BIRD CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This project looks at the effect of noisy data and class count on convolutional neural network performance in bird classification. One downside to convolutional neural networks is the computational power they require. Thus, we also used semi-supervised learning in our training to test its affect on computation time and accuracy. Our results show quicker individual training times with noisy data being unfit for use. These results support the use of semi-supervised learning in models which require updating over time, such as once a month.

1 INTRODUCTION

Birds are one of the only ubiquitous species on the planet, and being able to identify whichever bird you took a picture of can be quite difficult unless someone spends a large portion of their time dedicated to learning the differences between specific species. What if it was possible to use Machine Learning and Computer Vision to solve this problem instead? In order to improve the birding experience and to soften the barrier of entry, as well as to gain a better understanding of popular image classification methods, we have endeavored to explore potential classification models through trying to classify different bird species.

In order to do this we had to find data. We gathered two major datasets, one through our own webscraping script and one through Kaggle. We also used those to create two minor datasets which consisted of the bird species that are shared between each of the major datasets. This data was chosen for this project because the scraped dataset represents a ”real world” dataset with many flaws and a wide variation of quality while the Kaggle dataset represents a very clean, well maintained dataset.

On this data, multiple Convolutional Neural Networks **(CNN)** and Semi-supervised Learning **(SSL)** models were applied. Overall, we found that the cleaned data, and thus the models applied to the Kaggle dataset, performed significantly better than the alternative in all comparisons. Additionally, we found that the models we built on the minor datasets performed significantly better than those on the full dataset, particularly due to less classes.

2 DATA

In order to accomplish our analysis, we have compiled four datasets for analysis. The first two datasets are the **Kaggle** and **scraped** datasets. The scraped was gathered through a webscraping code which we developed to scrape the Cornell Macaulay Library, which is a free resource of crowd sourced bird images for the purpose of machine learning. 157 classes (species) of birds were gathered through this scraping, with up to 100 images being collected for each class. Overall, there were a total of 14,643 images in this dataset belonging to the 157 classes. These images are of drastically varying quality, with some images being of comparable quality to the Kaggle dataset and some images being significantly worse quality and almost unusable.

Alternatively, the Kaggle dataset was gathered through Kaggle in the following link. This Kaggle source has over 500 classes, but was reduced down to 167 classes randomly. There are a varying number of images per class in this dataset, but there are a total of 27,006 images in this dataset belonging to the aforementioned 167 classes combined.

These two datasets were selected to represent an ideal and problematic dataset which we could work with in the real world. The kaggle dataset consists of a large number of very clean, cropped data.

This is meant to represent an ideal set of images that one would hope to have. Examples of the differences in the data can be seen in figure 1, as well as in the appendix.

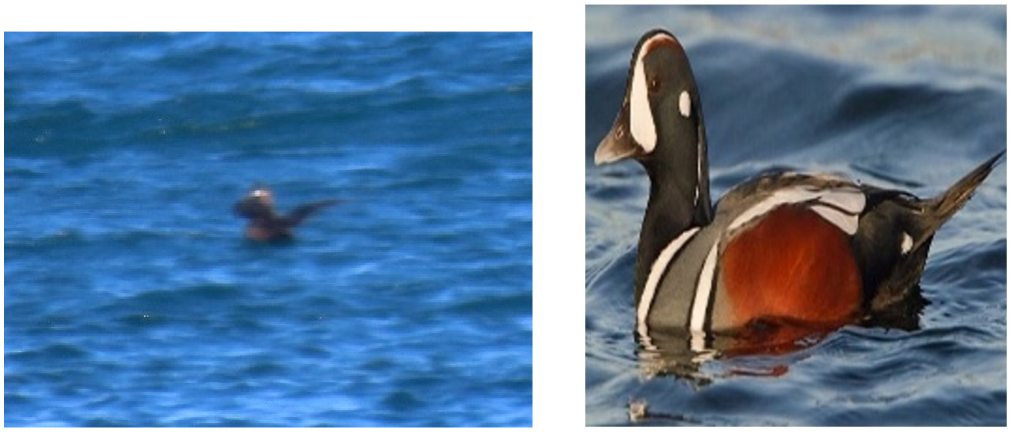


Figure 1: **(Left)** An example of blurry, low quality image from the scraped dataset **(Right)** An example of the same class but from the kaggle dataset. More examples are provided in the appendix.

