Automated Species Recognition for Remote Sensing of North Carolina Wildlife

Dylan Bates, A, A

December 16, 2019

Abstract

We use transfer learning to train the Oxford VGG16 model on images of 16 species of wildlife from the Netherlands and Panama. Despite significant flaws in the dataset, we are able to achieve greater than 79.9% accuracy on out-of-sample images by leveraging a priori temporal information from sequences of images, which approaches the theoretical maximum of 83.3% for this dataset.

1 Introduction

Wildlife camera-traps provide an effective way for researchers to track animal populations. However, in order to do so, individual images have to be labeled, which can be prohibitively expensive and time-consuming. In North Carolina alone, there are 3780 wildlife cameras[2], spread out over all 100 counties in the state. However, current classification techniques require volunteers to manually label thousands of images, resulting in several weeks of data collection downtime, while the current set of data is processed, labeled, and analyzed. This setup is common across the country[15], and around the world[11] on a far larger scale. In order to speed up the image labelling process, we use machine learning to classify species of animals captured on these cameras. Specifically, we retrain the top layers of VGG16 on 16 species of animals.

2 Data Set

We used the public Missouri Camera Traps dataset from the Labeled Information Library of Alexandria: Biology & Conservation. This dataset consists of approximately 25,000 labeled images of 20 species of animal. Some of the images were full colour, while some were black and white infrared images, taken at night. The images varied in size from 1920×1080 to 2048×1536 pixels, each with a 32 pixel border that contained metadata about when the image was taken.

The most unique feature of the dataset was that instead of single photographs, the pictures were presented as a sequence of between 3 and 305 images. The cameras took pictures when motion was detected, meaning sequences could be under a second or several minutes in length. Most sequences were 10 images in length, with each picture taken about a second apart.

This poses two major challenges to our model. First, the images are highly non-independent, as an animal may only move a few pixels before a new picture was taken. For many sequences, >99% of the image remains fundamentally the same from one image to the next, which could cause some issues with training. We took great care in ensuring that images from within the same sequence only ended up in one of the training, validation, or testing sets. Second, the cameras continued to snap pictures after the animal had left the frame. This means the last few images of each sequence frequently do not contain an animal at all, despite being labeled the same as all the other images in the sequence. This problem was more difficult to solve, and required a novel way of predicting sequences of images instead of single images.

3 Machine Learning Techniques

We originally planned on using Google's InceptionV3 model, a 42-layer deep convolutional neural network, trained on millions of ImageNet images. We replaced the top layer with a 1024 neuron fully connected layer with `ReLu` activation, and finally a 16 neuron output layer with `softmax` activation, representing probabilities each image contains one of the 16 species of animal. We froze the bottom layers of the model, and trained the top layer only, in order to use the powerful feature extraction InceptionV3 offers, with the top layer tailoring the weights to our specific images. Unfortunately, during testing, a fatal incompatibility between Keras and InceptionV3 arose: Batch Normalization in Keras is fundamentally broken[13], and InceptionV3 contains 19 Batch Normalization layers.

More specifically, Keras has a built-in mechanism called `learning_phase`, that controls whether the network is in train or test mode. The user is only able to change the learning phase to a specific value, before any model or tensor is added in the graph. When a Batch Normalization layer is frozen (as is the case during transfer learning) it keeps updating its batch statistics during training. As a result, when you fine-tune the top layers, their weights are adjusted to the mean/variance of the new dataset. However during testing, since the test data has a different mean/variance than the original dataset, even a fully trained model can result in a significant error rate. This is a well documented issue[3][4][5][6] with no official solution.

Fortunately, we encountered this error early on. While debugging, we overfit InceptionV3, by training it to 100% accuracy on the training data. However, while testing the model on the *exact same data*, the accuracy was only 16%. If the model is unable to even classify the data it is trained on, there's no way it could ever classify out of sample data, which explains why our out of sample

accuracy was ∼4% – worse than guessing.

