Bird Identification Using the iNaturalist Dataset

Springboard | Capstone 2: Final

By: Lauren Broussard

Summary

- This project uses deep learning to classify images of birds. The photos are of 6 types of wildlife - Plants, Birds, Reptiles, Amphibians, Fungi, and Insects - primarily taken in their natural habitats. Using images taken in the wild can help create more realistic conditions for potential client use.

- Data Wrangling and Exploratory Data Analysis (EDA) were done primarily using the pandas and cv2 libraries in Python. The original dataset came from iNaturalist's 2019 Kaggle competition. Of the data provided, 15,000 total images were used - 50% were of birds, and the other 50% were not birds.

- A model was trained using deep learning and convolutional neural networks to classify images, resulting in an accuracy score of 87%. Corresponding code for the project and other write ups (including slides) can be found on Github.

- Successful image classification of birds can be helpful for clients like bird watchers, wildlife researchers, or social media managers to speed up their tasks.

- Future recommendations for this project include increasing processing capability, training on more images, and classifying individual birds by species.

Business Problem

Can we use image classification to identify birds? iNaturalist, a site that allows users to add images of wildlife they observe, hosts an annual Kaggle competition that encourages participants to help improve image classification for their repository. Successful image classification in the natural world has many benefits -- for instance, it can help conservationists and zoos to identify birds in the wild without disturbing their habitat.

Sample images from iNaturalist image files

This project uses deep learning, and specifically convolutional neural networks (CNN) to read images from 1010 different species and 6 different wildlife types. While the original Kaggle competition sought to classify one species from another - for instance, a picture of a seagull versus a picture of an eagle, this project seeks to look at images from a more high level to classify a specific type wildlife type -- birds. **We are interested in whether or not we can train a model to detect whether or not there is a bird in the photo.** The ability to do this would have implications for a number of potential clients.

Potential Client(s):

> *Wildlife Researchers or Zoos:* Being able to classify birds may help to track migratory patterns, or to identify when there are birds in areas where they would not be expected.
>
> *Bird Watchers/Enthusiasts:* Someone looking around for birds in the sky, or in trees might be able to use this kind of technology to quickly scan natural habitats to find whether or not there is a bird nearby, which could result in time savings.
>
> *Other Clients*: For social media managers, successful classification of birds can help with quick tagging of their posts, or for sites to more quickly label an image for a screen reader for additional accessibility.

Data Collection and Wrangling

Dataset and Images

Data and images were downloaded from the competition page on the Kaggle website. According to the iNaturalist overview on Kaggle, the images were collected and validated by multiple users from iNaturalist. The images were part of the Fine-Grained Visualization Categorization workshop, FGVC6, which included fewer species and images that may be harder than past competitions to classify - i.e. an image of a green frog against other green leaves.

The original dataset included **268,243 images with annotations,** including the following files, as described by the documentation:

File descriptions

- **train_val2019.tar.gz** - Contains the training and validation images in a directory structure following {iconic category name}/{category name}/{image id}.jpg .
- **train2019.json** - Contains the training annotations.
- **val2019.json** - Contains the validation annotations.
- **test2019.tar.gz** - Contains a single directory of test images.
- **test2019.json** - Contains test image information.
- **kaggle_sample_submission.csv** - A sample submission file in the correct format.

Source: Kaggle iNaturalist 2019 FGVC6 Competition/Data

The following files were used to approach the classification problem for this project:

- **train2019.json, val2019.json (57MB, 816.46KB)**: includes annotation information about each of the images, including the path to the image file, image height and width information, and encoded species information.

- **train_val2019 (79GB)**: contained in a larger train_val2019.tar.gz file, this folder includes all of the wildlife images, stored as jpeg files and separated in folders by high level category (i.e. Plants, Amphibians, Birds, etc.).

Data Wrangling

Data was imported and prepared in Python primarily using pandas. Though not always the case, this data came in largely standardized, so did not require as much cleaning as in previous projects. The following steps were taken to import and prepare the data for modeling:

*Loading Data***.** Annotations from both the *train2019.json* and *val2020.json* files had the following keys: 'info', 'images', 'licenses', 'annotations', 'categories'. For each of the two files, two DataFrames were created using data in 'images' and 'annotations'. The rest of the keys in the json file were ignored. For the *images* dataframe, the following fields were included: file name, image height, image width, as well as the image id. The *annotations* DataFrame included: image id and category id, the latter of which corresponded to a code denoting the species of the image. The dataframes were then merged, to create one main file of all data.

