Bird Identification using the iNaturalist Dataset

Springboard | Capstone 2 Milestone Report #2

By: Lauren Broussard

Summary:

- This project uses deep learning to classify images of birds. The photos are taken of 6 types of wildlife (Plants, Birds, Reptiles, Amphibians, Fungi, Insects), primarily in their natural habitats - in grass, in the air, in trees, etc - which creates more realistic conditions for potential clients.

- Successful image classification can be helpful for clients like wildlife researchers, who are looking at migratory patterns of birds.

- 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% of birds, and 50% not of birds.

- I was able to train a model using deep learning and convolutional neural networks in keras to classify images with an accuracy score of 87%. Corresponding code for the project and other write ups (including slides) can be found on Github.

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

Business Problem

There are millions of different types of wildlife in the world. Can we use image classification to distinguish some of them from others - namely, 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 of images. Successful image classification in the natural world has many benefits -- for instance, it can help conservationists and zoos to identify species in the wild without disturbing their habitat, or can be used to identify potential endangered species that are being traded. Paired with geolocation data, image classification of wildlife 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.

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 pigeon 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: Being able to classify birds may help to track migratory patterns, or to identify when there are birds in areas they would not be expected to be.

Zoos, Wildlife Conservatories: It would be useful to be able to quickly classify images of the birds they work with and/or come across. It might also be useful for wildlife protection agencies who are scanning online databases for a particular type of bird that might be being sold, kept, or traded.

Other Clients: It could be useful for social media managers, who can use the image classification for 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 iNaturalist competition page on the Kaggle website. According to the iNaturalist overview on Kaggle, the images in this case were collected and validated by multiple users from iNaturalist. The images were part of the Fine-Grained Visualization Categorization workshop, FGVC6, and included fewer species that may be harder 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 competition 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

For the purposes of this project, all data was initially downloaded, but the following files were primarily used to approach the classification problem:

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

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

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, which corresponded to a code denoting the species of the image. The dataframes were then merged, to create one master 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 was named "wildlife_num" which mapped a numerical value for each wildlife type as follows: Amphibians: 1, Birds: 2, Fungi: 3, Insects: 4, Plants: 5, Reptiles: 6.

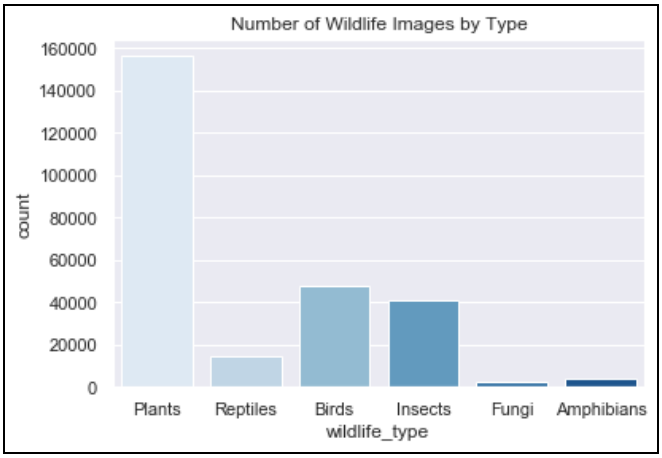
image_id	file_name	height	width	species_id	wildlife_type	type_num	is_bird
0	train_val2019/Plants/400/d1322d13ccd856eb4236c...	800	600	400	Plants	5	0
1	train_val2019/Plants/570/15edbc1e2ef000d8ace48...	533	800	570	Plants	5	0
2	train_val2019/Reptiles/167/c87a32e8927cbf4f06d...	600	800	167	Reptiles	6	0
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 are 265,213 images in the dataset, 1010 different species, and 6 different wildlife types.

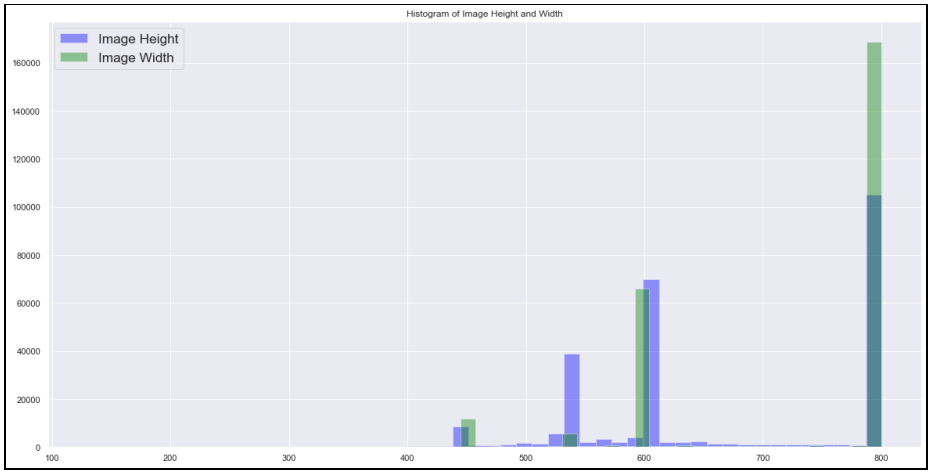
Wildlife Types & Species

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 column species_id is a value denoting the species of the image. **The dataset included 47,867 images of birds in 126 different species.** There is a class imbalance in our dataset - with birds representing only 18% of the dataset, which we will remedy by taking equal samples of birds vs not birds later in the project.