Since any pre-trained model containing batch normalization layers would be affected by this, we switched to Oxford's VGG16. Compared to InceptionV3, VGG16 is hilariously simple and clunky. VGG16 is trained on the same ImageNet images, but is only 16 layers deep: 13 3×3 convolutional layers with maxpooling, followed by two 4096 neuron fully connected layers with `ReLu` activation, and a final 1000 neuron output layer with `softamx` activation[10]. As before, we started by replacing the top fully connected layers with a 1024 neuron fully connected layer with `ReLu` activation, and finally a 16 neuron output layer with `softmax` activation. We froze the bottom layers of the model, and trained the top layer only.

Despite it's simplicity, VGG16 contains 138 million parameters (compared to InceptionV3's 25 million), which requires a lot more computational power and memory while training. Though it did not perform quite as well as GoogLeNet (InceptionV1) in the ILSVRC 2014 competition[9], this is partially due to the advanced image processing and ensemble weighting Google performed. In an apples-to-apples comparison, with a single network looking at a single crop of each test image, VGG16 actually outperforms InceptionV1[1]. The lack of Batch Normalization layers means it lacks the significant advantages offered by Batch Normalization in InceptionV3 (increasing the learning rate, decreasing the need for Dropout, etc.)[7]. However, we can reduce VGG16's disadvantages by adding dropout, using a low learning rate, and normalizing each image during preprocessing, in order to get comparable results.

3.1 Preprocessing

The 25,000 images in the dataset ranged from 1920×1080 to 2048×1536 pixels, and each had a 32 pixel border at the top and bottom containing metadata identifying the image. The first thing we did was write a batch script to crop out that information, and resize the image to the appropriate resolution for the network (299×299 for InceptionV3 and 224×224 for VGG16). Since we were going to augment the images in training anyway, the full size cropped images ended up being the final ones.

Our original method involved splitting the 25,000 images with an (80%, 10%, 10%) training, validation, testing split, and training the only the top layer of InceptionV3. This approach yielded less than satisfactory results (∼4% test accuracy) despite having thousands of training images and one of the most powerful trained classification networks at our disposal. One glaring issue was that some of the images were full colour RGB images, while others were greyscale infrared images, taken at night. Both InceptionV3 and VGG16 expect a 3-channel image, so another batch script analysed the colour of each image and removed any nighttime photos. This left a total of only 528 sequences containing 7098 images. Notably, this removed all images of Paca, Common Opossum, and Wood Mouse, and all but one sequence containing a Spiny Rat, reducing our dataset to a very unbalanced collection of 16 species.

The next major challenge we faced was that most of the sequences still

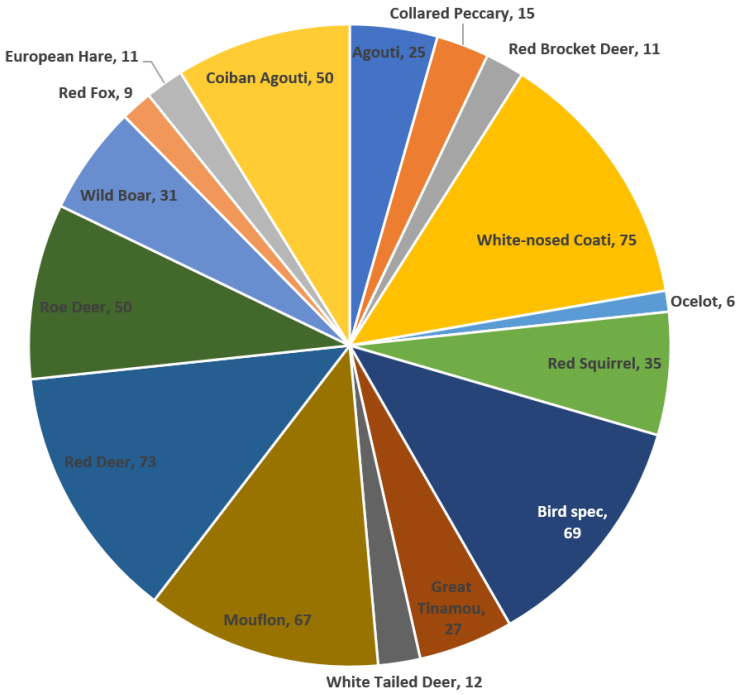

Figure 1: Number of images per species in the training set.