However, both datasets suffer from one large problem: Class similarity. In the complete datasets, there are many different classes that look incredibly similar, especially when the images are of a lesser quality. This is a major problem in many image classification datasets, and we wanted to address how large of an impact this might have on our analysis. One example of two sparrows can be seen in figure 2, and more can be seen in the appendix.

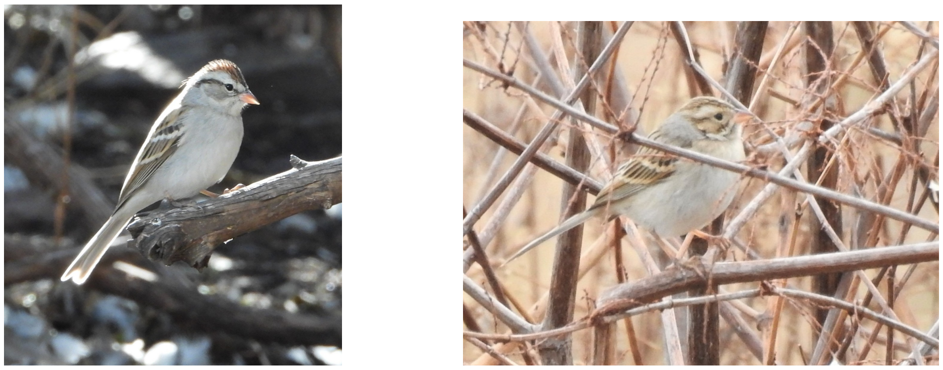


Figure 2: An example of two similar classes. Above are two different types of sparrow, but the difference between the two is difficult to determine.

In order to address the issue of class similarity, we also created two ”partial” datasets. Each one of these partial datasets were created by taking all the classes from each ”full” dataset that are shared between both. Overall, there are 27 classes which are shared between the two larger datasets, so each of these smaller datasets only has those like classes. The partial scraped dataset has a total of 2609 images and the partial kaggle dataset has a total of 4441 images. All data statistics are summarized in table 1.

3 Methods

The main model used in our project was a Convolutional Neural Network (CNN). CNNs are extremely common in image recognition due to their ability to learn spacial features from input data. Because CNNs work with convolutions and not raw pixel data, they are less impacted by noise, such as backgrounds, positions, lighting, and angles. Furthermore, there is less of a need for feature engineering because of the convolutions.

Data Name	Classes	Images
Full Scraped	157	14,643
Full Kaggle	167	27,006
Partial Scraped	27	2,609
Partial Kaggle	27	4,441

Table 1: Dataset Details

CNNs are made up of several layers. These include convolutional layers, pooling layers, and fully connected layers. In the convolutional layer, the model applies a filter to the image to produce a set of activation maps. It stores information in a matrix of size image length X image width X 3 (RGB) values. We used a 3 x 3 convolutional filter in order to extract the optimal values. We then applied the Rectified Linear Unit (ReLU) transformation function to each convolutional layer.

We also used max pooling layers. Pooling reduces computation time by downsampling the features from the convolutional layer. Since we used max pooling, we took the maximum value from a filtered area to be our representative value.

The fully connected layer is used to generate a score, or a prediction of what class the input belongs to, based on the output of the previous layers.

In our case, our model contained 5 convolutional layers and 5 max pooling layers. We also had a dense layer consisting of 64 hidden nodes, which outputted the class prediction.

The loss function we used was Sparse Categorical Cross Entropy. Entropy represents the uncertainty of the possible outcomes of a variable. If there is high entropy, there is high uncertainty for the distribution. It also penalizes confidently incorrect predictions more than unconfident incorrect predictions, meaning the model takes into account the degree of how wrong a position was. Sparse cross entropy in particular uses one hot embeddings to save memory and time. This is useful when dealing with large datasets with many classes, as was the case for us on this project.

