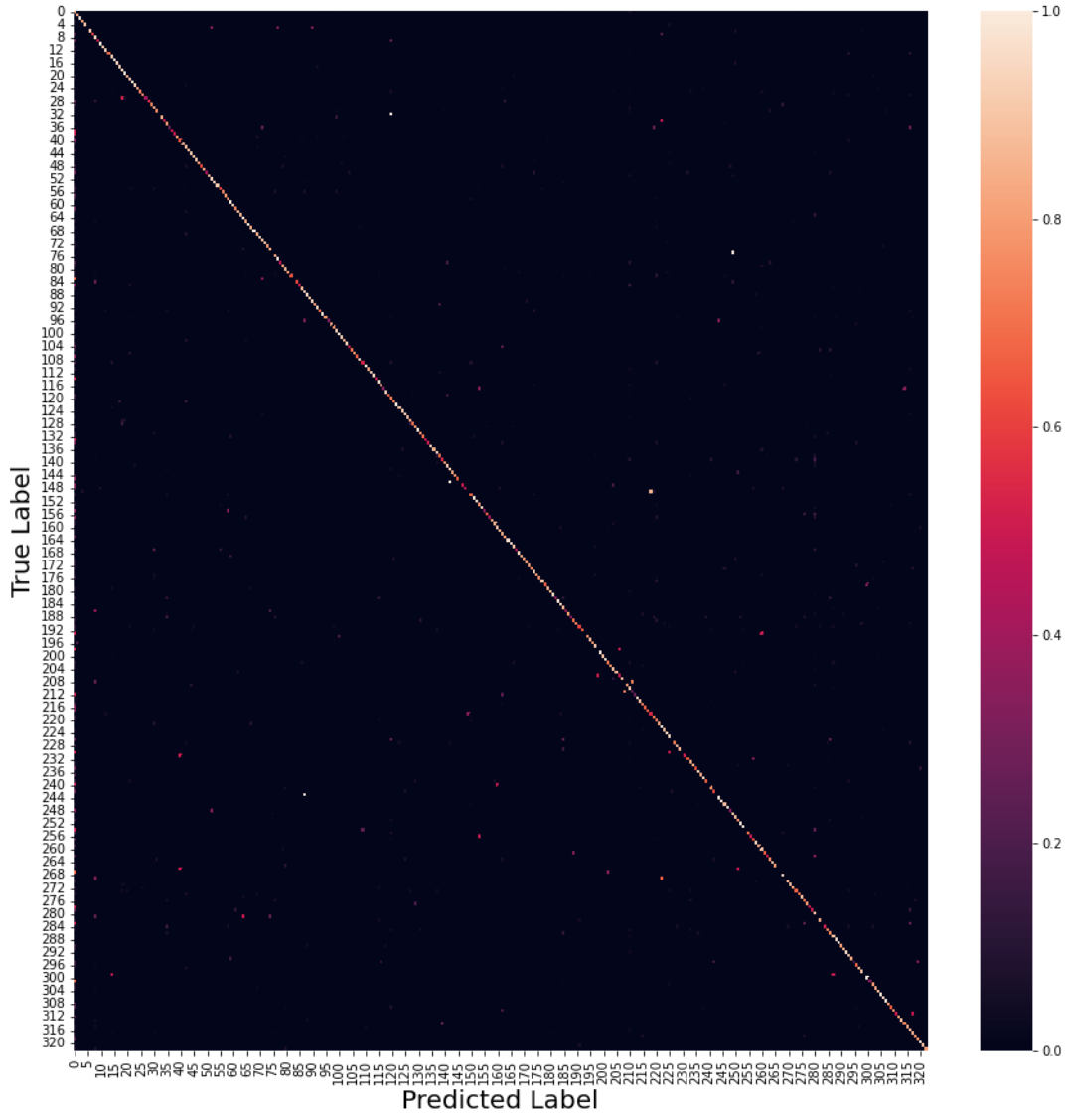

Figure 1. InceptionResNetV2 Confusion Matrix

The confusion matrix shows the true classes numbered on the y-axis and the predicted classes on the x-axis. The color legend shows the percentage a scale from 0 to 1 where 1 (white) means that all images are predicted to be that class and 0 (black) means that no images are predicted to be that class. Correctly predicted images are on the diagonal. Here we can see that most images and classes are accurately predicted but there are a few classes that are incorrectly identified and out of these some of them are consistently confused with one other class (light cells not on the diagonal).

The TensorBoard graphs below show the training and validation accuracies and loss over time with the x-axis representing the number of epochs and the y-axis representing the accuracy (Figure 2) (Figure 3).