contained empty images, with no identifiable animal in frame. These would have to be removed, or else the network would learn the background of the images, and not the animals themselves. Since the sequences were extremely unbalanced anyway (ranging from 3 to 305 images), instead of continuing to delete images to balance the dataset (which would have left us with only 7 images per species), we decided to separate sequences into training and testing with a 90%/10% split, then clean *only the training set* in order to isolate the best images from each sequence to properly train the network. We manually went through all 7098 images and selected the single best image from each sequence to put into the training set. On several of the longer sequences (dozens to hundreds of images) two or more images were chosen if they contained multiple animals of the same species or the same animal in different orientations. Since the images were going to be compressed to as little as 1.6% their original size, preference was taken to have large animals close to the camera. Due to the image augmentation, preference was also given to animals close to centre frame, and unobscured by foliage. This left us with 566 "good" images in the training set (seen in Figure 1), and 61 sequences totalling 1057 images in the test set. The training set images were presented to the network as individually labeled, while the testing set images were presented as an entire sequence with the same label.

Once loaded into Python (hosted on Google Colab), the images were further processed while being loaded into the model. Each image was scaled using the default nearest neighbour scaling mode to the appropriate resolution (224×224 for VGG16), and then normalized so that each pixel value ranged from -1 to 1. During the later stages of training, images in the training set were flipped horizontally, and later rotated, reflected, sheared, scaled, and translated in order to help the model generalize. During validation and testing, images were only scaled to 224×224 and normalized from -1 to 1, with no other augmentation.

4 Classifying Sequences

Since the dataset was composed of labeled sequences instead of labeled images, up to 4/5 of the images in a sequence could have no visible animal at all. As discussed, it made no sense to train the network on these blank images, so they were removed from the training and validation sets. However, future sequences input to the model for classification would likely also contain empty images, so it was important to learn how to label them as well. 176 images in the testing set (16.7%) contained images with no visible animal, giving a theoretical maximum accuracy of 83.3% on a perfectly trained model.

To this end, knowing that images taken within a few seconds of each other almost certainly contained the same animal, we chose to predict the label for all of the images in a sequence, then use a variety of weighting and averaging functions to classify the entire sequence with a single label. We chose three different methods of weighting the output vectors, with varying degrees of aggressiveness and accuracy, in order to improve the test accuracy.

4.1 Basic Averaging

Computer vision accuracy is traditionally calculated as the number of correctly labeled images divided by the total number of images (hereafter called "naïve accuracy"). This would work poorly for us, since many sequences contain images with no visible animal, which would almost certainly be classified incorrectly.

The most basic approach to fix this is to average all the output vectors across each sequence, and predict the entire sequence contains the animal with the highest weight. This is a rather weak metric in comparison to the others, because all the weights really average out over sequences with hundreds of images, reducing the reported confidence. Additionally, if a sequence contained >50% empty or low quality images, the reported label was usually incorrect. Since real world images are often empty or low quality, we wrote a more aggressive algorithm to try and fix this.

4.2 Reverse Averaging

Instead of averaging output vectors over a whole sequence, producing more and more equivalent weights across each class, we turned the idea on its head by

measuring how much each output vector varied from the (expected) mean.

In a dataset of 16 classes, the expected weights for an image containing no animals are $\frac{e}{16} = [0.0625, 0.0625, \ldots, 0.0625]$. For every image in a sequence, we measure how far the individual probabilities vary from e/n, and multiply those ratios together. This serves to increase the confidence that a single animal is in the image if the probability of it being in the image is greater than 1/16. If the probability of the animal being in multiple images is greater than 1/16, these weights will compound, greatly increasing the confidence in the reported label. On the other hand, if the confidence of an animal being in an image is less than 1/16 in multiple images, it will quickly lower the confidence of that animal being the one reported, as the weights quickly converge to 0 after only a few images.

The algorithm is as follows:

1. Input the m images from a single sequence into the network, producing m output vectors v^1 to v^m, where each $v^i = [v_1, v_2, \ldots, v_n],\ 1 \leq i \leq m$.