The optimization function we used was the Adaptive Moment Estimation (Adam). This function adapts a learning rate for each variable in the network. The gradient moves quick initially but then slows over time. The function is also computationally efficient, which works well for our large amounts of data. This function is widely used for all sorts of machine learning problems.

4 SEMI-SUPERVISED LEARNING (SSL)

Because of the training time of the convolutional neural networks, we looked to find a compromise between time and accuracy. The models containing all classes took 23 hours to train while the models using the intersection of classes took 4.5 hours to train. This led us to semi-supervised learning, which combines methods of supervised and unsupervised learning. Supervised learning provides greater accuracy while unsupervised learning is less computationally intense.

A semi-supervised model begins by training a model on a small portion of the labelled data. For example, 25% of the total data with a standard train/validation split. This model will be trained in the same manner as the final model. This initial trained model is then used on the remaining data, which has the labels removed. This is the unsupervised portion of the training. Unlabelled points with high prediction confidence, such as 80%, then have this predicted label applied to them and are added back to the original set of training data. This is called pseudo-labelling. The training and pseudo-labelled dataset is then used to retrain the training model. This is repeated a parameterized number of times, often five to ten, with accuracy determined by a test set.

Semi-supervised data does not work for all datasets. If semi-supervised data is done correctly on a fitting dataset, the models should see improving results over iterations of pseudo-labelling. Additionally, good supervised/unsupervised splits should also yield performance improvements.

5 Applications

In this section we will discuss our specific models that we created.

5.1 Traditional Models

In order to have something to compare our Semi-Supervised learning approaches, we built a normal CNN model for each dataset as described above.

In order to keep results comparable, each of these models were built using the exact same parameters. Each model was built to have 5 layers, with each layer consisting of a 64 node conv2d layer followed by a 2x2 max-pooling layer. At the end of these 5 layers, the results were flattened and ran through one 64 node dense layer before final predictions were generated. Each model used 100 epochs in order to ensure that the models are able to train long enough. Additionally, a batch size of 32 was used to improve general runtimes.

In building these models, we experimented with different parameters; namely with the depth of our neural network. We attempted to run these CNNs with 1-5 layers, and found that 5 layers performed significantly better than the alternatives. Additionally, we attempted running these models for fewer epochs in order to improve runtimes, but found that any fewer epochs could potentially lead to significantly poorer performance.

A train / val / test split of 80/10/10 was also used for all models to ensure that enough data was used for each individual step.

These models had a significant number of parameters. Parameter information is outlined below in table 2.

Model Name	**Parameters**
Full Scraped	287,901
Full Kaggle	182,119
Partial Scraped	279,451
Partial Kaggle	173,019

Table 2: Traditional Model Details

5.2 Semi-Supervised Learning Models

We created two semi-supervised models on the partial data containing the intersecting classes between the scraped and Kaggle datasets. The models are meant to provide a view into how the performance changes differ going from standard training to semi-supervised between noisy and clean data.

The models are the same as the previously described traditional models in terms of network size and layers. This mean five convolution and pooling layers, a flattening layer, and a dense neural network with one hidden layer containing 64 nodes. The scraped model still has 279,451 parameters while the kaggle model has 234,395.

When it comes to the semi-supervised learning specific aspects, the models use the same specifics to allow for direct comparison between the two models. We used five run throughs of pseudo-labelling steps, with the model training using 50 epochs per trained model. 90% of data was reserved for semi-supervised learning, with the remaining 10% used for testing. Of the 90% reserved for semi-supervised learning, half of that was used for initial training with the other half for pseudo-labelling. This means 45% of data was labelled and 45% was unlabelled. When training the model, 90/10 train/test splits were used. Finally, predictions were considered significant enough for psueo-labels when they were 90% confident.