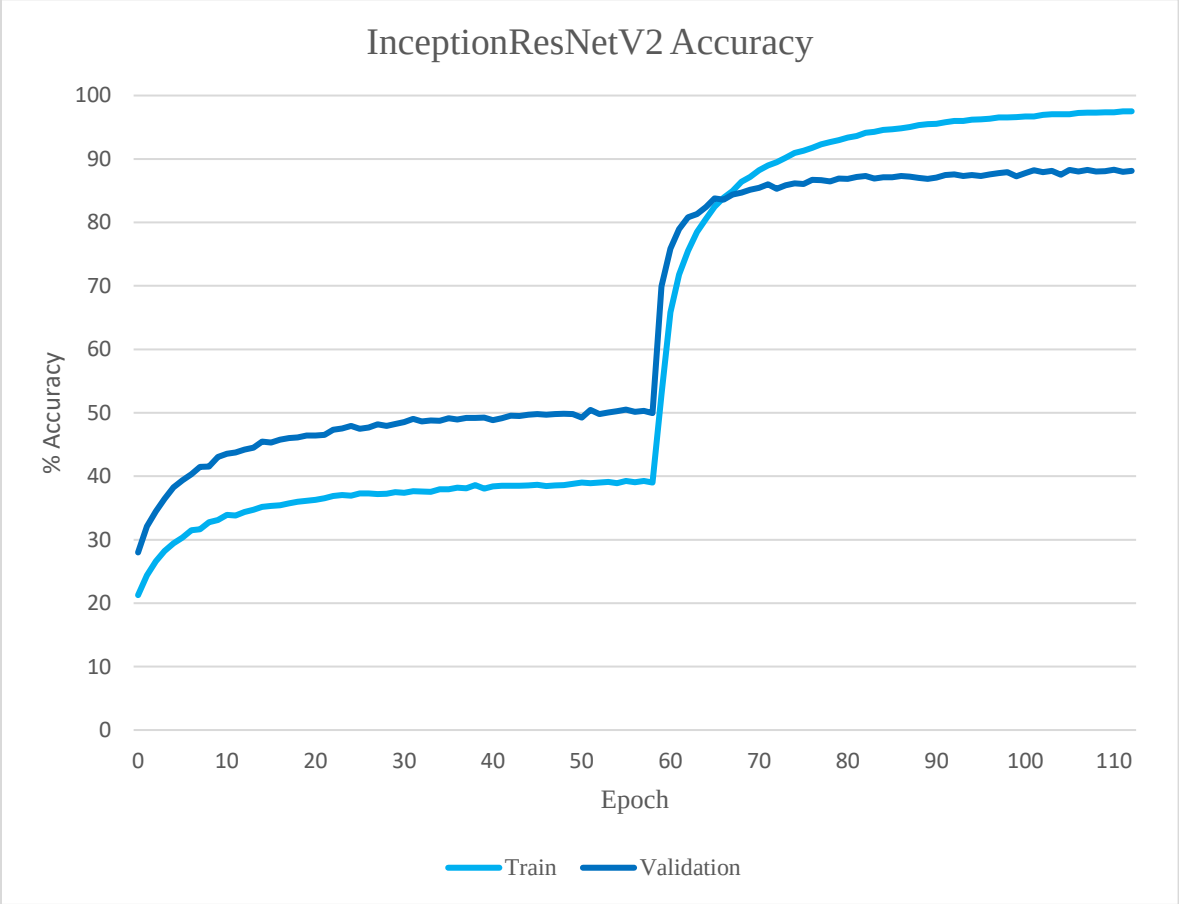

Figure 2. InceptionResNetV2 Accuracy Over Epochs

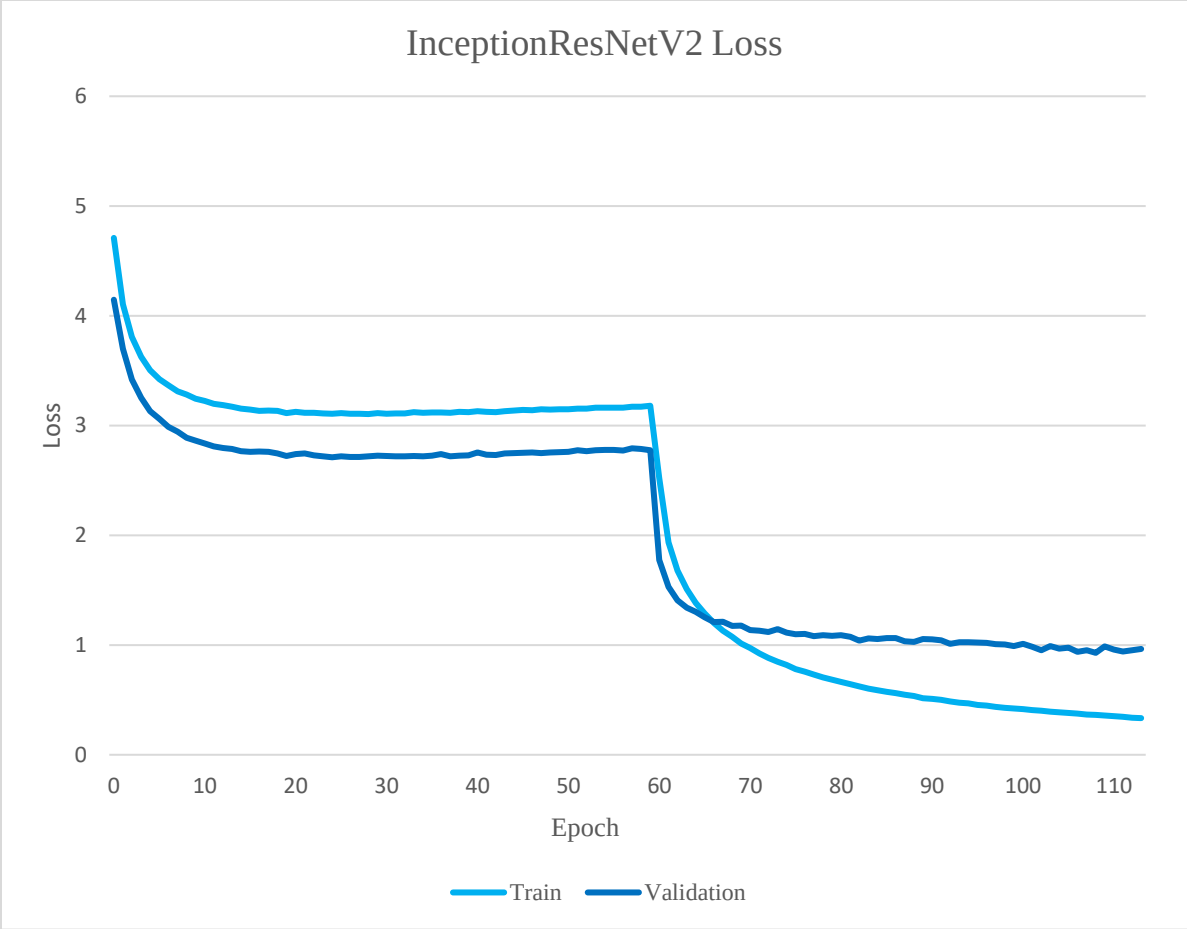

Figure 3. InceptionResNetV2 Loss Over Epochs

Both graphs show that over time the performance of the model improves. In both a large increase in performance is visible at the point when the model switched from initial training to fine-tuning. There is little divergence towards end of the training, indicating little overfitting.

The macro average precision, recall and f1-score were found along with accuracy. The results of this are shown below (Table 7).

Table 7. InceptionResNetV2 Precision, Recall, F1-score and Accuracy

Precision	Recall	F1-score	Accuracy
.83	.75	.77	.87

Macro average was used for these values because it gives the mean score for each class and thus accounts for differences in number of images in each class which. As seen in

the table, accuracy is high but the f1-score is much lower indicating that there is a lower average precision and recall that isn't included in the accuracy.

EfficientNetB7:

This section of the study tests whether the type of neural network architecture used has an impact on performance. EfficientNetB7, no weight balancing, and 512x512 model was used to test the performance change from changing the neural network architecture. The EfficientNetB7 confusion matrix is shown below (Figure 4).

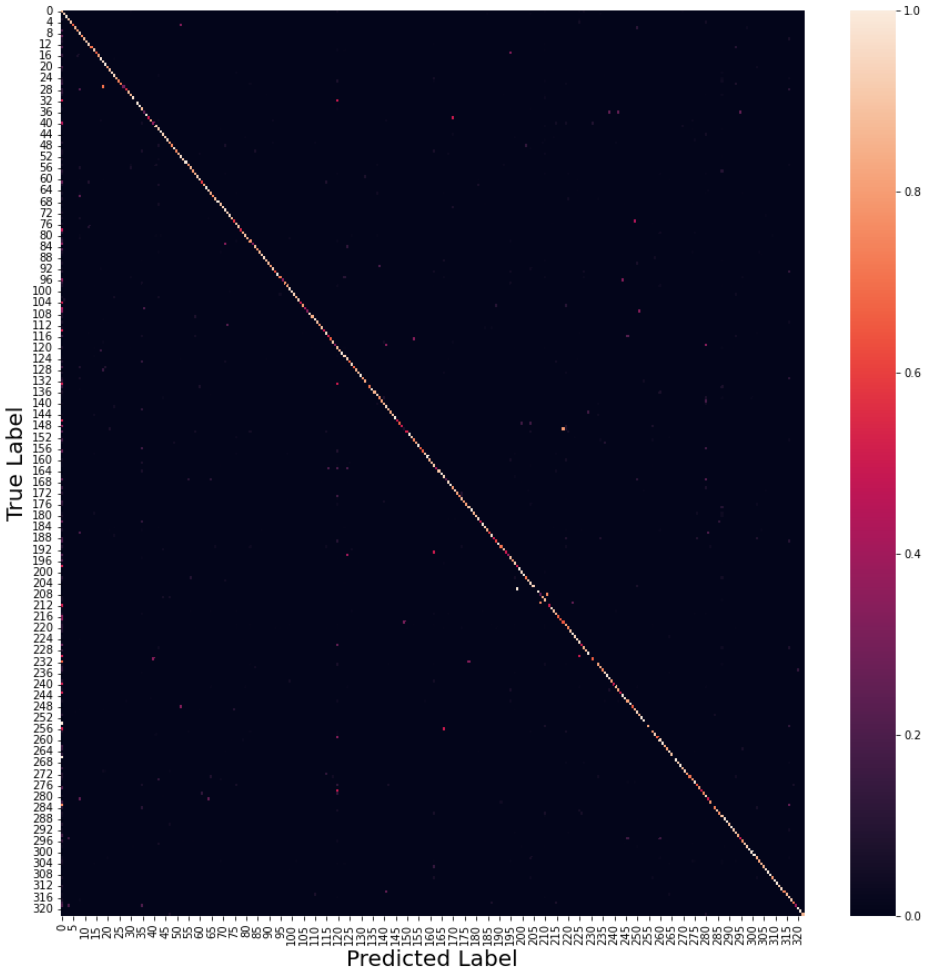

Figure 4. EfficientNetB7 Confusion Matrix

Similarly, to the InceptionResNetV2 model, most classes are accurately predicted but there are some classes that are consistently confused with one other class. With this said, there is more confidence in the predictions along the diagonal line and some of the images that were misidentified in the InceptionResNetV2 model are not misidentified here.

The TensorBoard graphs below show the training and validation accuracies and loss over time with the x-axis representing the number of epochs and the y-axis representing the accuracy for both InceptionResNetV2 and EfficientNetB7(Figure 5) (Figure 6).