2. Multiply each vector by the number of classes n. This is equivalent to dividing each weight by $1/n$, or comparing if it's more likely (> 1) or less likely (< 1) to be in the image compared to the expected value.

3. Multiply the m vectors elementwise, so that $w_j = \prod_{i=1}^{m} v_j^i,\ 1 \leq j \leq n$. For each class j, if there are many $v_j^i > 1$, then $w_j >> 1$. Similarly, if there are many $v_j^i < 1$, then $w_j \approx 0$.

4. Divide w by $\sum_{j=1}^{n} w_j$ to create our new probability vector. On a "nice" sequence, this will pretty much be a one-hot vector with the correct animal

The reason this works so well in theory is that it only takes a handful of images correctly classified with moderate confidence to correctly classify the entire sequence, because the other weights rapidly approach 0, if their value is less than the $1/n$ threshold.

In practice, this method of weighting can still fail if the sequences are particularly bad. For example, SEQ91281 contains 3 images of an ocelot taken over 3 seconds. While the ocelot is clearly visible, centre frame in the first image, it has left the frame of the image in the remaining two. Two out of three images in the sequence are blank, and are incorrectly classified. One architecture, F1024, trained to overfit the training set for 20 epochs (details below), classified the first image as an ocelot with 99.98% confidence, but classified the second and third images as a bird with 73.3% and 91.7% confidence. Using the algorithm above, both averaging and reverse averaging classified this sequence as containing a bird. For sequences like these, an even more extreme metric was needed.

4.3 Maximum Averaging

This shouldn't be called averaging at this point; it's not even averaging. Of the m images in sequence producing m different n-dimensional vectors, it determines

Figure 2: F1024 is 99.98% confident the left image is an ocelot, 71.3% confident the middle image is a bird, and 91.7% confident the right image is a bird.

the class of the single highest weight of the $m \times n$ weights given, and classifies the entire sequence as containing that animal.

In Figure 2 above, since the model is 99.98% confident that one of the images contains an ocelot, the entire sequence is classified as ocelot. This metric may be too aggressive, in the case that one image in a sequence is bad enough that the model reports a strong false positive, it can throw off an entire sequence. For example, SEQ81344 in Figure 3 below contains 105 images of the same animal that it predicts are mouflon with average 77.4% confidence, but a single image of a mouflon too close to the camera is predicted to be a red deer with 99.93% confidence, so all 105 images are classified as red deer. However, we rarely see this in practice, and this super simple algorithm ends up having comparable or better out of sample accuracy than my big-brain reverse averaging idea. Interestingly, the reverse averaging is 100% confident the sequence contains mouflon. Perhaps an ultimate algorithm weighted toward maximum averaging for smaller sequences and reverse averaging for longer sequences could be used.

Figure 3: F1024 is 99.3% confident the left image is mouflon, but 99.93% confident the right image is of red deer. Yes, these are two different images.

5 Results

We began by testing six different top layer architectures on top of vanilla VGG16. We used the same set for training and validation on each, and attempted to overfit the model to 100% accuracy. The goal was to determine whether each architecture was successfully able to learn the important features

of the training set, before seeing how well it generalized to out of sample data. Our test set consists of 166 sequences totaling 1057 images that the network has not seen before.

The nomenclature is as follows:

1. F1024: fully connected layer with 1024 neurons.
2. D0.2: Dropout layer with 0.2 (20%) of neurons frozen.
3. A1: Basic image augmentation; randomly horizontal reflection of the training images.
4. A2: Complex image augmentation; rotation, reflection, shear, scale, and horizontal/vertical translation
5. *: Architecture did not converge on training set. Validation loss failed to decrease, and training was stopped.
6. **Bold:** Model with the highest accuracy at each stage of testing.
7. *Italic:* Best accuracy metric for that particular model.