6 RESULTS

Part 1: Traditional Full data models

Model Name	Accuracy	Loss	Runtime (Minutes)
Traditional Full Scraped	0.3507	2.5929	1038.3
Traditional Full Kaggle	0.6350	1.4006	1033.3

Table 3: Traditional Full Model Results

Part 2: Traditional Partial Model

Model Name	Accuracy	Loss	Runtime (Minutes)
Traditional Partial Scraped	0.6797	1.1161	259
Traditional Partial Kaggle	0.8365	0.6269	128.3

Table 4: Traditional Partial Model Results

Part 3: SSL models

Model Name	Accuracy	Loss	Runtime Per Loop (Min)	Total Runtime (Min)
SSL Scraped	0.3374	2.4874	79.2	396
SSL Kaggle	0.6411	1.5573	115	575

Table 5: SSL Model Results

7 FINDINGS

The results return what we expected given our model specifications. The effects of class count and semi-supervised learning did yield interesting observations, however. To begin, the Kaggle models with clean data always performed significantly better than the scraped models with noisy data. The closest performance different was between the partial class data without semi-supervised learning, where the scraped model yielded 67.97% accuracy while the Kaggle model yielded 83.65%. This is in stark contrast to the full and SSL applications of each model, where the scraped results performed 28.43% and 30.37% better respectively. Additionally, the noisy data does not appears to be fit for SSL training based on its low accuracy on a low class count. Based on this information, there are three clear takeaways.

First, the great increase in classes also coincides with a decrease in accuracy. This is to be expected, as there are more incorrect choices for the prediction. In a similar vein, noisy data has a much larger impact on accuracy than we expected, especially in the SSL model. This is likely due to the poor images in the scraped data causing less confident predictions, which leads to an SSL loop progression that simply does not add many pseudo-labels over time, if they are even correct. The other takeaway is that the SSL model provides greater efficiency for real world applications. While the total run time is significantly longer, the loops take much less time to train than the traditional model does. In a real world application where data is gathered over time, this would allow a semi supervised model to perform much better. For example, once enough data has been accrued, an initial SSL model can be trained. After that, as more data comes in (eg. monthly), the model can then undergo a semi-supervised loop where pseudo-labels are assigned and the model is retrained for updated results, allowing the model to prevent having to spend more time training on copious amounts of unnecessary data. This would increase initial training time, but would then require much less resources to be allocated to retraining the model in the future because of the shorter individual loop training time.

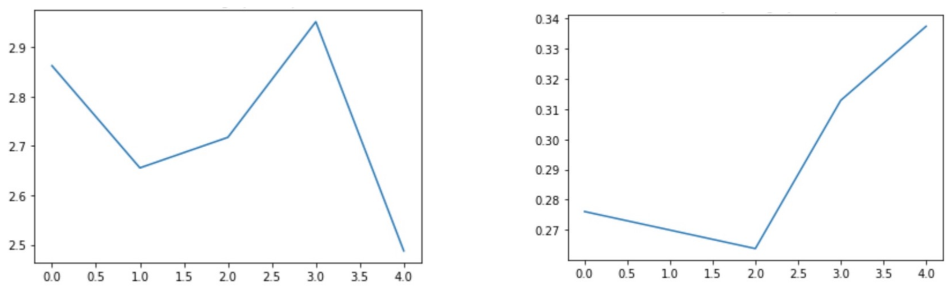


Figure 3: **(Left)** The progression of the test loss from the scraped SSL model through each epoch.**(Right)** The progression of the test accuracy from the scraped SSL model through each epoch.

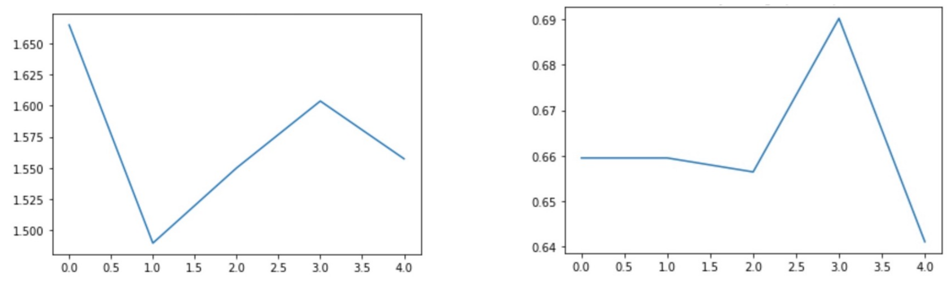


Figure 4: **(Left)** The progression of the test loss from the Kaggle SSL model through each epoch.**(Right)** The progression of the test accuracy from the Kaggle SSL model through each epoch.