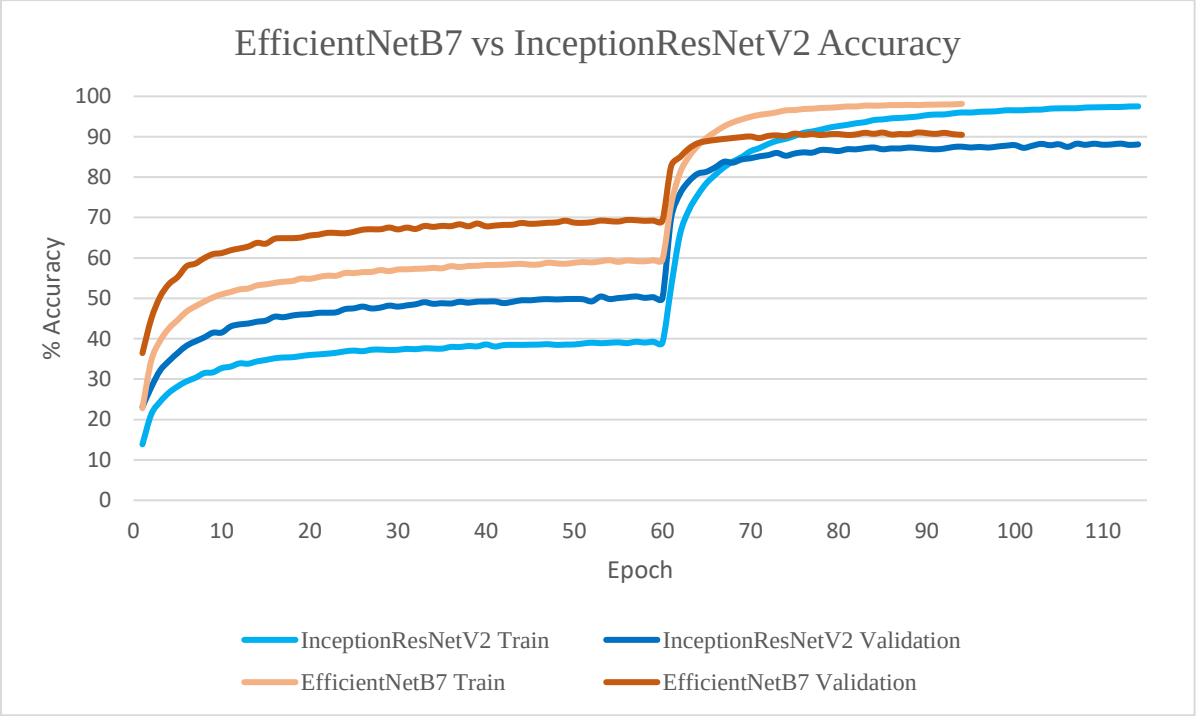

Figure 5. EfficientNetB7 vs InceptionResNetV2 Accuracy Over Epochs

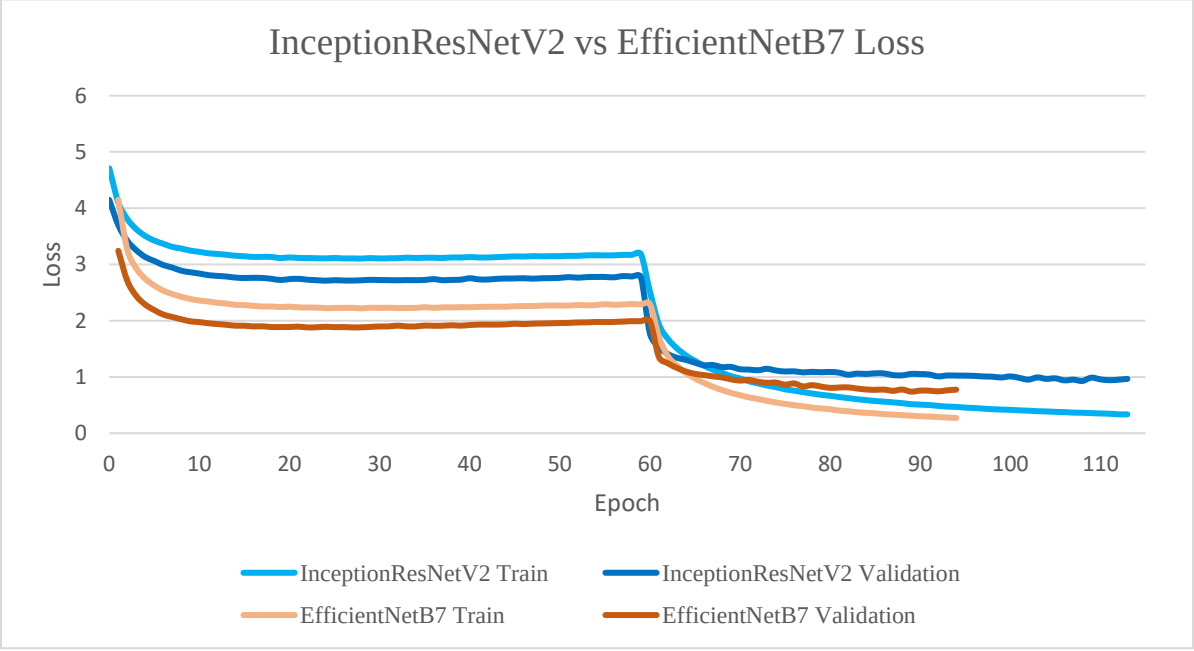

Figure 6. EfficientNetB7 vs InceptionResNetV2 Loss Over Epochs

Both graphs show that over time the performance of the model improves. EfficientNetB7 finishes training in a shorter number of epochs and at the end of training diverges less than InceptionResNetV2.

The macro average precision, recall and f1-score were found along with accuracy for both the models. The results of this are shown below (Table 8).

Table 8. InceptionResNetV2 vs EfficientNetB7

Model	Precision	Recall	F1-score	Accuracy
InceptionResNetV2	.83	.75	.77	.87
EfficientNetB7	**.89**	**.80**	**.84**	**.90**

The results show that EfficientNetB7 achieves higher precision, recall, f1-score and accuracy compared to InceptionResNetV2.

Class Balancing:

This section of the study tests whether class balancing can have an impact on performance. The models used are the trained InceptionResNetV2, with and without weight balancing, and 512x512 model and the EfficientNetB7, with and without weight balancing, and 512x512 model. The Balanced InceptionResNetV2 confusion matrix is shown below (Figure 7).

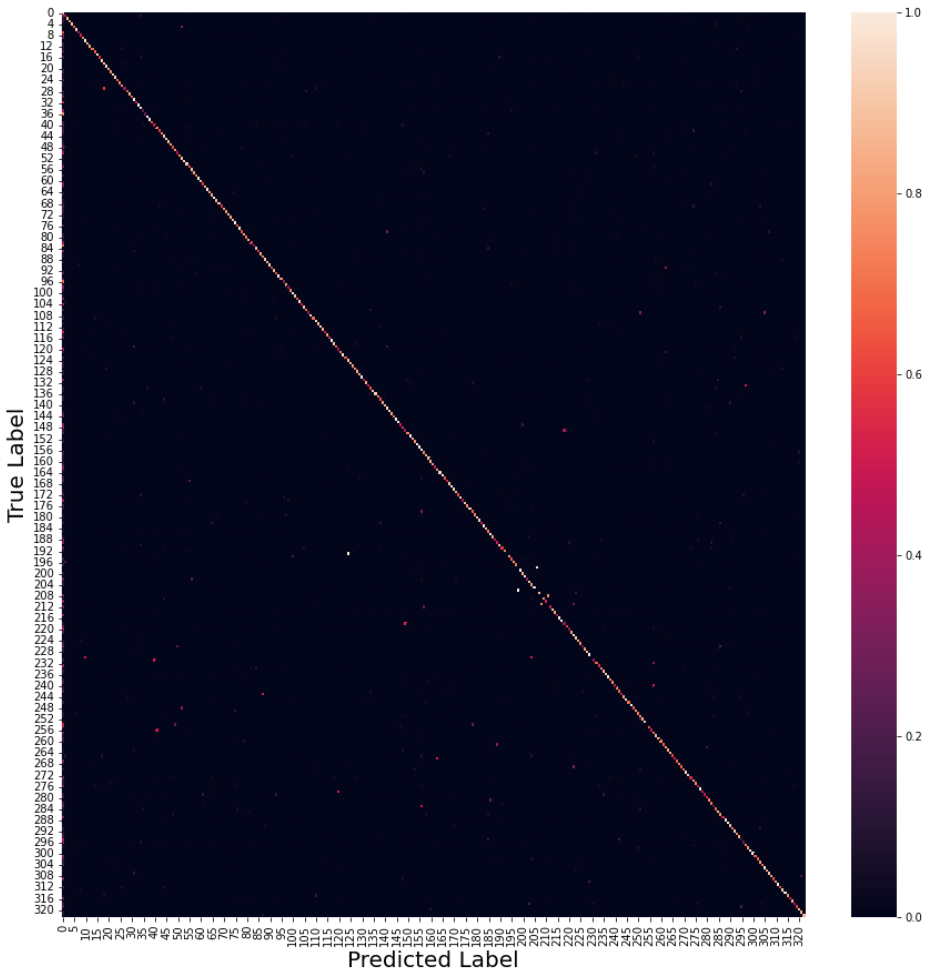

Figure 7. Balanced InceptionResNetV2 Confusion Matrix

This confidence in the predictions of this confusion matrix seems to be lower than that of the unbalanced InceptionResNetV2 model with fewer instances of classes being confused with another other class.