Model	**Epochs**	**Train**	**Val**	**Test**
F1024	20	1.0000	1.0000	0.6206
F1024	32	1.0000	1.0000	0.6301
F4096/F4096	13*	0.1378	0.1325	0.0766
F1024D02	32	1.0000	1.0000	0.6367
F1024/D0.5	32	0.9876	1.0000	**0.6509**
F4096D0.5/F4096D0.5	32	0.9647	0.9859	0.6225

Table 1: Train, Validation, and Testing Accuracies for different architectures.

Model	**Naïve**	**Average**	**Reverse**	**Max**
F1024-20	0.6206	0.6230	**0.7049**	***0.7213***
F1024-33	0.6301	0.6066	0.6393	*0.6885*
F4096/F4096	0.0766	*0.1475*	*0.1475*	*0.1475*
F1024D0.2	0.6367	0.6066	0.6066	*0.6557*
F1024D0.5	**0.6509**	0.6393	0.6721	*0.6557*
F4096D0.5/F4096D0.5	0.6225	**0.6557**	0.6557	*0.6885*

Table 2: Out of sample test accuracy, measured using 3 different custom metrics.

As seen in Table 1, the only architecture that failed to converge was with two sequential 4096 neuron fully connected layers. This architecture was no longer tested after this point. Since most of the architectures had converged within 32 epochs, that was the number chosen for all later testing. Each architecture was

trained using the Adam optimizer until the validation loss has failed to decrease for 12 epochs in a row, or 32 epochs; whichever was sooner. Next, we unfroze the last two convolution blocks, and trained for a few more epochs using SGD with a low learning rate. Finally, we calculated the out of sample test accuracy, using the three different metrics outlined in Section 4.

Table 2 presents the same architectures as Table 1, but also calculates the out of sample test accuracy using our custom accuracy metrics. Reverse Averaging and Max Averaging consistently outperform the naïve test accuracy, while Basic Averaging is sometimes worse and sometimes better.

Selecting the 4 successful architectures from Tables 1 and 2, we randomly partition the training set with an 80%/20% training/validation split. The test set remains the same 166 sequences of 1057 images from before.

Model	**Epochs**	**Train**	**Val**	**Test**
F1024	15	1.0000	0.5046	0.6291
F1024D0.2	18	0.9981	0.5321	0.6594
F1024D0.5	16	0.9674	0.5596	**0.6613**
F4096D0.5/F4096D0.5	25	0.9721	0.5596	0.6500

Table 3: Train, Validation, and Testing Accuracies with an 80/20 train/val split.

Model	**Naïve**	**Average**	**Reverse**	**Max**
F1024	0.6291	**0.6393**	**0.6721**	***0.7049***
F1024D0.2	0.6594	0.6066	0.6230	*0.6721*
F1024D0.5	***0.6613***	0.6230	0.6066	0.6557
F4096D0.5/F4096D0.5	0.6500	0.6066	0.6230	*0.6557*

Table 4: Out of sample accuracy metrics, with an 80/20 train/val split.

As seen in Table 3, the validation loss failed to decrease well before 32 epochs. It appears that dropout is helping the out of sample test accuracy, at the expense of the training accuracy, so seems to be helping with overfitting. That being said, when looking at the data in Table 4, it's unclear whether or not dropout is helping us at this stage. The Reverse Averaging accuracy, which has been very successful so far, is actually performing worse than naïve test accuracy, so we proceed with all 4 networks.

At this stage, we are still only training each network on a random 80% split of the training data - only 457 different images. While these images were selected to be among the most representative in the dataset, they hardly represent all the variation possible among images of the 16 different species. In an effort to make the network generalize a little better, we perform basic image augmentation by horizontally flipping a random subset of the training images during the preprocessing stage. This helps prevent the network from thinking that all deer only face left, for example.

Model	Epochs	Train	Val	Test
F1024	17	1.0000	0.5413	0.6140
F1024D0.2	18	0.9923	0.5046	0.6083
F1024D0.5	21	0.9846	0.4862	**0.6452**
F4096D0.5/F4096D0.5	18*	0.3391	0.3486	0.3955

Table 5: Train, Validation, and Testing Accuracies with basic augmentation.