One last thing to note is the interesting progression of the test loss and accuracy per loop in the SSL models. The scraped progression is noted figure 3 while the Kaggle progression is noted in figure 4. Neither model sees a constant decrease in the loss function or a constant increase in accuracy on the test set. The scraped model has its lowest accuracy three loops in and its worst loss four epochs in. However, its accuracy peaks at its final loop. Meanwhile, the Kaggle model has its lowest loss one loop in, and its accuracy peaks at its fourth loop while its worst accuracy is in its final loop.

This is an interesting observation, and there is not an immediately clear reason. It is worth noting that in the case of the Kaggle model the accuracies are not wildly different. It is likely that this irregularity in testing accuracy progression is caused by some confident incorrect pseudo-labels influencing the model training in the final loop, which would explain why the cross entropy is not at its lowest in the final loop. Additionally, the Kaggle model psuedo-labelled about 80% of the data reserved for unsupervised training. Since so many points become psuedo-labelled, it is likely that a more efficient implementation of semi-supervised learning for this data would be to perform supervised learning on 25% of data with 65% reserved for pseudo-labelling and the rest for testing. This would allow the model to run faster without essentially training the model on 80% of the training data by the final loop.

8 Discussion and Further Research

Throughout the procession of our project, we were exposed heavily to the trade-off between efficiency and accuracy. Some of our models that were the most accurate also ran the longest with entire training periods taking upwards of eight hours to run. In addition, a task like cleaning the images could be extremely tedious and could take much longer than model training itself since it would need to be done manually in the scope of this project. These high processing times for computationally expensive tasks like image classification are not uncommon, and luckily, there are some alternative methods besides CNNs to tackle the problem.

Semi-Supervised learning was an approach we added to the scope of our project, and its underlying method of using some labeled and some unlabeled data made it certainly run faster per epoch than our entirely supervised models that used only labeled data. Another approach we could have added to our supervised and semi-supervised models would be the use of Stochastic Gradient Descent (SGD) instead of Adam as our optimizer. Using SGD with large step sizes can result in faster convergence and better generalization performance in some cases. With larger step sizes, the optimization algorithm can make more significant updates to the parameters at each iteration, which can help to escape from local minima in the loss landscape and converge faster. For large datasets like the ones we used, SGD can tend to be better because of its efficient nature. However, due to time and computational constraints, we were not able to implement this approach, but we predict that it would shorten run time by converging quicker than some of the models using Adam.

Like Semi-Supervised learning, transfer learning is another popular technique in deep learning that can be applied to improve the performance and efficiency of image classification models like our bird classification task. By using pre-trained models, such as VGG, Inception or ResNet, and replacing their last layers with new ones trained for the bird classification task, models can be quickly and efficiently trained on the new task (in our case, bird image classification) while still benefiting from the knowledge learned from the pre-trained model. This approach can reduce the amount of labeled data needed to train a new model and improve the accuracy and efficiency of the model. This is another possible approach we could've used on our large datasets, and we would expect much faster run time and possibly even better accuracy.

In conclusion, for a task like image classification, there is an expansive array of currently popularized methods that could be employed for the task. Our application of CNNs and Semi-Supervised models gave us a strong start to classifying large image datasets, but there is still a great variety of other methods that we could've experimented with. As for any program or project, time and computational resources are constraints that must be kept in mind, and we found the balancing of these constraints with the retaining of accuracy to be an advanced task.

9 APPENDIX

Part 1: Image Quality

There are many different flaws present in the scraped dataset. These flaws include far away / blurry images (shown in figure 1), poorly balanced / high contrast images (shown in figure 6), many birds of the same class or of multiple classes too far away to accurately classified (shown in figure 7), partially or fully obscured birds (shown in figure 12). However, despite this there are still a good portion of quality images in this dataset (shown in figure **??**). Additionally, 2 shows another example of classes being similar which could potentially impact results.