The TensorBoard graphs below show the training and validation accuracies and loss over time with the x-axis representing the number of epochs and the y-axis representing the

accuracy for both the Balanced InceptionResNetV2 and Unbalanced InceptionResNetV2 (Figure 8) (Figure 9).

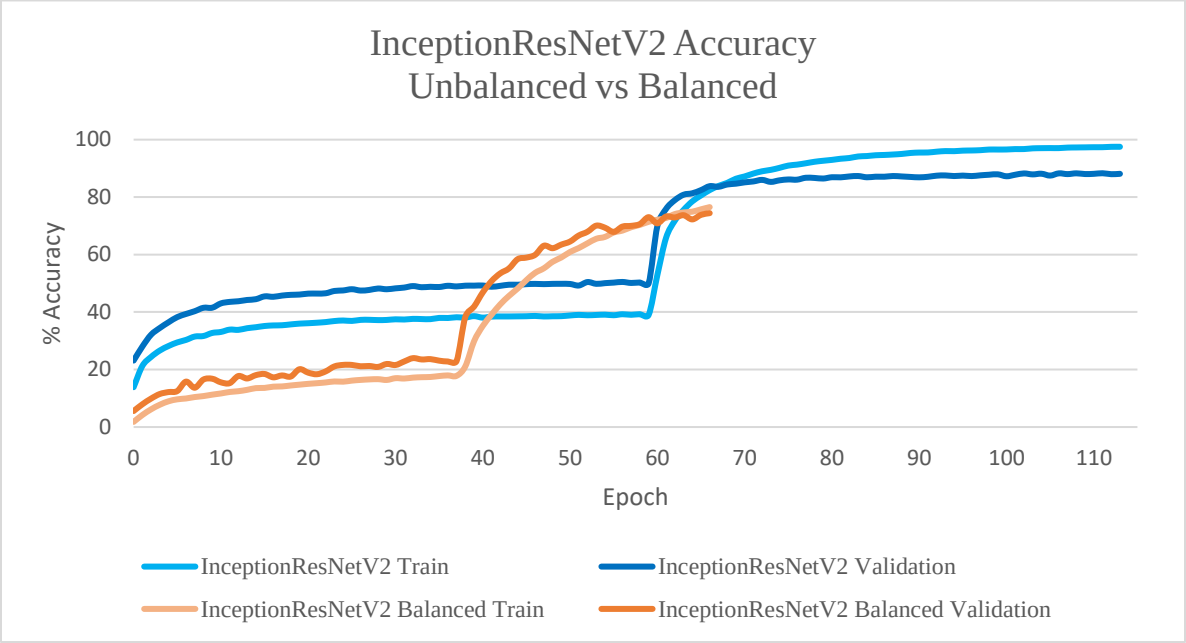

Figure 8. InceptionResNetV2 Balanced vs Unbalanced Accuracy Over Epochs

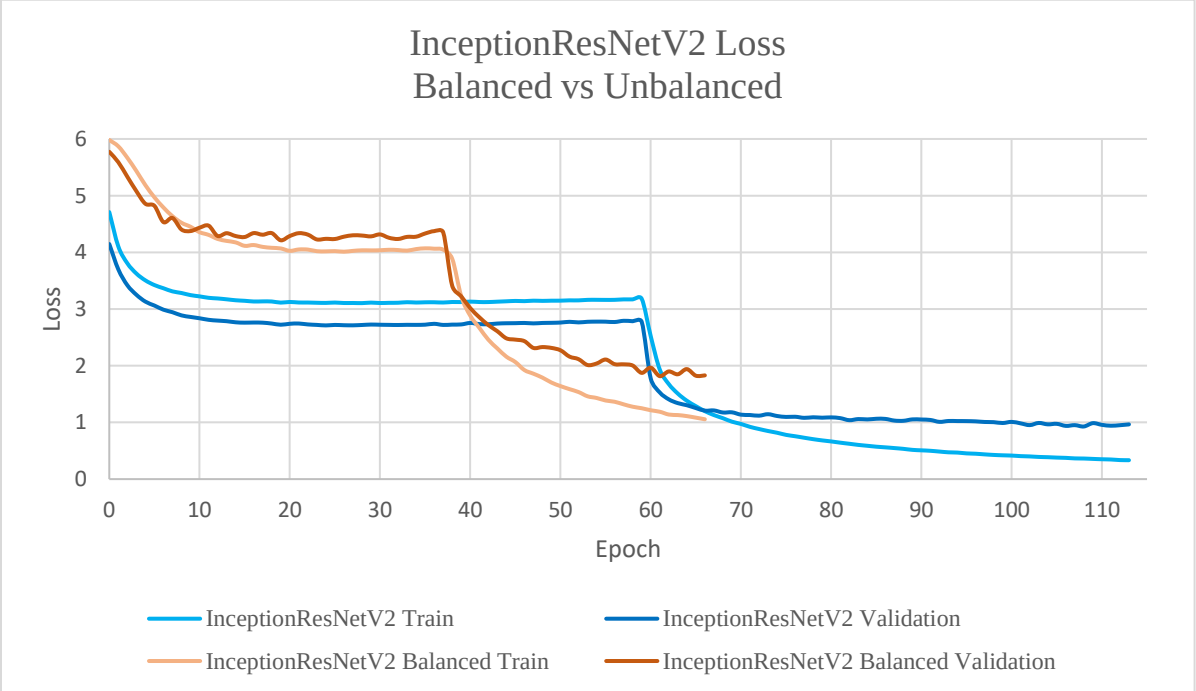

Figure 9. InceptionResNetV2 Balanced vs Unbalanced Loss Over Epochs

Both graphs show that over time the performance of the model improves. With this said the unbalanced model ends training much quicker than the unbalanced model at the cost of lower accuracy and higher loss.

The macro average precision, recall and f1-score were found along with accuracy for both the models. The results of this are shown below (Table 9).

Table 9. InceptionResNetV2 vs Balanced InceptionResNetV2

Model	Precision	Recall	F1-score	Accuracy
Unbalanced InceptionResNetV2	**.83**	.75	**.77**	**.87**
Balanced InceptionResNetV2	.72	.75	.70	.69

The unbalanced model performs better in precision, f1-score and accuracy compared to the balanced model. Also, the unbalanced model achieves the same recall as the balanced model.

The Balanced EfficientNetB7 confusion matrix is shown below (Figure 10).