*Adding/Removing Columns***.** To be more descriptive, the column 'category_id' was changed to 'species_id'. Additionally, since we were interested in labeling Birds, we pulled the 'iconic category' name from the subfolders in the 'file_name' field. This created field was named "wildlife_type."

```
df.head()
```

image_id	file_name	height	width	species_id	wildlife_type
0	train_val2019/Plants/400/d1322d13ccd856eb4236c...	800	600	400	Plants
1	train_val2019/Plants/570/15edbc1e2ef000d8ace48...	533	800	570	Plants
2	train_val2019/Reptiles/167/c87a32e8927cbf4f06d...	600	800	167	Reptiles
3	train_val2019/Birds/254/9fcdd1d37e96d8fd94dfdc...	533	800	254	Birds
4	train_val2019/Plants/739/ffa06f951e99de9d220ae...	600	800	739	Plants

Missing Data or Duplicates. There were no duplicated rows, or missing data in the annotation files.

Numerical Encoding. Two numerical columns were created - one binary column called 'is_bird,' that gives a 1 if the 'wildlife_type' is Birds, and 0 if it is not. The other column, type_num, gave a numerical value to each wildlife_type.

image_id	file_name	height	width	species_id	wildlife_type	type_num	is_bird
0	train_val2019/Plants/400/d1322d13ccd856eb4236c...	800	600	400	Plants	5	0
1	train_val2019/Plants/570/15edbc1e2ef000d8ace48...	533	800	570	Plants	5	0
2	train_val2019/Reptiles/167/c87a32e8927cbf4f06d...	600	800	167	Reptiles	6	0
3	train_val2019/Birds/254/9fcdd1d37e96d8fd94dfdc...	533	800	254	Birds	2	1
4	train_val2019/Plants/739/ffa06f951e99de9d220ae...	600	800	739	Plants	5	0

Exploratory Data Analysis

With the data prepared, we then looked at features of the dataset and images. Overall, there were 265,213 images in the dataset, 1010 different species, and 6 different wildlife types.

Wildlife Types & Species

The vast majority of images in the dataset (nearly 160,000) are of Plants, followed by Birds, Insects, then Reptiles and Amphibians, and finally Fungi. **The dataset included 47,867 images of birds in 126 different species, accounting for only 18% of all images.** This class imbalance was remedied later in the project by taking equal samples of images of birds vs. images not of birds.

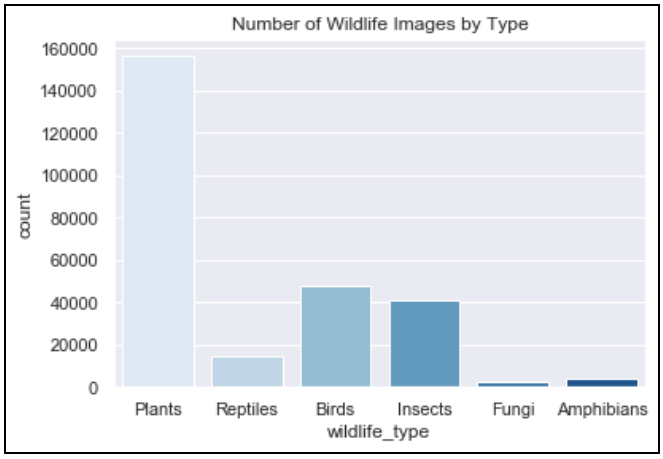
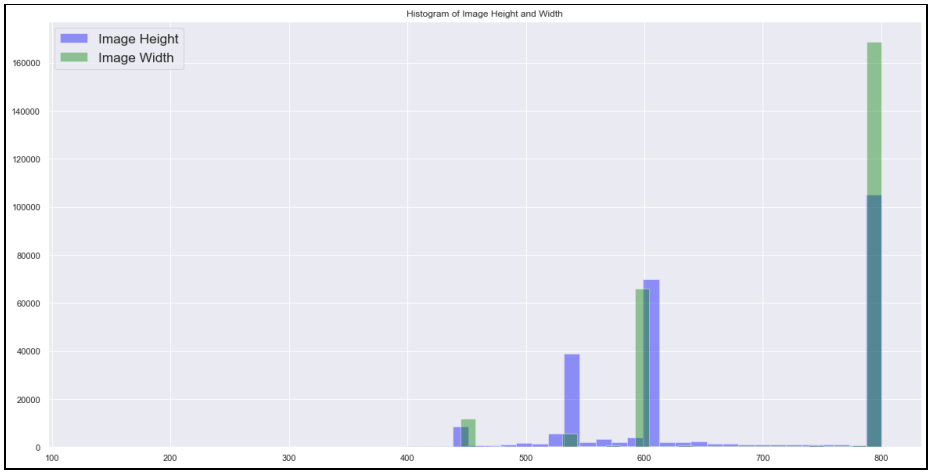