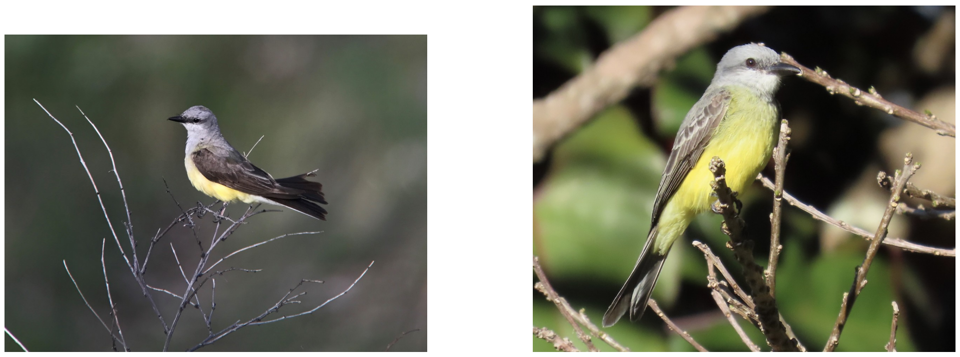


Figure 5: Above is another example of similar classes. **(Left)** An example of a Western Kingbird **(Right)** An example of a Tropical Kingbird

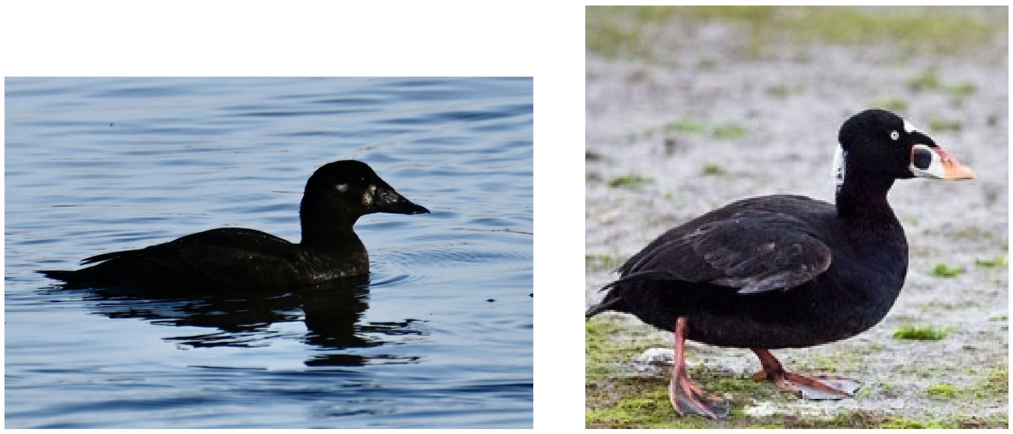


Figure 6: **(Left)** An example of a poorly balanced image from the scraped dataset **(Right)** An example of the same class of bird from the kaggle dataset

