SpringBoard Captstone 3 – BirdBrain

Classifying bird species for fun and marketing

We currently have a single Bird Photobooth 2.0 in the backyard. It was provided by a local bird food company in exchange for commercial use of the images. It produces 9000+ 4K images a month on average if we remember to change the battery regularly. It is motion activated, any motion sets it off, some of the images will have no birds or deer or something other than what we are looking for.

I also have four RaspberryPi Zero Ws with 8 Megapixel cameras and batteries with solar charging that will be added to that single camera at each of the feeding stations.

There is another bird feeder camera from a kickstarter campaign that is due to arrive any time now.

That is a lot of images. More than any person could reasonably be expected to filter through for quality images of birds.

1. Data

On Kaggle.com I stumbled across as dataset of 325 bird species for classification https://www.kaggle.com/datasets/gpiosenka/100-bird-species it has since grown to 400 species. This is a global dataset for bird species images sorted into directories by species. There is a train, validation and test set already separated out.

The images are 224x224 pixels in full color.

2. Data Wrangling

During the initial inspection of the dataset I discovered that all images were in good valid condition no corruption. I also found an image in both the test and validation sets that were also in the training set

so I flipped those each horizontally to create new distinct images to reduce over fitting on those two species or a false sense of accuracy.

3. Modeling

This is an image dataset, there was no feature engineering, the pixels are the features and the directory names are the classes. After looking at a number of classification neural network architectures, I decided the new EfficientNetV2 set of models would be my starting point. I considered rolling my own but transfer learning using the pretrained weights seemed a better option. At that point I did not have a GPU new enough to be of any help in training against my dataset. So, I started small with EfficientNetV2B0 which for the image size is ideal. Being the smallest model it trained in a reasonable amount of time and worked well as a proof of concept. There was the small factor of the images being used in the real world were much larger with a lot of background to bird and the possibility of multiple birds at a single feeder. So I kicked off the EfficientNetV2S model training on the same data and set about pulling down the TensorFlow CenterNet object detection model and labeling a number of my own images with LabelImg for bounding boxes around birds. That was a bit of a waste of time as the CenterNet object detection model worked well right out of the box. Now with two trained models I tested both the B0 and the S models after running the images through the CenterNet object detection model and pulling out the image data for just within the bounding boxes. This gives multiple bird only images that can then be classified and the data returned for each bird. I began seeing some minor accuracy issues between the classification and the real world images. I began training the EfficientNetV2L model against the dataset (this would have taken weeks to train so I bought a newer computer and it trains roughly in 20 hours to 50 epochs). There was a notable dip in accuracy with this initial training. Simply be resizing the images to the 480x480 pixel size that is expected by the EfficientNetV2L I got a 10% jump in performance without retraining the model. So, I retrained the model and got an extra 2% on top of that. The initial build of the birdbrain reached 83% accuracy over the test set and preformed well on real world images.

4. API

I need this to provide data to allow for the sorting of images at scale. To do this I decided on using FastAPI to create a restful micro service. The service takes an image and returns a JSON string that states the class of object detected, the probability of that being the class detected the pixel location of that object in the image, and if it is a bird class, the species and the probability of that identification.

The API has a fairly simple structure:

GET: / - Returns a page to use for submitting for the JSON data or the image with bounding boxes displayed

POST: / - Returns the JSON data for the classification, scores and location in the image for each object.

POST: /ShowBounds – Returns the image with the bounding boxes drawn on it.

POST: /UploadNewModelPackage – Upload a zip file containing the model, pickle and label files for staging.

POST: /CompareModel – Returns an array of JSON data for the classification, scores and lcoations of the newly staged model and the current production model to compare results.

PUT: /PromoteModel – Used to promote a successful new model to be the production model.

DELETE: /DropModel – Used to remove a staged model that did not preform as well as the production model.

GET: /ping – This is a keep alive method that returns the string “pong” and is used to keep the api resident in memory for some platforms.

GET: /health – This is a health check method that tests that the model loads and the service is up and running for some platforms.

5. Future Improvements

There are several items that are on the road map to improve the model and the api as well as flush out the system for automating the image classification.

1) Add the updated 400 species dataset to the existing set as it is in the same format and will simple be a drop in replacement for the dataset.

2) Add image augmentation to flip, and rotate the images to increase the variations of the images to improve accuracy.

3) Using the Cornell Labs NABirds dataset, convert the images to the format for the existing data set and add the 550 North American specific bird dataset.

4) Add dynamic model loading to the API, currently the model names used by the API are hard coded. Using dynamically loaded models will mean less work replacing the entire neural network architecture when a more advanced model is created and trained.

5) Add the file system watcher script to look for files added to storage directory to be automatically processed with the API and moved to the correct storage location.

6. Recognition

I would like to thank Jeffery Ryan my springboard mentor for all of his insights and advice. The springboard team for the well curated course structure and office hours for interesting ideas and insights in Data Science and Machine Learning. As well as my girlfriend for her obsession with her birds during the pandemic and thus this project.