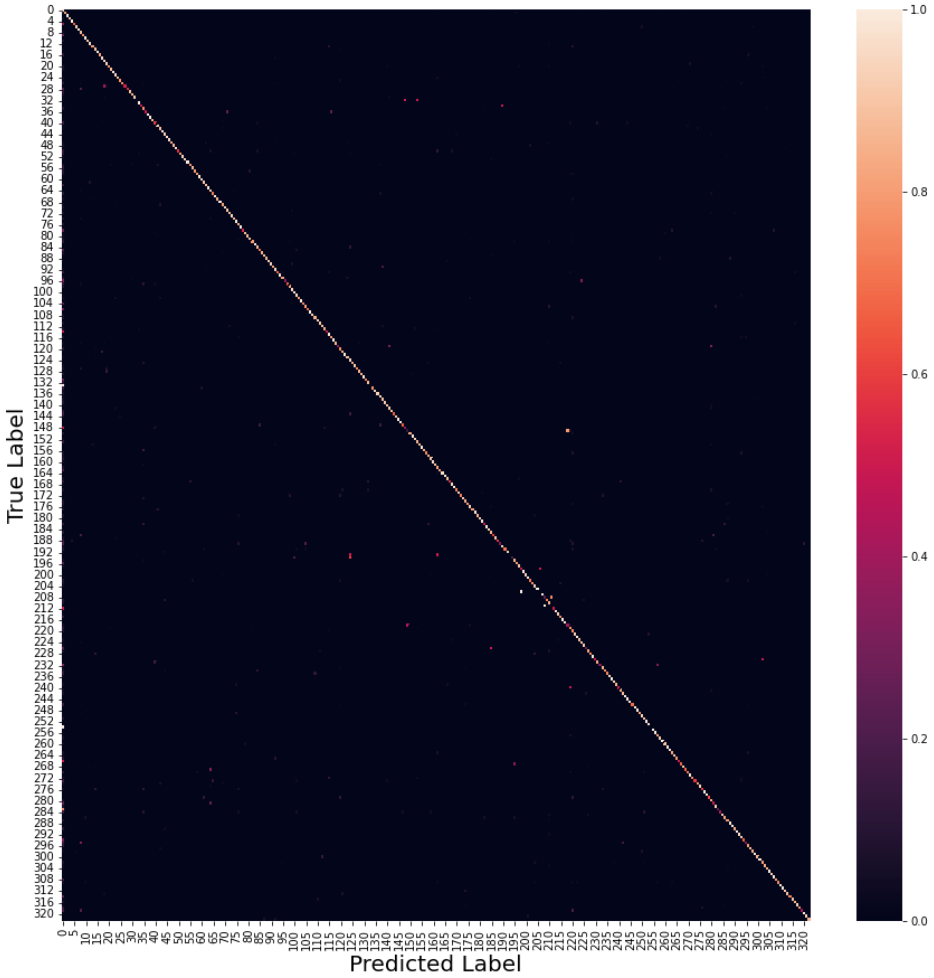

Figure 10. Balanced EfficientNetB7 Confusion Matrix

Similarly, to the unbalanced EfficientNetB7 model most classes are accurately predicted but there are some classes that are incorrectly identified and out of this misidentified prediction some of them are consistently misidentified with one other class, but the amount of these misidentified predictions seems to be much lower.

The Tensorboard graphs below show the training and validation accuracies and loss over time with the x-axis representing the number of epochs and the y-axis representing the accuracy for both the Balanced EfficientNetB7 and Unbalanced EfficientNetB7(Figure 11) (Figure 12).

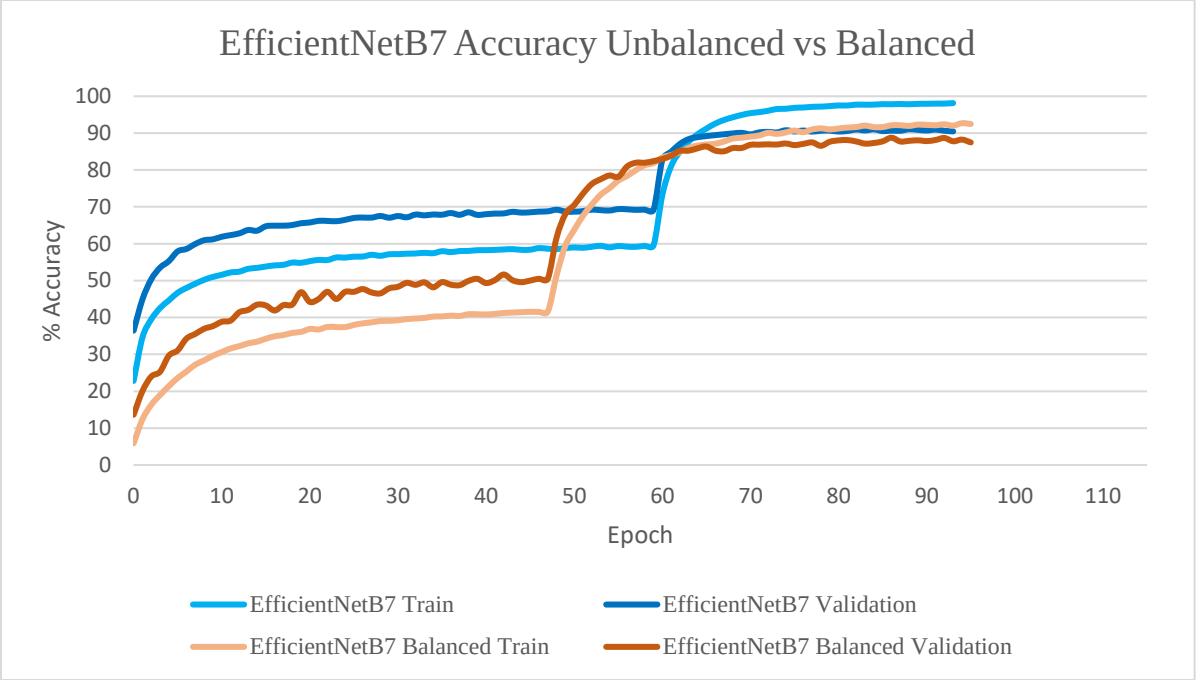

Figure 11. EfficientNetB7 Balanced vs Unbalanced Accuracy Over Epochs

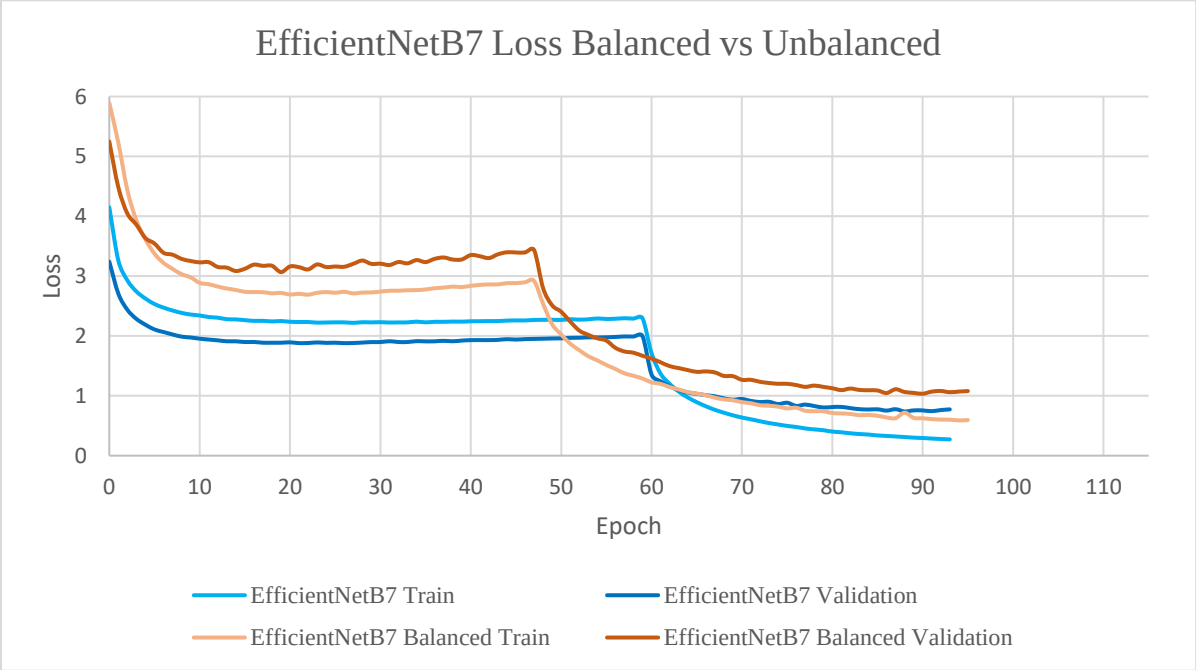

Figure 12. EfficientNetB7 Balanced vs Unbalanced Loss Over Epochs

Both graphs show that over time the performance of the model improves. They also both end training near the same number of batches and have similar diverging.

The macro average precision, recall and f1-score were found along with accuracy for both the models. The results of this are shown below (Table 10).

Table 10. EfficientNetB7 Vs Balanced EfficientNetB7

Model	Precision	Recall	F1-score	Accuracy
Unbalanced EfficientNetB7	**.89**	.80	**.84**	**.90**
Balanced EfficientNetB7	.86	**.82**	.83	.87

The balanced EfficientNetB7 model obtained better recall at the cost of lower precision, f1-score, and accuracy.

Image Sizing:

This section of the study tests whether input image size can have an impact on performance. All the models tested here use InceptionResNetV2 with no weight balancing however, there are 3 different input image sizes. These are 768x768, 512x512, and 256x256.

The 768x768 confusion matrix is shown below (Figure 13).