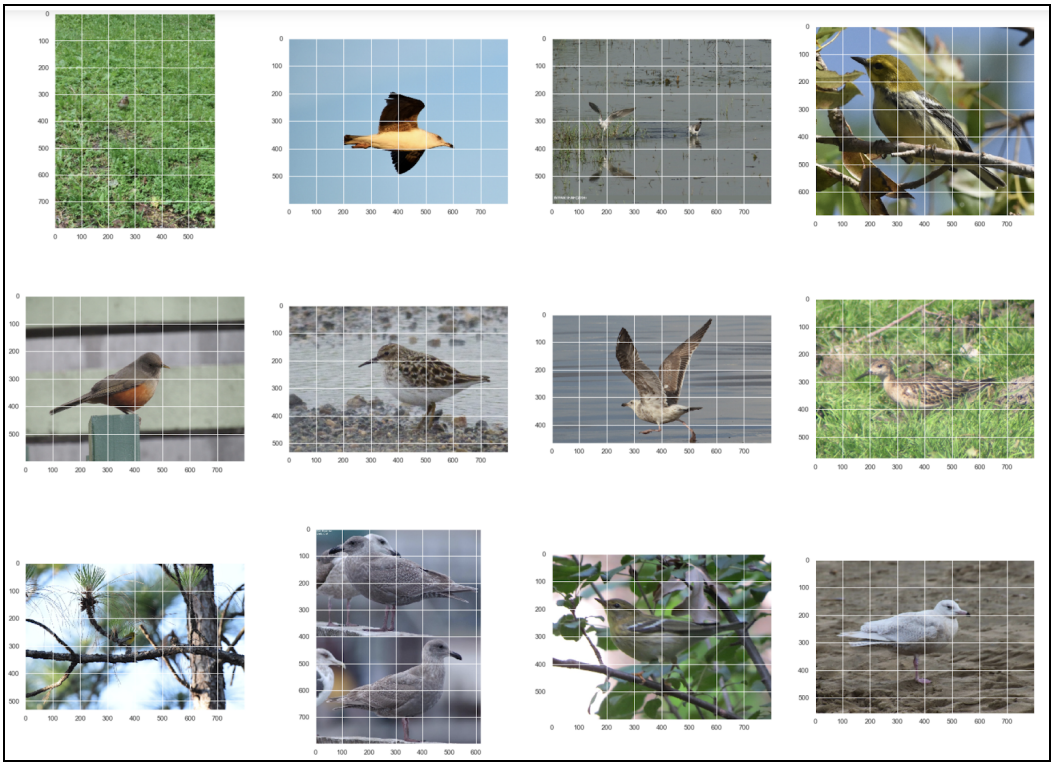
Pixel Height and Width

Each image has a maximum height and width of 800 pixels. The average height is approximately 663 pixels, while the average width is 720. The minimum height and width is 133 and 188, respectively. Images in the dataset tend to be wider (horizontal) than they are taller (vertical).



Sample Images (Birds)

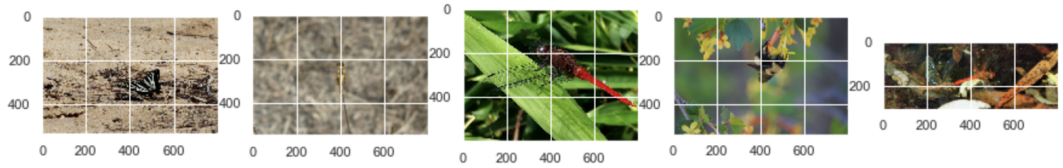
Since our target is to classify birds, let's first look at some images of them in our dataset. The birds are in different environments - in grass, in the air, in trees - and in different positions -- mid-flight or perched. It will be important to be able to train a model that can learn the appropriate features of a bird, even when the image is slightly obscured, or when the bird is not the only thing in the image.