Model	Naïve	Average	Reverse	Max
F1024	0.6140	0.5902	0.6230	*0.6885*
F1024D0.2	0.6083	0.5902	0.6066	*0.7049*
F1024D0.5	**0.6452**	**0.6557**	**0.6557**	***0.7213***
F4096D0.5/F4096D0.5	*0.3955*	0.3279	0.3115	0.3279

Table 6: Out of sample accuracy metrics, with basic augmentation.

As seen in Table 5, basic image augmentation did nothing to increase the out of sample test accuracy. Notably, for the three networks that converged, it increased the amount of time it took to train, which makes sense. That being said, when considering the the two 1024 neuron networks with dropout, the Reverse Average accuracy and especially the Max Average accuracy increased greatly, as seen in Table 6. Since image augmentation appears to be working, we perform more complex augmentation on each image, and try to train the remaining architectures again. This time, we randomly rotate, reflect, shear, scale, and translate the image, each up to a factor of 0.1. Every epoch, the network has been training on the same 457 images in the training set. This time, every time the network sees an image, it will be augmented differently. Instead of just seeing a wild boar, it will now get dozens of unique variants of wild boar, over dozens of epochs. Hopefully, the network will learn specifically what features make an image "boar-like," so that when it sees new images of a boar during testing, it will be better able to classify it.

Model	Epochs	Train	Val	Test
F1024	19	0.9185	0.4037	0.6263
F1024D0.2	21	0.8906	0.5229	0.6272
F1024D0.5	32	0.8464	0.4587	**0.6433**

Table 7: Train, Validation, and Testing Accuracies with complex augmentation.

As seen in Table 7, the number of epochs has again increased before validation loss failed to decrease. Interestingly, this is the first time these networks have struggled to entirely (or close to entirely) learn the training set, reflected in the lowest training accuracies yet. We are no longer overfitting the data! However, the training accuracy is still significantly higher than the testing accuracying. This seems to indicate that we could continue with more augmentation

Model	Naïve	Average	Reverse	Max
F1024	0.6263	**0.6393**	***0.6557***	0.6557
F1024D0.2	0.6272	0.6230	0.6230	*0.6557*
F1024D0.5	**0.6433**	0.6230	0.6393	***0.7049***

Table 8: Out of sample accuracy metrics, with complex augmentation.

to the images. Despite this, the out of sample test accuracies continue to rise, while our custom metrics stagnate or fall.

At this point, we look at the confusion matrix of the best performing model: a 1024 neuron fully connected layer, with 50% Dropout.

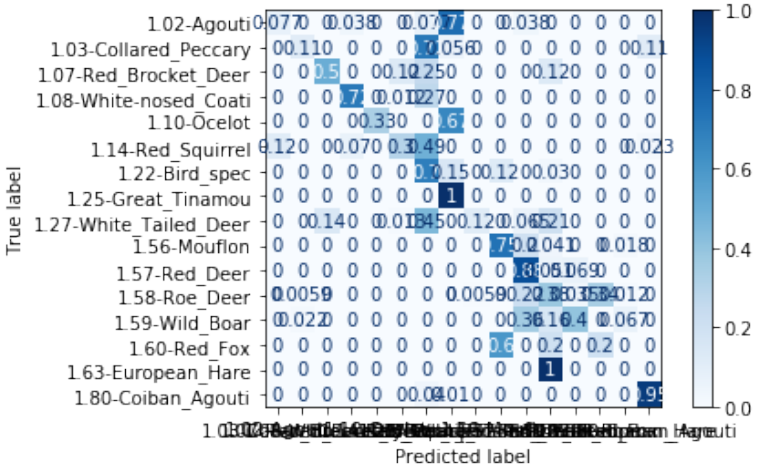

Figure 4: Barely visible confusion matrix of F1024D0.5

As poorly seen in Figure 4, the model was accurately able to classify every Great Tinamou, but thought that every European Hare was actually a Roe Deer. From Figure 5, it isn't difficult to see why. The test set contains only one sequence of European Hare, photographed in the same location as one of the training images of the Roe Deer. The model is still learning the backgrounds. Greater care would need to be taken to separate training and testing sets, or more advanced image augmentation used to reduce this in the future. Additionally, half of all species are frequently classified as birds. This is likely due to the fact that the birds in multiple training images are so small and camouflaged that the image appears to contain no animal at all. While classifying the 176 empty images in the training set, "bird" seems like a reasonable category, in this context.