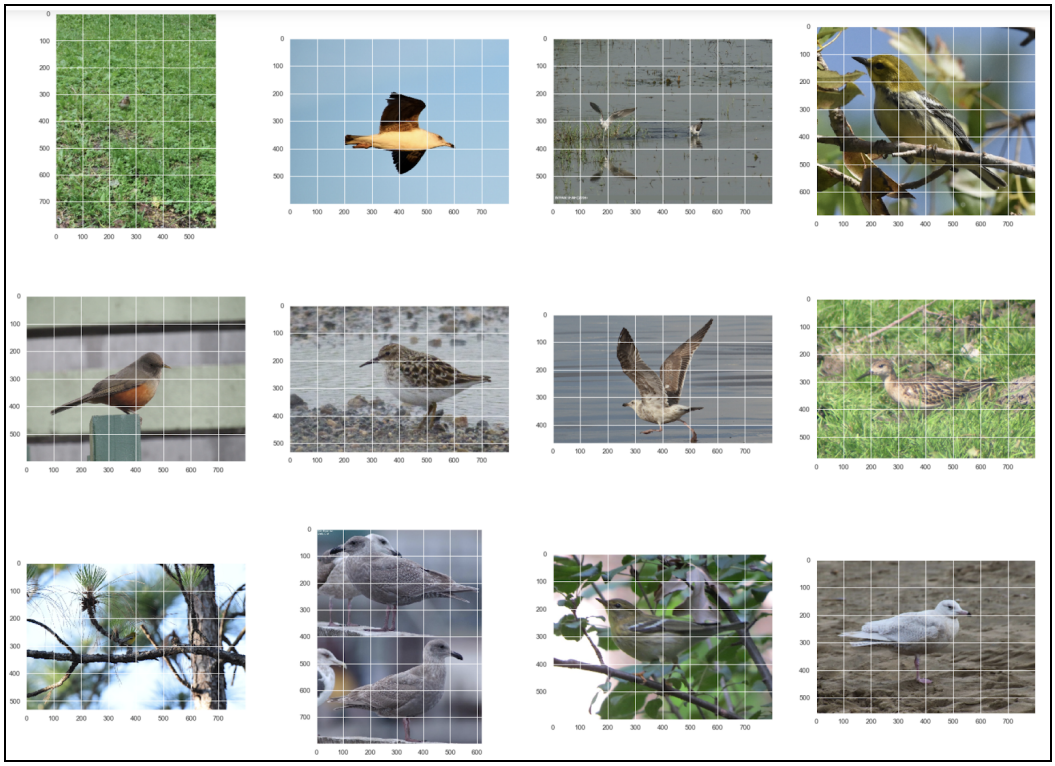
Image Height and Width

Each image had a maximum height and width of 800 pixels. The average height was approximately 663 pixels, while the average width was 720. The minimum height and width was 133 and 188, respectively. Images in the dataset tended to be wider (horizontal) than they were tall (vertical).



Sample Images (Birds)

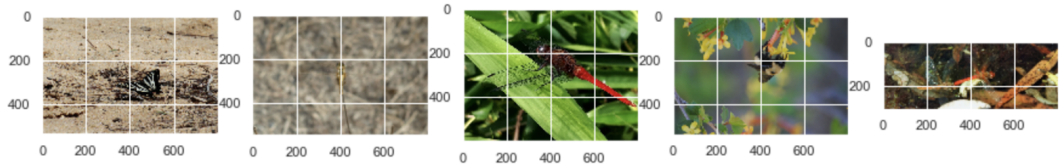
Below are sample images of birds from the dataset. They appear in various environments -- in grass, in the air, in trees -- and in different positions - mid-flight or perched. It will be important to be able to train a model that can learn the appropriate features of a bird, even when the image is slightly obscured, or when the bird is not the only thing in the image.