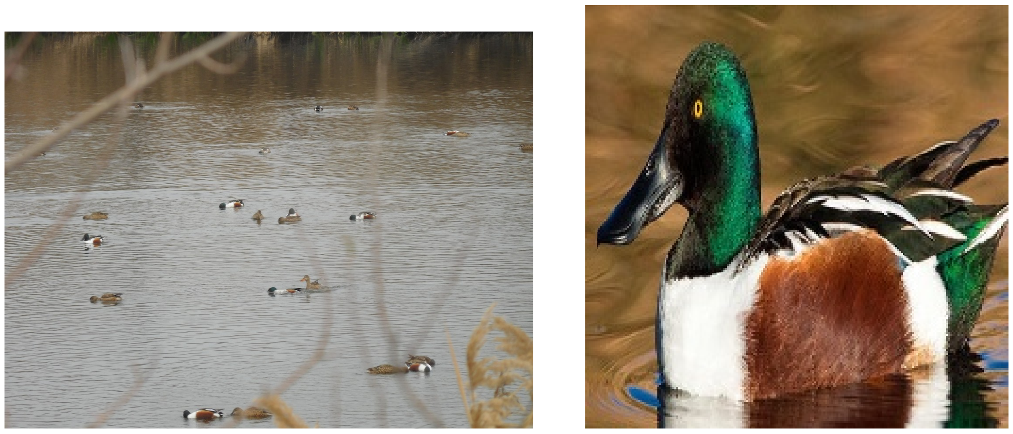


Figure 7: **(Left)** An example of many birds in one picture from the scraped dataset **(Right)** An example of the same class of bird from the kaggle dataset

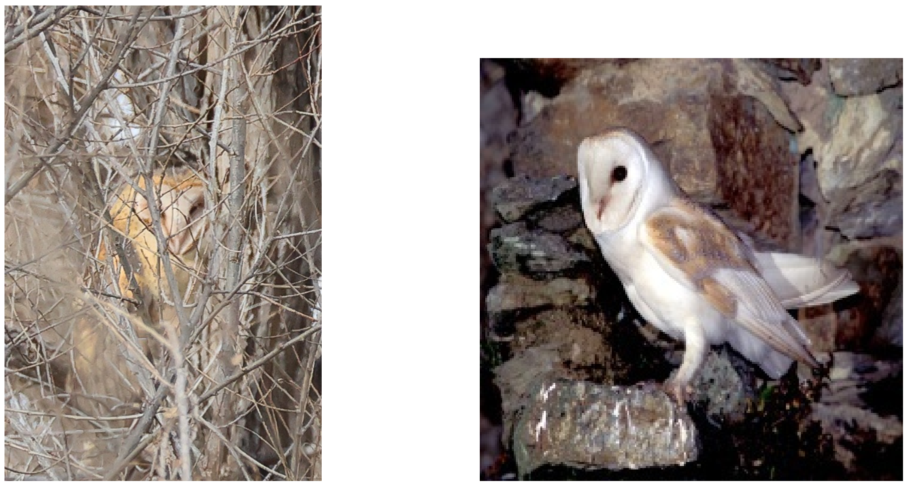
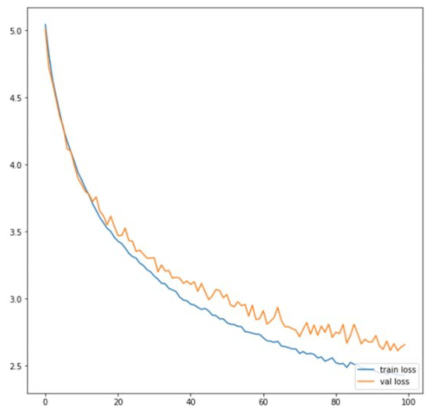


Figure 8: **(Left)** An example of an obscured bird from the scraped dataset **(Right)** An example of the same class of bird from the kaggle dataset

Part 2: Model Performance

To show the progression of our model loss and accuracy over epochs, we have included this section in the appendix to include the epoch-by-epoch results. Generally, we see steady positive improvement in accuracy accompanied by a steady decrease in the loss function for each model. In each graph, the orange represent the validation set's value and the blue represents the training set's value.



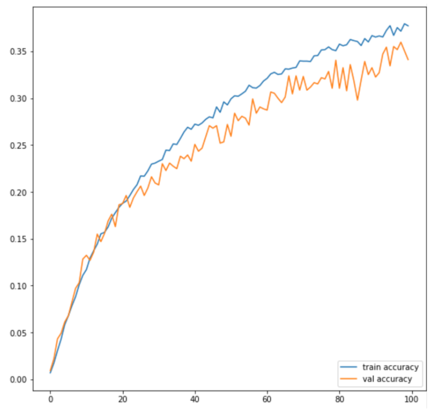
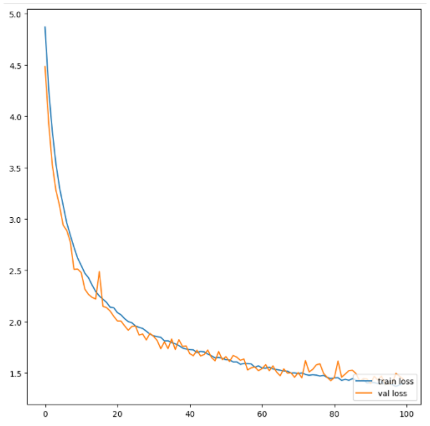