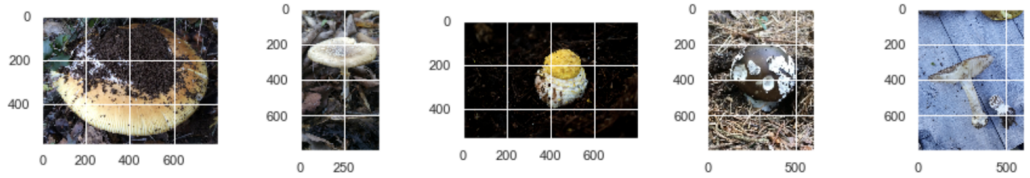
Sample Images (All Categories)

Below are sample images showing five jpeg images of each of the six higher level wildlife categories. 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 our categories). In other cases, the animal is being held in a person's hand.

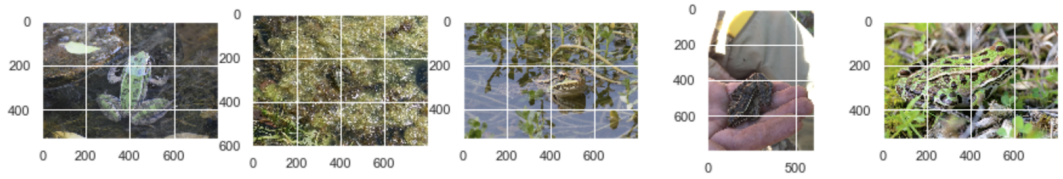
Insects



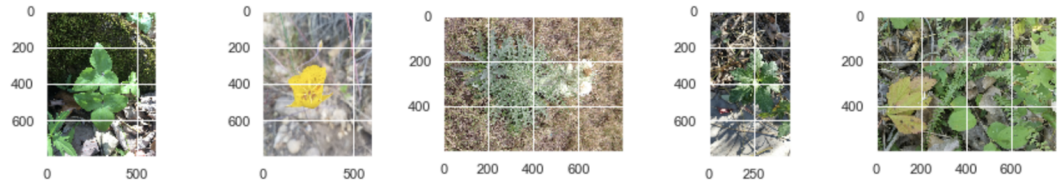
Fungi



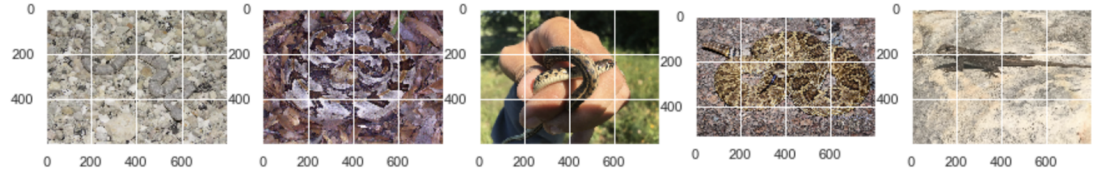
Amphibians



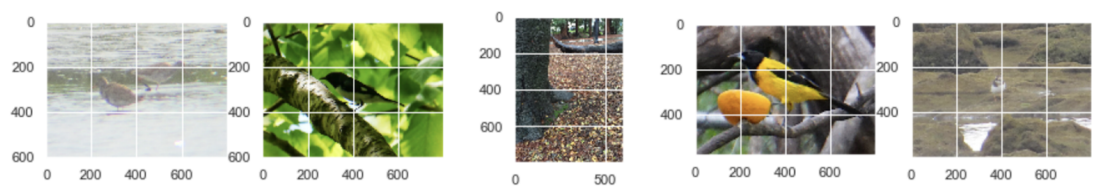
Plants



Reptiles



Birds



In-Depth Analysis & Modeling: Deep Learning

About Neural Networks

This project uses deep learning, specifically neural networks for image classification. They 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 understanding the connections between pixel data. The first few layers in the network look at things like the orientation of lines or simple texture of an image, while layers further down look at more complex features of an image.

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 the 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. While some other machine learning problems simply split training and testing data, many image classification projects split the data further into training, validation, and then testing data. This is the approach I took. **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 were not birds.**

Set-Up

Prior to running the models, constants were set to use in the model, including a set height and width to resize images, batch sizes, and number of epochs. The image below shows the values used in the final model.

```
# Model configuration
img_dir = "../data/raw/" # where images are stored
img_width, img_height = 128,128 # resize images to account for smallest img size
batch_size = 128
no_epochs = 25
no_classes = 2 # choices of bird or not bird
patience = 7 # for Early Stopping callback
```