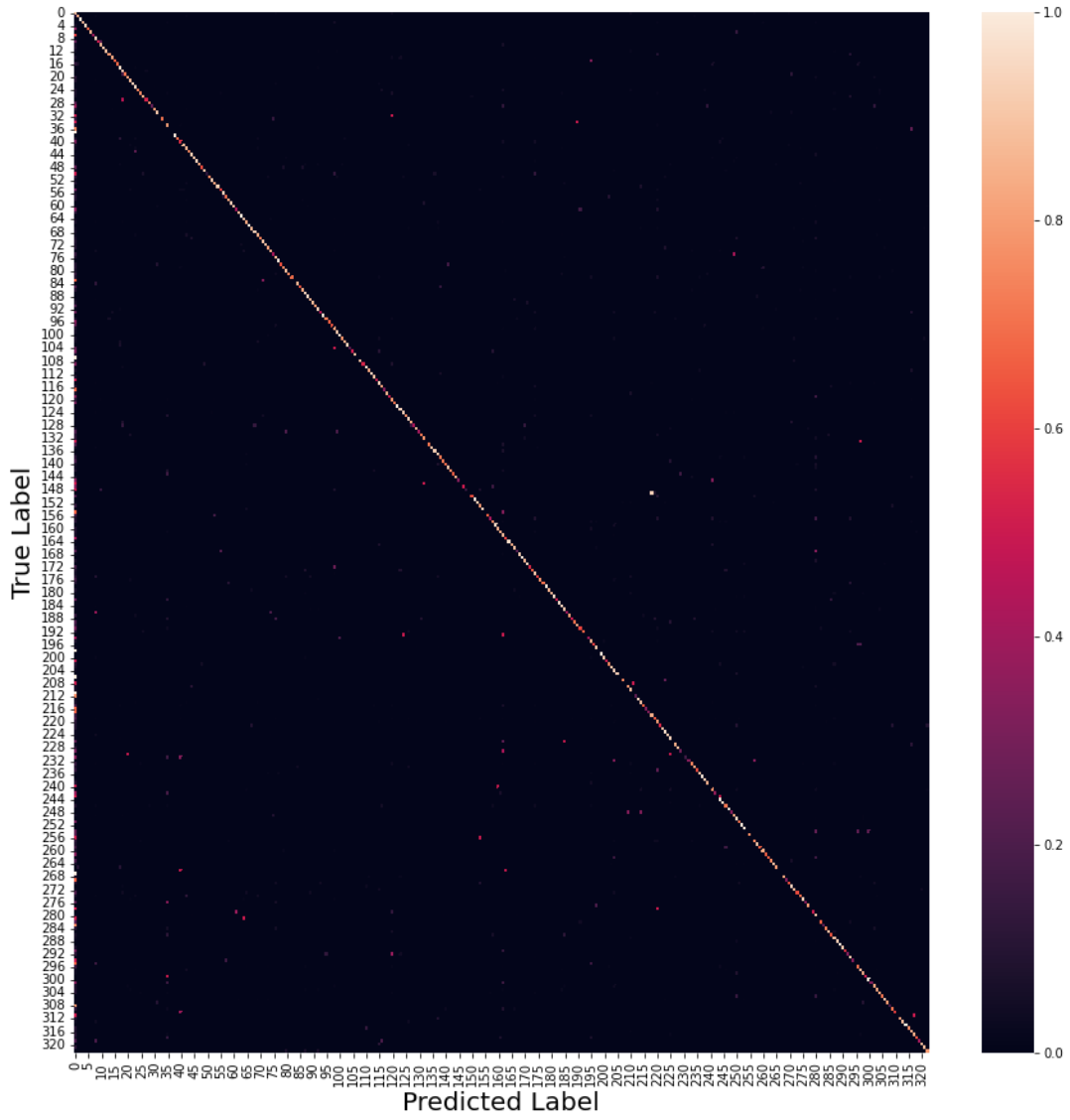

Figure 13. InceptionResNetV2 768x768 Confusion Matrix

Most classes are accurately predicted but there are some classes that are incorrectly identified and out of this misidentified prediction some of them are consistently misidentified with one other class.

The 256x256 confusion matrix is shown below (Figure 14).

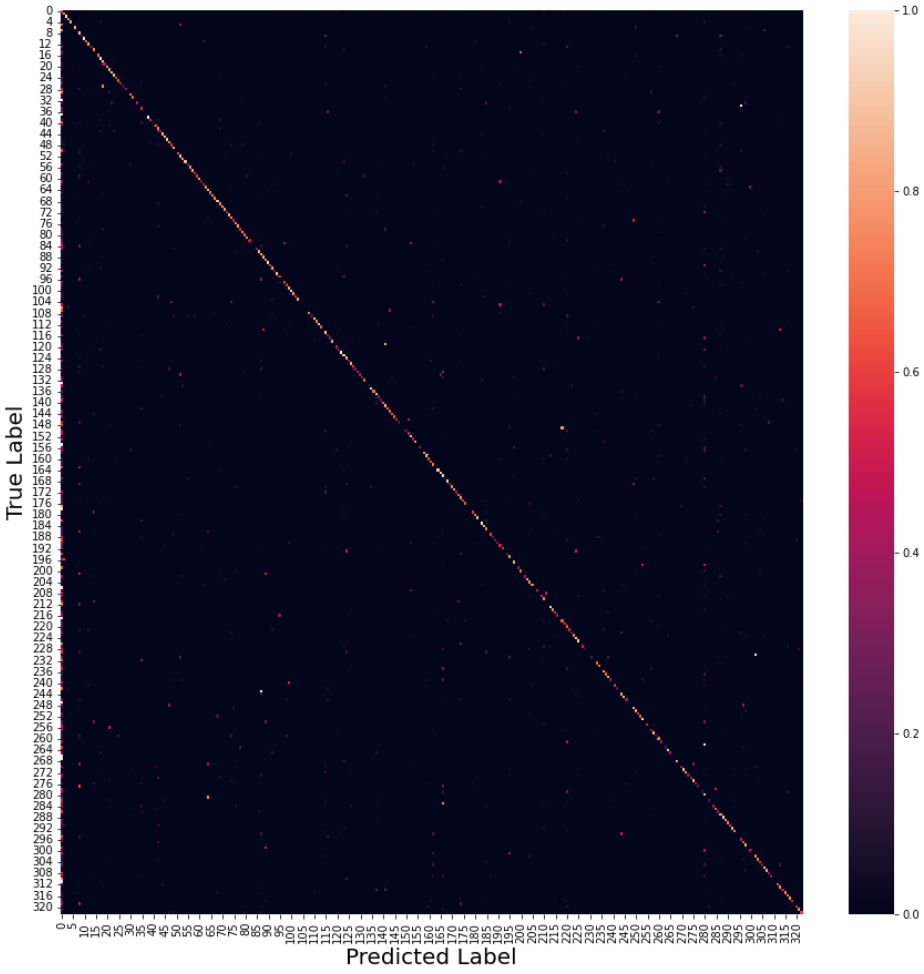

Figure 14. InceptionResNetV2 256x256 Confusion Matrix

Most classes are accurately predicted but there are some classes that are incorrectly identified and out of this misidentified prediction some of them are consistently misidentified with one other class and at a higher rate than that of the 512x512 model.

The Tensorboard graphs below show the training and validation accuracies and loss over time with the x-axis representing the number of epochs and the y-axis representing the accuracy for the 512x512, 768x768, and 256x256 models (Figure 15) (Figure 16).