Figure 9: **(Left)** Progression of the full scraped loss function **(Right)** Progression of the full scraped accuracy



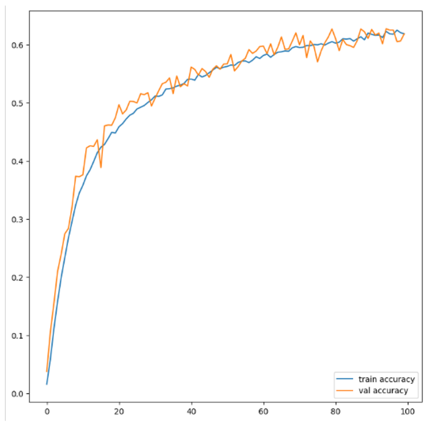
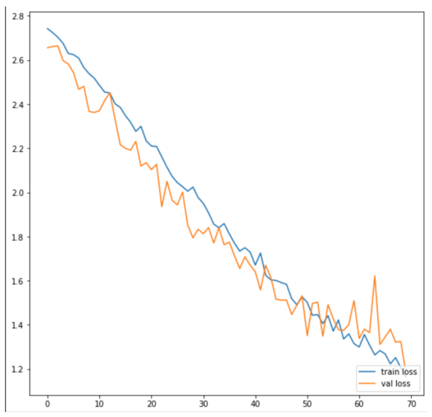


Figure 10: **(Left)** Progression of the full Kaggle loss function **(Right)** Progression of the full Kaggle accuracy



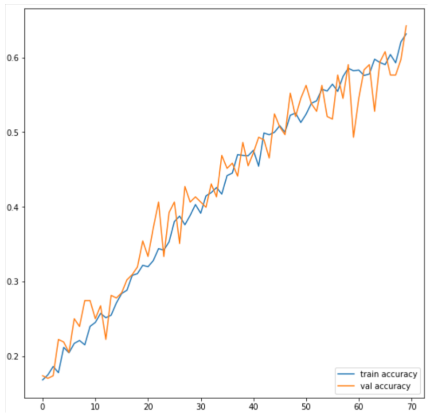
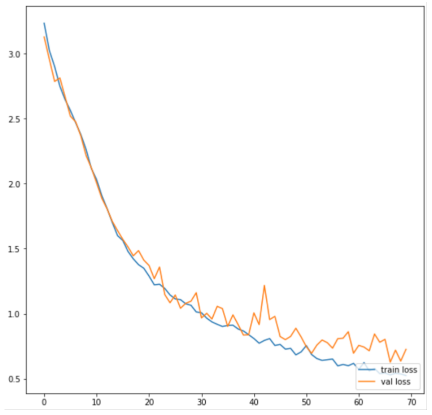


Figure 11: **(Left)** Progression of the partial scraped loss function **(Right)** Progression of the partial scraped accuracy



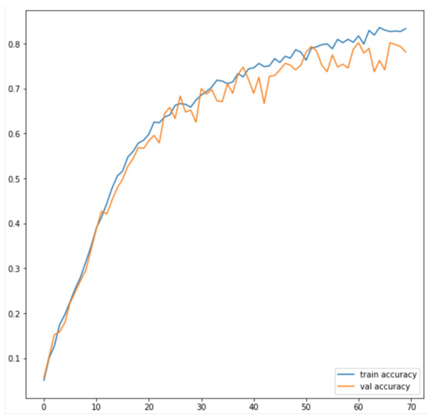


Figure 12: **(Left)** Progression of the partial Kaggle loss function **(Right)** Progression of the partial Kaggle accuracy