Unsurprisingly, there is also a significant classificaiton error between the different species of deer. Possible solutions involve a second model specifically trained to classify species of deer, or offloading those images to experts for further analysis.

Figure 5: Test set European Hare (left) and training set Roe Deer (right).

5.1 Challenges

Through this entire process, it has felt like a struggle against the data to try and make it work with our program. Admittedly, we've come very far from that first 4% accuracy attempt, but at this point, it feels like we're hacking the results by defining our own metrics, much the same way Google claimed superiority over VGG in ILSVRC 2014 by using an ensemble of 7 networks each taking 144 crops of each image[12], then going back and modifying their network to include features present in VGG's model (specifically replacing 5×5 convolutional layers with stacked 3×3 convolutional layers).

The fact is undeniable though: the dataset is pretty trash. Animals are difficult to classify in general because of their inherent deformability; the same animal can look very different, depending on what it's doing. The backgrounds also posed a large problem. Forest images are very cluttered, making it difficult to identify exactly what thing in the photograph actually represnets the classification label[9]. The majority of the images were unusable because they were black and white. Of the coloured images, a surprising number of them contain no visible animal. Even when they do, it is often very difficult to identify them based on the one image alone. Additionally the images are enormous, but the animals in them are very small, and when compressed small enough to be input to VGG16, they are practically unrecognizable. Contestants in ILSVRC are able to get around this because the objects in the training images are large enough that when an image is cropped - sometimes hundreds of times, the object is still identifiable. If we have a small rodent in the corner of the frame, cropping the image at all - say in the centre and in all four corners, as is standard - would result in a four out of five crops not containing the rodent. This is exactly the thing we spent all that time avoiding in Section 3.1. As such, we were forced to rescale each image nonuniformly, changing its aspect ratio. This technique has been used on other image trap animal classification studies[8]. But this only introduces further distortions that make it more difficult to identify what animal is each image.

Even on many of the full size images, it can be difficult to figure out exactly

which type of agouti or deer is present. To a non-expert, a collared peccary might look exactly like a wild boar. And a Great Tinamou is a bird. Why is it in its own category? In fact, it seems very likely that some agouti, birds, and deer are mislabeled. Unsurprisingly, the network also struggles to classify the differences between these species. These errors can be poorly seen in the confusion matrix in Figure 4. To this end, we created one last dataset consisting of more broad categories: Agouti, Bird, Deer, and Pigs, in addition to the other more unique species: European Hare, Mouflon, Ocelot, Red Fox, Red Squirrel, and White-nosed Coati.

On this modified dataset, the exact same images are used (including the empty ones), but now they belong to fewer classes. Unsurprisingly, the out of sample accuracy rises as a result. We report the out of sample sequence accuracy for our best model in Table 9, and also convert back to a per-image instead of per-sequence accuracy.

Model	**Species**	**Test**	**Best Sequence Acc**	**Best Image Acc**
F1024D0.5	16	**0.6433**	**0.7049** (Max)	***0.7994*** *(Max)*
F1024D0.5	10	**0.7408**	**0.7869** (Max)	***0.8354*** *(Avg)*

Table 9: Testing Accuracies: best sequence accuracy and best images accuracy.

As seen in Table 9, the F1024D0.5 network consistently gave us the best results across both the original, and modified datasets. The confusion matrix in Figure 6 shows that it still occasionally classifies 7 different species as birds, still makes mistakes on small sequences like European Hare and Ocelot, but the custom accuracy metrics can help increase the accuracy. 176 out of 1057 test images contained no visible animal, meaning this dataset has a theoretical maximum accuracy of 83.3% from a perfect network without guessing, or just under 90% when randomly guessing the classification of empty images.