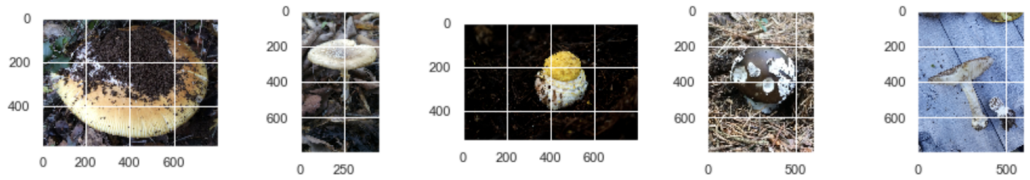
Sample Images (All Categories)

Below are sample images showing five images of each of the six higher level wildlife categories. Again, many images are obscured in some way by other things, for instance, the amphibians are in water or are partly camouflaged in plants (another one of the categories). In other cases, an animal is being held in a person's hand.

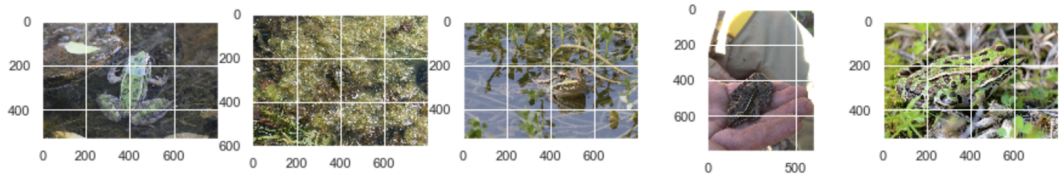
Insects



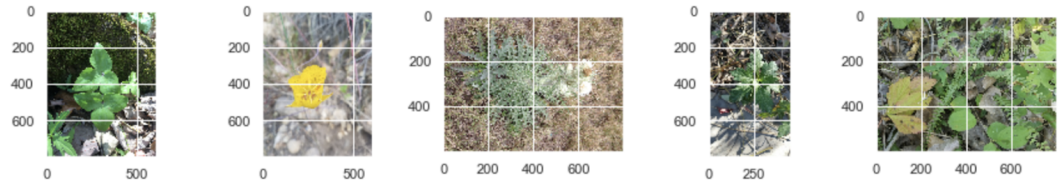
Fungi



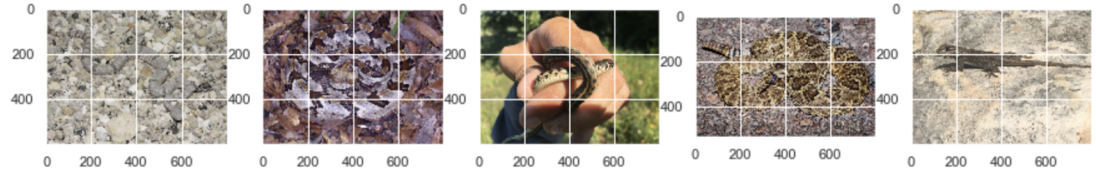
Amphibians



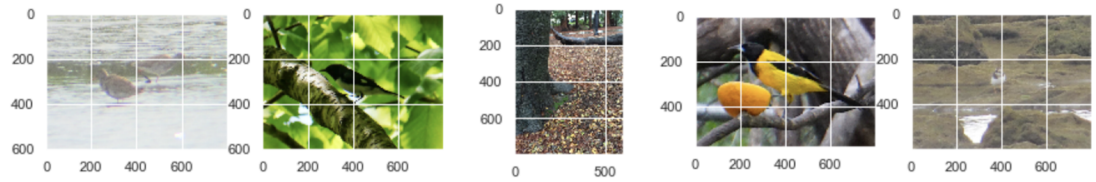
Plants



Reptiles



Birds



In-Depth Analysis & Modeling: Deep Learning

About Neural Networks

This project uses deep learning, specifically neural networks for image classification. Neural networks take a series of input nodes and associated weights to make further connections and ultimately make decisions. They are useful for a number of deep learning problems, specifically image classification, which requires the model to understand the connections between pixel data. The first few layers in the network look at simple features like the orientation of lines textures of an image, while deeper layers look at more complex features.