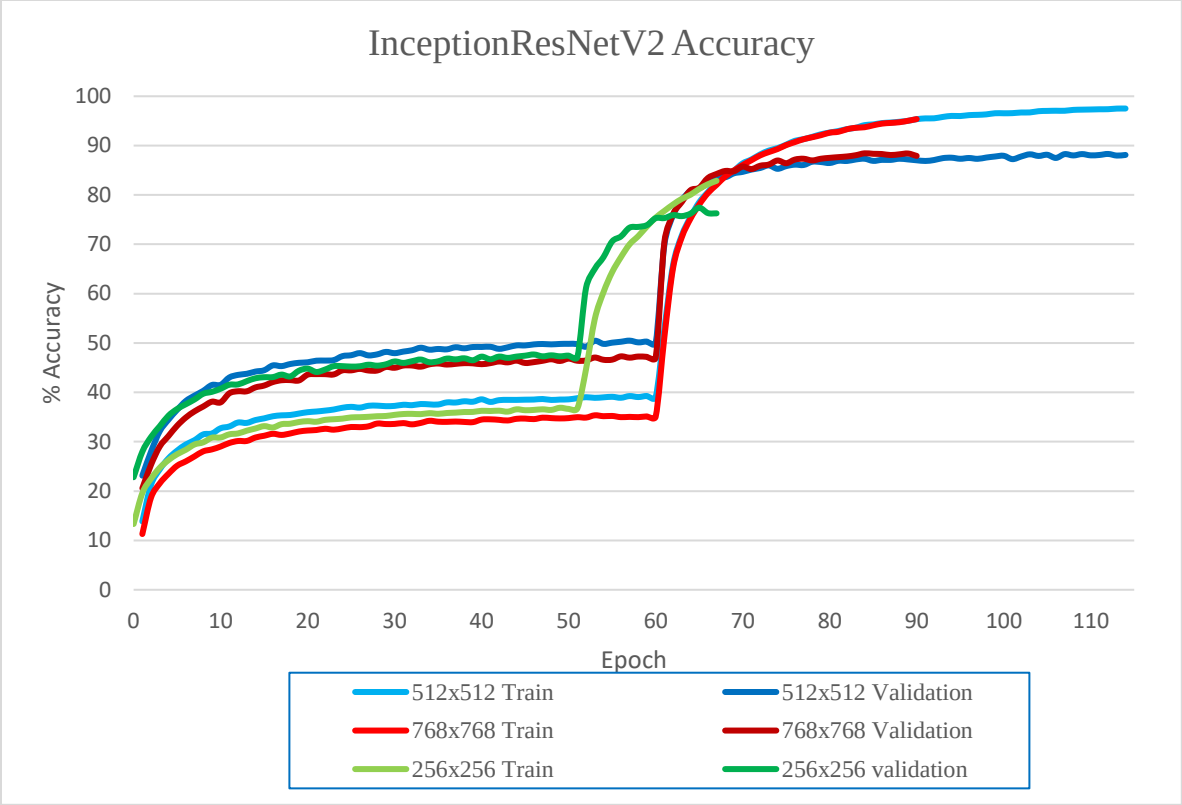

Figure 15. InceptionResNetV2 512x512 vs 256x256 vs 768x768 Accuracy Over Epochs

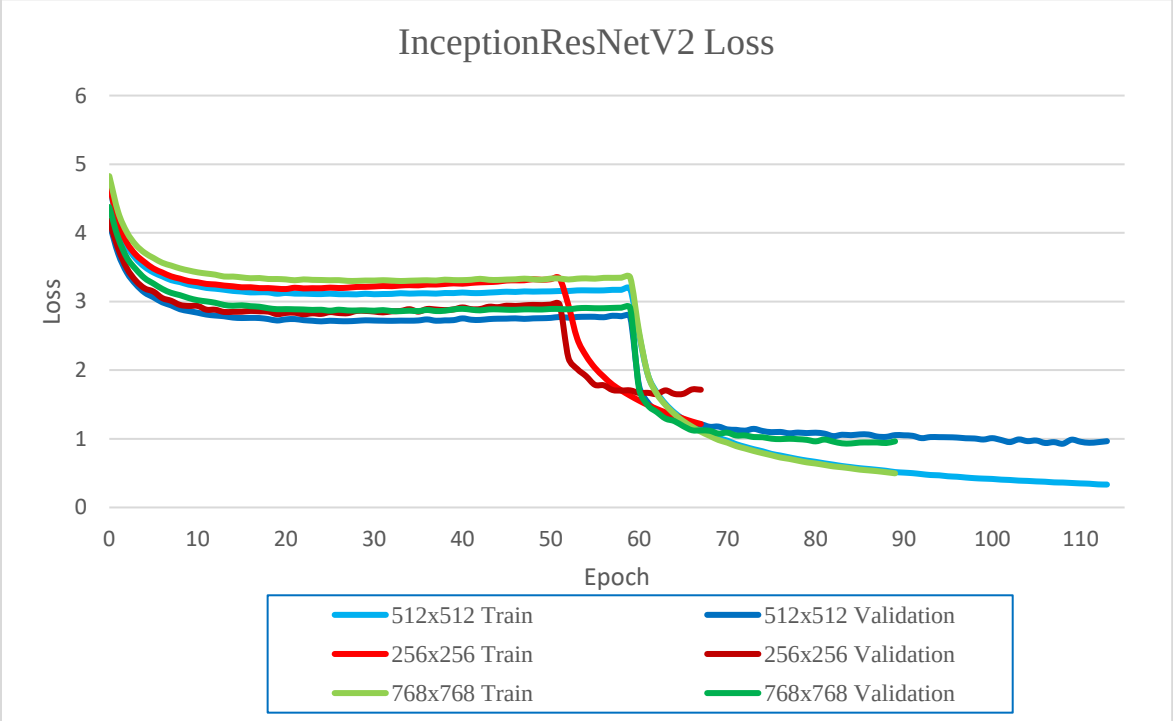

Figure 16. InceptionResNetV2 512x512 vs 256x256 vs 768x768 Loss Over Epochs

Both graphs show that over time the performance of the model improves. The 256x256 model takes the shortest amount of time with the lowest performance while the 512x512 takes the longest time and both the 512x512 and 768x768 achieve similar performance. All of the models diverging at the end of the training seem to be of a similar size.

The macro average precision, recall and f1-score were found along with accuracy for both the models. The results of this are shown below (Table 11).

Table 11. InceptionResNetV2 512x512 vs 256x256 vs 768x768

Model (InceptionResNetV2)	Precision	Recall	F1-score	Accuracy
512x512	.83	**.75**	**.77**	.87
256x256	.70	.51	.56	.75
768x768	.83	.70	.74	.87

The 256x256 model has the lowest precision, recall, f1-score and accuracy out of all the models. Precision and accuracy are the same for the 512x512 and 768x768 models. The 512x512 model achieves the highest recall and f1-score of all the models.

CHAPTER 5: Discussion

Key Findings:

The objective of this study is to determine if deep learning is a realistic method for identifying arthropods to the taxonomic level of genus, and to determine the most optimal machine learning methods that may be used for this specific goal.

Based on the full model's results 80% of predictions were correctly classified and 92% of predictions contained the true label in the top 5 results. This shows that the model created can be a realistic method for identifying arthropod genera across North America.

While these results are promising this study also wanted to determine if a more optimal model could be created. To do this, 6 models were tested and compared to determine 3 different methods for increasing model performance. These are changing the neural network architecture, performing class balancing, and changing input image size.

To test whether changing the neural network architecture played a role in model performance the control architecture, InceptionResNetV2, was tested against EfficientNetB7. The results of this showed that for precision, recall, f1-score, and accuracy EfficientNetB7 performed better than InceptionResNetV2. It also achieved this performance in less time than the InceptionResNetV2 model. Therefore, this indicates that changing the underlying neural network of a model can increase performance.

Class balancing was performed on both InceptionResNetV2 and EfficientNetB7. For InceptionResNetV2 it was found that there was no increase in performance for the balanced model and the only metric that did not decrease was recall which stayed the same. Therefore, for InceptionResNetV2 it is detrimental to use the class balancing algorithm used. This lack of increase in recall for InceptionResNetV2 could be due to the balancing of the rare classes not overcoming the weight of the common classes. This idea may be supported with the large decrease in accuracy between the balanced and unbalanced datasets which could happen because while the balancing is enough to move a prediction from the common class but not enough to move the prediction to the true prediction. Class balancing was also used on EfficientNetB7 where the balanced model performed better on recall but lower on the other metrics. This is an expected result where class balancing usually increases recall at the cost of other metrics like precision. Therefore, if recall is to be maximized for EfficientNetB7, using class balancing would be an effective method.

Input image size comparisons were performed on InceptionResNetV2 using 512x512, 768x768, and 256x256 images. It was found that 256x256 performed worse than the other two models on all metrics. This indicates that as you decrease image size there may be a loss in features available for learning and thus performance can degrade. Comparing the 512x512 and 768x768 models the 512x512 model achieved better recall and f1-score whereas the other metrics were equal between the two models. This indicates that there is an optimal size of images and that performance does not increase indefinitely with increasing image size most likely due to that features on the image being learned on are present in both models. This could be cause because both databases, BugGuide and iNaturalist, have varying image sizes and if most input images are smaller than 768x768 then upscaling the image would not provide more useful features for the model to learn on.

Limitations:

While our results are promising there are other strategies that could increase the accuracy and efficacy of our model.

When web scraping our data from the citizen science databases, we did not do any data sanitization. While earlier in the paper we described the overall effect on our dataset as negligent to the learning this lack of data sanitization could lead to errors in the accuracy of identification in specific classes. For example, if images that were incorrectly identified, include artifacts, are of low image quality, or are of generally low quality for deep learning tasks were not common and evenly distributed this would not have a large impact to the accuracy of the model; however, if these were common for a particular class this could lead to a large impact in accurately identifying that class.