6 Conclusion

Working with some terrible images, we were still able to train a convolutional neural network to correctly classify 80% of out of sample test images by training the network on a collection of "nice" images, then exploiting temporal information about the images (all images in a sequence have the same label) in order to test the network on "bad" images. This is completely reasonable, because future data will come in the form of sequences as well, so the same 80% accuracy should be expected on future similarly sized, full-colour images.

Compared to existing systems, our convolutional neural net approach actually performs comparably to state of the art non-CNN methods. Yu et al. curated this dataset in 2013. In their paper[14], they use sparse coding spatial pyramid matching (ScSPM), which extracts dense scale-invariant feature transform (SIFT) descriptor and cell-structured local binary patterns (cLBP) as the local features, that generates global features via weighted sparse coding and max

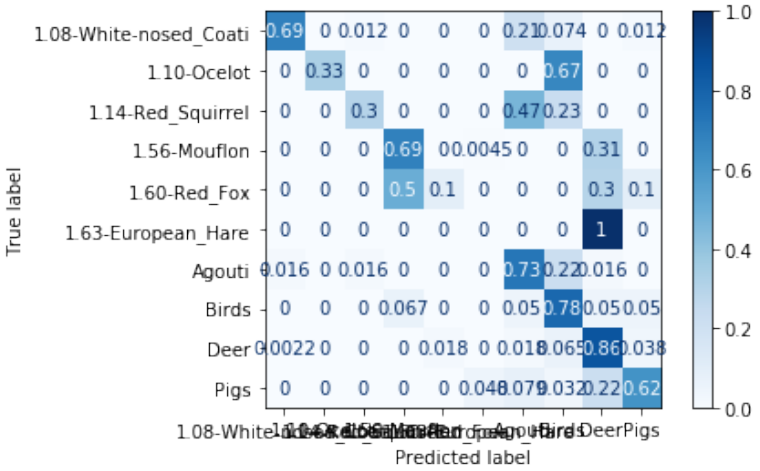

Figure 6: Confusion Matrix with 10 species. It still thinks bunnies are deer.

pooling using multi-scale pyramid kernel, and classifies the images by a linear support vector machine algorithm. Notably, they have to manually crop each image around the animal (if visible), and were able to achieve 82% accuracy on 7196 images from 18 species in this dataset.

Although we fell a hair short of that standard (and chose not to attempt to classify nighttime images at all), our method does not require any manual cropping, which is arguably even more time consuming than manual image identification. Were we testing the network on carefully chosen and cropped squares around each animal, our accuracy would rise as well.

References

[1] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.

[2] Candid Critters. North Carolina candid critters. `https://www.nccandidcritters.org`.

[3] GitHub. fit_generator get much lower accuracy then the evaluate_generator just one line above? `https://github.com/keras-team/keras/issues/6895`, 2017.

[4] GitHub. model.evaluate() gives a different loss on training data from the one in training process. `https://github.com/keras-team/keras/issues/6977`, 2017.

[5] GitHub. Setting learning_phase to 0 leads to extremely low accuracy. `https://github.com/keras-team/keras/issues/7177`, 2017.

[6] GitHub. Change bn layer to use moving mean/var if frozen. `https://github.com/keras-team/keras/pull/9965`, 2018.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[8] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S. Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences*, 115(25):E5716–E5725, 2018.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[11] Alexandra Swanson, Margaret Kosmala, Chris Lintott, Robert Simpson, Arfon Smith, and Craig Packer. Snapshot serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. *Scientific Data*, 2:150026, 06 2015.

[12] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[13] Vasilis Vryniotis. The batch normalization layer of keras is broken. http://blog.datumbox.com/the-batch-normalization-layer-of-keras-is-broken/, Apr 2018.

[14] Xiaoyuan Yu, Wang Jiangping, Roland Kays, Patrick Jansen, Tianjiang Wang, and Thomas Huang. Automated identification of animal species in camera trap images. *EURASIP Journal on Image and Video Processing*, 1, 09 2013.

[15] Z. Zhang, Z. He, G. Cao, and W. Cao. Animal detection from highly cluttered natural scenes using spatiotemporal object region proposals and patch verification. *IEEE Transactions on Multimedia*, 18(10):2079–2092, Oct 2016.