Pre-Processing

Since the computing power on my personal laptop was limited, I ran the project end to end with a smaller amount of the image data to create an initial model, and gradually increased the number of images over time. Further to this end, I pulled a random subset of images to train, validate, and test with. **Finally, to balance out the dataset, an equal but random 50/50 split was chosen, so that half of the images showed birds, and the other half showed a different wildlife type.** All pre-processing, set-up, and modeling was done using the keras library in Python.

Set-Up

Prior to running the models, constants were set, including a constant resized height and width of images, a batch size of 128, and an initial number of epochs (or trials) of 25, and an Early Stopping patience level of 10. The image below shows the values used in the final model.

```
img_dir =  ../data/raw/
img_width, img_height =  128 128
batch_size =  128
no_epochs =  50
no_classes =  2
patience =  10
```

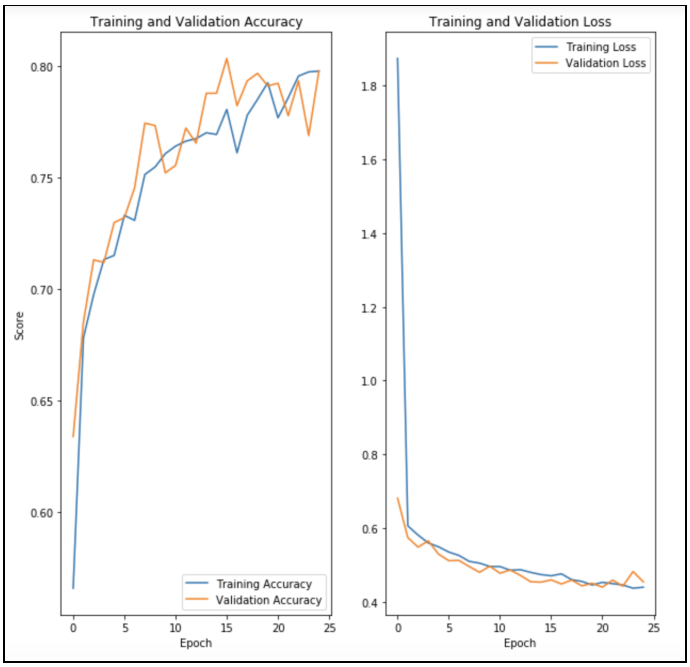
Augment Images

Random augmentation was done for the images in our training and validation datasets, including rescaling, zooming, shifting, and standardizing the orientation of images. Such augmentation can help prevent overfitting.

```
train_datagen = ImageDataGenerator(
                    rescale=1./255,
                    horizontal_flip = True,
                    zoom_range = 0.3,
                    width_shift_range = 0.15,
                    height_shift_range=0.15)
```

Initial Model - Fully Connected Network

To get a baseline of performance, a simple neural network was run on 5,000 images, with 25 Epochs, and an Early Stopping Patience of 2. It was a fully connected network using Dense layers -- meaning every layer (in our case, the pixels) is connected to all the pixels in the previous layer. This included five hidden layers and a flatten layer. The first layer included 32 nodes, all hidden layers used ReLU activation (which applies a weight of 0 or 1 depending on the value of the input node), and the output layer used sigmoid activation (which is appropriate for a binary classification). The initial iteration used a softmax activation in the final layer, but this created static predictions (approximately 50% accuracy). Switching to a sigmoid activation, the accuracy for both training and validation data increased to nearly 80% over 25 epochs. The resulting training and validation graph is depicted on the right.



Training/Validation Accuracy & Loss: Model 1, sigmoid activation

```
model1 = Sequential()

model1.add(Dense(32,activation = 'relu', input_shape = (img_width,img_height,3)))
model1.add(Dense(32, activation = 'relu'))
model1.add(Dense(32, activation = 'relu'))

model1.add(Flatten())
model1.add(Dense(32, activation = 'relu'))
model1.add(Dense(100, activation = 'relu'))
model1.add(Dense(1, activation = 'sigmoid'))
```

Simple Neural Network Architecture: Model 1

While the model seemed to do well in training and testing, with a validation accuracy score of nearly 80%, it did not do as well when generalizing to making predictions - the model performs about as well as chance, with an accuracy score of 50%.