There are more robust methods to balance classes and the method we used to compare class balancing between our models was relatively rudimentary. For example, a method like SMOTE, while harder to implement, has shown promising results in dealing with the imbalanced class problem. A future study finding the benefits of one balancing method over another for our data could be helpful in determining if the error found between these methods is significant. Ecological identification methods have suffered from not being able to have adequate solutions for imbalanced datasets, scarce data and open-world applications (Villon et al., 2022). While some of these were tested for like looking at how our data is balanced and using large amounts of data looking at how we could tackle these differently and focusing on how to approach the new classes in open-world applications is something that needs to be addressed.

Significance:

The results of this study show that machine learning is a viable method to identify arthropods from images and methods for increasing performance can be used in these types of applications. This is important because the use of machine learning techniques could be the next frontier in biology that could lead to more robust identifications of arthropods that lack bias and expediate tedious and time-consuming identification that are part of many studies of arthropods. This solution is robust enough to compete with experts and has enough variability in classes to compete in many different fields of study.

Future Work:

In the future this study will try to move towards optimization of the classification model using recent advancements seen in deep learning to create a more accurate model with more features. Combining the model created in this study with object detection to identify and pinpoint areas of interest in video data would allow for the ability of the model to differentiate images with multiple specimens in one image and create individualized specimen counts for each genera.

Object Detection:

Object detection is the process of finding the areas of interest in an image and highlighting that area, usually through a bounding box. This can be done by attaching the YOLO network to our neural network before training. YOLO works by partitioning the image into rectangles and finding the accuracy of each partition and then partitioning the most accurate section over and over until you find the partition with the highest accuracy. This allows the model to determine what part of the image is most likely to contain the object that is trying to be predicted. You can do this for multiple classes and allows multiple classes to be in a single image as well to count the number of instances of a specific class. After object detection is completed other issues can be addressed such as those posed with open-world applications (Villon et al., 2022). One of these issues that can be addressed with an object detection model is including an “unknown” class.

Chapter 6: Conclusion

Deep learning techniques can revolutionize how we approach many areas of science. While many areas of ecology and biology have been improved through deep learning, few attempts at training models capable of recognizing thousands of organisms on a continental scale have been made. While there have been attempts at arthropod identification in entomology, most of the applications have been focused on models identifying crop pests and specific groups of arthropods. The work shown here shows that leveraging citizen science data we can identify arthropods with high confidence and accuracy. We found that deep learning is a viable tool for large-scale identification of arthropods with our result of 80% top-1accuracy on 7,524 classes/genera. This work compared various training strategies and showed that advancements in deep learning techniques including neural network architectures, image processing and detection will further increase the accuracy and efficiency of automated species identification. This could lead to increased accuracy in research and greater variability in what can be identified and compared in studies. The increased speed of identification could also lead to faster research or even speed up diagnosis of arthropod related conditions. It was shown that changing the neural network architecture led to higher performance of the model, that class balancing can lead to increased recall, and that increasing input sizes of images can correlated with better performance, but this increase also is logarithmic in some cases. While this model achieved the goals of this study future research into optimizing our model such as through the use of object detection could expand the use cases of the model.

References

Åkesson, A., Curtsdotter, A., Eklöf, A., Ebenman, B., Norberg, J., & Barabás, G. (2021). The importance of species interactions in eco-evolutionary community dynamics under climate change. *Nature Communications*, *12*(1), 4759. https://doi.org/10.1038/s41467-021-24977-x

Anonymous. (2022). *Welcome to BugGuide.Net!* BugGuide. https://bugguide.net/node/view/15740

Boer, M. J. A., & Vos, R. A. (2018). *Taxonomic Classification of Ants (Formicidae) from Images using Deep Learning* [Preprint]. Bioinformatics. https://doi.org/10.1101/407452

Bug ID - Using Machine Learning to Identify Arthropods. (n.d.). Retrieved March 27, 2022, from https://bugid.nkn.uidaho.edu/

Clark, A. (2015). *Pillow (PIL Fork) Documentation*. readthedocs. https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Ding, W., & Taylor, G. (2016). Automatic moth detection from trap images for pest management. *Computers and Electronics in Agriculture*, *123*, 17–28. https://doi.org/10.1016/j.compag.2016.02.003

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for Multi-Class Classification: An Overview. *ArXiv:2008.05756 [Cs, Stat]*. http://arxiv.org/abs/2008.05756

Hansen, O. L. P., Svenning, J.-C., Olsen, K., Dupont, S., Garner, B. H., Iosifidis, A., Price, B. W., & Høye, T. T. (2020). Species-level image classification with convolutional neural network enables insect identification from habitus images. *Ecology and Evolution*, *10*(2), 737–747. http://dx.doi.org/10.1002/ece3.5921

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hecht-Nielsen, R. (1992). Theory of the Backpropagation Neural Network. *Neural Networks for Perception*, *2*, 65–93.

Huh, M., Agrawal, P., & Efros, A. A. (2016). What makes ImageNet good for transfer learning? *ArXiv:1608.08614 [Cs]*. http://arxiv.org/abs/1608.08614

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

IIDS NKN. (n.d.). Retrieved March 27, 2022, from https://www.iids.uidaho.edu/nkn.php

INaturalist. (n.d.). INaturalist. Retrieved March 25, 2022, from https://www.inaturalist.org/

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), 27. https://doi.org/10.1186/s40537-019-0192-5

Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., & Zhang, G. (2015). Transfer learning using computational intelligence: A survey. *Knowledge-Based Systems*, *80*, 14–23. https://doi.org/10.1016/j.knosys.2015.01.010

Mahesh, B. (2019). *Machine Learning Algorithms -A Review*. https://doi.org/10.21275/ART20203995

Mathison, B. A., & Pritt, B. S. (2014). Laboratory Identification of Arthropod Ectoparasites. *Clinical Microbiology Reviews*, *27*(1), 48–67. https://doi.org/10.1128/CMR.00008-13

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Merkel, D. (n.d.). *Docker: Lightweight Linux Containers for Consistent Development and Deployment*. 5.

Occdownload Gbif.Org. (2020). *Occurrence Download*. The Global Biodiversity Information Facility. https://doi.org/10.15468/DL.TQ384K

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *ArXiv:1602.07261 [Cs]*. http://arxiv.org/abs/1602.07261

Tan, M., & Le, Q. V. (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv:1905.11946 [Cs, Stat]*. http://arxiv.org/abs/1905.11946

team, T. pandas development. (2020). *pandas-dev/pandas: Pandas* (latest) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

Ueda, K. (2019, October 25). *Identification Quality On iNaturalist—General*. INaturalist Community Forum. https://forum.inaturalist.org/t/identification-quality-on-inaturalist/7507

Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., & Belongie, S. (2018). The iNaturalist Species Classification and Detection Dataset. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8769–8778. https://doi.org/10.1109/CVPR.2018.00914

Villon, S., Iovan, C., Mangeas, M., & Vigliola, L. (2022). Confronting Deep-Learning and Biodiversity Challenges for Automatic Video-Monitoring of Marine Ecosystems. *Sensors*, *22*(2), 497. http://dx.doi.org/10.3390/s22020497

Appendix A: Complete List of Components Used

COMPONENTS	QUANTITY
VWS-1542881-DPN-BLACK STEEL / PLASTIC MINI-ITX, MICRO-ATX, E-ATX CUBE HIGH AIRFLOW CUBE CASE	1
X299 BASED MOTHERBOARD, SUPPORTS 1X LGA2066 CPU, 2X M.2 PCIE MODE2X M.2 PCIE MODE, 8X DDR4 SLOTS, 2X GBE NIC, 7X PCIE X16 SLOTS (QUAD X16 WITH 44 LANE CPU)	1
INTEL CORE I9-7900X 10 CORE PROCESSOR 13.75M CACHE BASE 3.30GHZ BOOST UP TO 4.30 GHZ 140W LGA2066	1
LIQMAX III HF CPU LIQUID COOLER	1
16GB DDR4 2666MHZ DESKTOP MEMORY MODEL	6
4TB WESTERN DIGITAL HDD DRIVE	1
NVIDIA RTX 2080 TI 11GB GPU	4
2000W MODULAR ATX PS2 POWER SUPPLY 80PLUS PLATINUM COMPLIANT	1
UBUNTU 16.04	1
DOCKER W/ NVIDIA-DOCKER WRAPPER, DOCKER IMAGE INSTALLATION/TESTING WITH AVAILABLE OPENSOURCE DOCKER IMAGES, NVIDIA/:EXXACT_DIGITS_PAGE:,:TENSORFLOW/TENSORFLOW, NVIDIA/CAFFE, ETC.	1