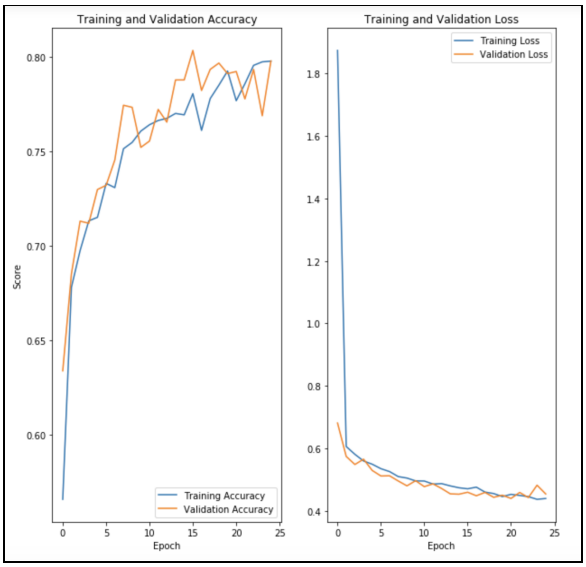
Augment Images

Random augmentation was done for the images in our training and validation datasets, which rescales, zooms, shifts, and standardizes the orientation of images, and can help prevent overfitting of the training data.

```
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. It was a fully connected network using Dense layers -- meaning every layer (in our case, the pixels) is connected to all the units 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 if the corresponding input node is positive or negative), and the output layer used sigmoid activation, which is appropriate for a binary classification. I began by trying 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.



Training/Validation Accuracy & Loss: Model 1

```
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'))
```

While the model seemed to do well in training and testing, 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	0.49	0.55	0.52	585
Bird	0.51	0.46	0.48	615
accuracy			0.50	1200
macro avg	0.50	0.50	0.50	1200
weighted avg	0.50	0.50	0.50	1200

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:

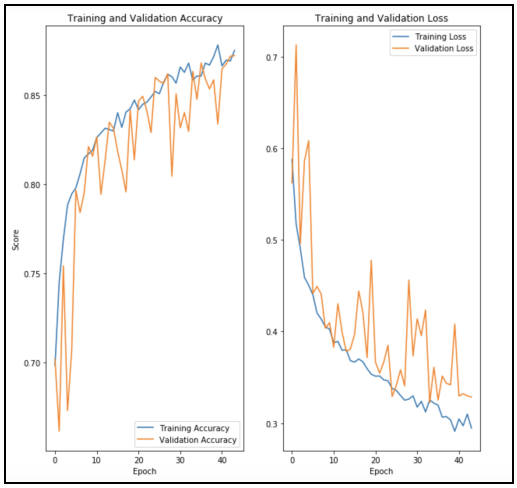
- **Max Pooling:** The max pooling layer can help when there are a large number of parameters, as is the case in this model. It summarizes groups of pixels based on their max value.
- **Dropout Layer:** The dropout layer randomly sets inputs to 0 at a rate of 0.6 at each step during training, which according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

The model also used ReLu activation again in every layer except for the final, which used Sigmoid Activation. The set-up of the layers, along with the number of nodes at each layer, is shown in the image below.

```
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'))
```

Key changes between models:

- ***Total # of Images:*** A total of *15,000 images were used* -- 8400 for training, 3000 for validation, and 3600 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 choose 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 7, 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**: 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 87%. The training seemed to increase the accuracy at a fairly steady pace, though the validation accuracy seemed to bounce around between 80% and 87%.

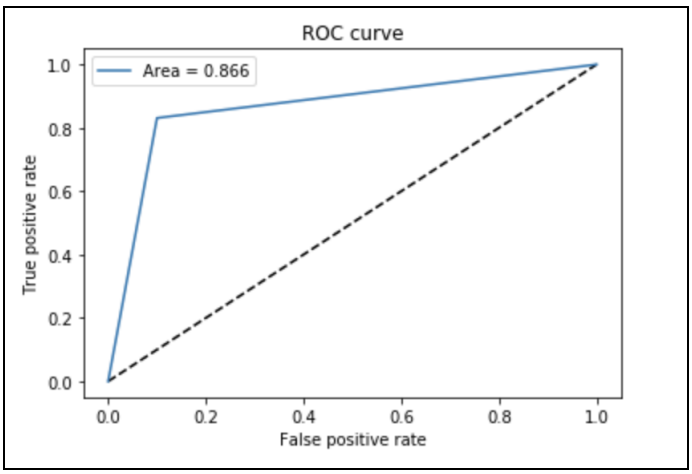
Prediction

The following confusion matrix and classification report gives results for predictions using the second model, which also achieved accuracy score of 87%. All values - precision, recall, and F-1 score - are in the 80% range. For instance, the ability of a model to classify a bird from all of the potential bird images the model was presented with is 83% (recall).

```
Confusion Matrix
[[1647  183]
 [ 299 1471]]
Classification Report
              precision    recall  f1-score   support

    Not Bird       0.85      0.90      0.87      1830
        Bird       0.89      0.83      0.86      1770

    accuracy                           0.87      3600
   macro avg       0.87      0.87      0.87      3600
weighted avg       0.87      0.87      0.87      3600
```

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 model was able to correctly classify 86.6% of all images presented, which is a great deal better than the original model. 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. This is important for researchers who would need to see/classify animals in their natural habitats.

Recommendations & Further Considerations

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

- **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.

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