Classification Report

	precision	recall	f1-score	support
Not Bird	0.48	0.54	0.51	585
Bird	0.51	0.45	0.48	615
accuracy			0.49	1200
macro avg	0.50	0.50	0.49	1200
weighted avg	0.50	0.49	0.49	1200

Classification Report: Model 1

Subsequent Model - Convolutional Neural Network (CNN)

The second model used a series of convolutional neural networks. It also included the following additional layers in the architecture:

- **Max Pooling:** The max pooling layer helps when there are a large number of parameters, as was the case in the previous model. It summarizes groups of pixels based on their max value.
- **Dropout Layer:** The dropout layer randomly sets inputs to 0 at a rate of 0.6 at each step during training, which can also help prevent overfitting.

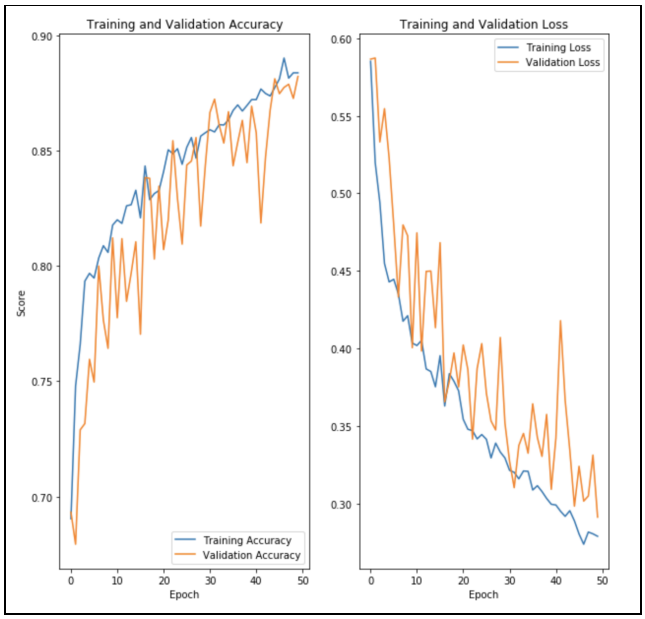
The model also used ReLu activation again in every layer except for the final, which used Sigmoid activation. The architecture of the model, along with the number of nodes at each layer, is shown in the image to the right.

```
model2 = Sequential()

model2.add(Conv2D(32, (3, 3), input_shape=(img_width,img_height,3))) #convolution layer
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2))) #

model2.add(Conv2D(32, (3, 3))) #receives input from previous conv layer
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(32, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(64, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())
model2.add(Dense(100))
model2.add(Activation('relu'))
model2.add(Dropout(0.6)) # what proportion to drop out
model2.add(Dense(1))
model2.add(Activation('sigmoid'))
```

Convolutional Neural Network Architecture: Model 2

Key changes between models:

- ***Total # of Images:*** A total of *15,000 images were used* -- 8,400 for training, 3000 for validation, and 3,600 for testing. The addition of images helps the model better learn what an image of a bird looks like, as it has more examples to learn from.
- **Increased Epochs:** The number of epochs increased from 25 to 50 for longer training.
- **Callbacks:** The early stopping monitor callback in the initial model was increased from 2 to 10, to allow the model to continue learning for more training.
- **Batch Size in Prediction:** For the final prediction, the batch size for the test generator was decreased from 128 (which was used for training) down to 1, so that the model saw only one image at a time to make a prediction on.
- **Total Parameters**: Finally, the number of parameters decreased from 16,782,889 to 268,489.



Training and Validation Accuracy & Loss: Model 2

This model achieved its highest validation accuracy at 88%. The training seemed to increase the accuracy at a fairly steady pace, though the validation accuracy seemed to bounce around for the second half of the epochs between about 80% and 88%.

Prediction

The following confusion matrix and classification report gives results for predictions using the second model, which achieved an overall accuracy of 87%. All values - precision, recall, and F-1 score - are above 80%. For instance, the ability of a model to classify a bird from all of the potential bird images the model was presented with is 86% (recall).

Classification Report

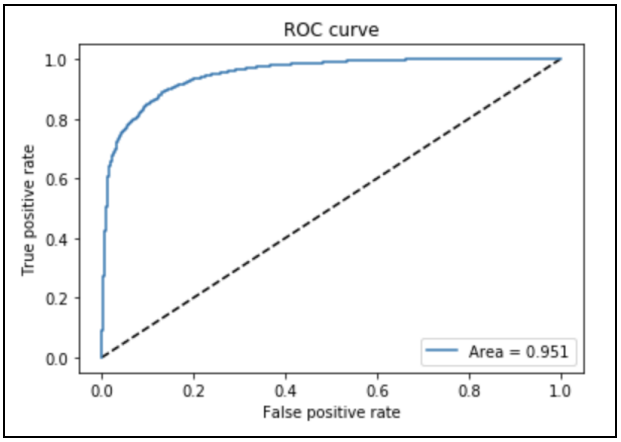
	precision	recall	f1-score	support
Not Bird	0.87	0.88	0.88	1830
Bird	0.88	0.86	0.87	1770
accuracy			0.87	3600
macro avg	0.87	0.87	0.87	3600
weighted avg	0.87	0.87	0.87	3600

Classification Report : Model 2

Results

The predictive power of the new model seems to have improved with the updated parameters and additional training time/images. The resulting ROC curve is shown to the right.

The area under the curve, 99%, shows us how well the model did at predicting. This is especially encouraging, as the model was able to cut through the noise around the images - i.e. it was able to classify well even when the bird was in various positions or backgrounds.



ROC Curve: Model 2; AUC = 0.951

Sample Images -- Incorrect Classifications. The model incorrectly classified 13% of the test images presented to it. When the model made an error in classification, it was more often that the model did ***not*** find a bird, when there was in fact one there.

Misidentified as Bird. Let's look at two scenarios. The first -- when the predicted probability of a bird being pictured was high (i.e. greater than 80%), although there was no bird in the photo.

ACTUAL: NOT BIRD (AMPHIBIAN)

Probability of Bird: 0.999685

ACTUAL: NOT BIRD (INSECT)

Probability of Bird: 0.990383

ACTUAL: NOT BIRD (PLANT)

Probability of Bird: 0.997390

Failed to ID as Bird. The second scenario of incorrectly classified images were when the model's predicted probability of a bird being pictured was low (i.e. less than 20%), although there was, in fact, a bird in the image.

PREDICTED: NOT BIRD

Probability of Bird: 0.075025

PREDICTED: NOT BIRD

Probability of Bird: 0.135277

PREDICTED: NOT BIRD

Probability of Bird: 0.098906

Sample Images -- Correct Classifications. The model did well at classifying birds on the whole, especially since many of the birds are in their natural habitats and the photo could be obscured. This is promising for automatic classification where it is necessary to not disturb an animal's habitat.

BIRD

Probability of Bird: 0.999685

NOT BIRD

Probability of Bird: 0.111596

BIRD

Probability of Bird: 0.931944

Recommendations & Further Considerations

These results are encouraging for researchers who would need to see/classify animals in their natural habitats without disturbing those habitats. These results are also good for bird watchers to be able to quickly tell whether or not there is a bird in their vicinity. Finally for a social media manager, tagging their images can be made much faster.

Some recommendations and caveats to consider for clients as well as for future iterations of the project:

- **Use Model to Begin Classification:** Since the model predicts much better than chance, it would be good to use as a starting point in classifying birds. For instance, if the model suggested to a bird watcher that there was a bird nearby, the bird watcher would have a starting point of where to look. Similarly, for a social media manager, the model can be used to suggest tags, and they may simply need to spot check instead of typing each bird tag.

- **Increase Number of Images for Training:** Due to processing power on my computer, I was not able to use the full dataset for the project. It would be important in future iterations to increase the processing power available, or in order to handle the increase in images.

- **Classify Further by Bird Species:** The next logical step in this project would be to look to classify birds by individual species. In doing so, it would further allow researchers to look at migration patterns of certain species.

- **Ensemble Methods:** Future iterations of the project could also look to other pre-trained models, such as VGG16, which is a model trained on the ImageNet database.

- **Pair Classification with Geolocation Data:** Paired with geolocation data, image classification of birds can also display where certain species are typically found in the world in order to help chart migration patterns or the existence of invasive species.

Resources

DataCamp - Image Processing in Keras: https://campus.datacamp.com/courses/image-processing-with-keras-in-python

iNaturalist: https://www.inaturalist.org/

iNat2019 Starter Keras Code, Kaggle: https://www.kaggle.com/ateplyuk/inat2019-starter-keras-efficientnet

Kaggle Competition Information: https://www.kaggle.com/c/inaturalist-2019-fgvc6/data

Keras.io Blog: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

LA County Threatened and Endangered Species:
https://data.lacounty.gov/Sustainability/LA-County-Threatened-and-Endangered-Species-2018-/7uw4-g37f