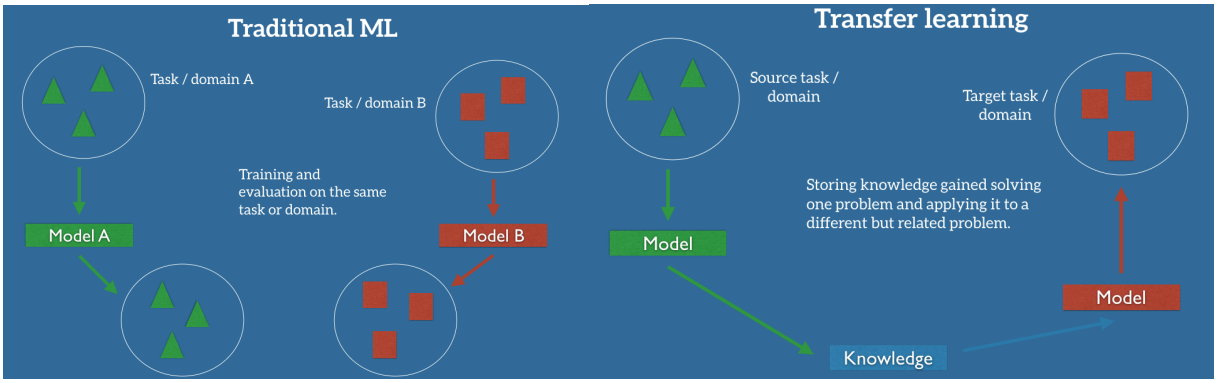
The Shallows

Deep Learning vs Tradition Machine Learning Given Small Datasets

Abstract: This data experiment focused on comparing a Convolutional Neural Net (CNN)'s performance with that of traditional Machine Learning models, namely Support Vector Classifiers given a limited number of samples per class. I found that at small sample sizes, a CNN is not as stable as SVMs, however as sample size grows, CNNs become better and better at a higher rate. As large, unstructured datasets become more and more prevalent it is, CNNs will be able to perform better than SVMs commercially.

Data: We were given a dataset of 4320 images of birds, labelled by species. There were 144 classes of birds in the dataset, and 30 images per class. In order to extract features from the data, we used Google's Inception CNN and used Transfer Learning to better label our dataset. Transfer Learning is essentially the concept of employing a pre-trained CNN to select the most relevant features of the data. According to Stanford professor and Chief Scientist at Baidu, Andrew Ng[1], Transfer Learning will be a massive driver of commercial success in Machine Learning going forward. Indeed, he expects it to make unsupervised learning obsolete in the future. This diagram,[2] taken from Sebastian Ruder visually shows Transfer Learning juxtaposed with traditional Machine Learning.



[1] From Andrew Ng's talk at Conference on Neural Information Processing Systems (NIPS) http://sebastianruder.com/highlights-nips-2016/index.html#thenutsandboltsofmachinelearning

[2] Image from: http://sebastianruder.com/content/images/2017/03/transfer_learning_setup.png

The Inception model is open source and is trained on millions of images over hundreds of hours and has learned what features make images more recognizable. Therefore, we can employ it to better classify our images when compared to the traditional image extraction methods like creating a Numpy Array of pixel values. The result of the Inception models are known informally as Bottlenecks.

The common structure of a CNN for image classification is long convolutional layer chain for feature learning. The learned features will be fed into the fully connected layer for classification. We train the final layer on our own images and create the bottleneck.

The following image shows the structure of TensorFlow's Inception network we are going to use. We have indicated the part of the network that we are getting the output from as our input feature.

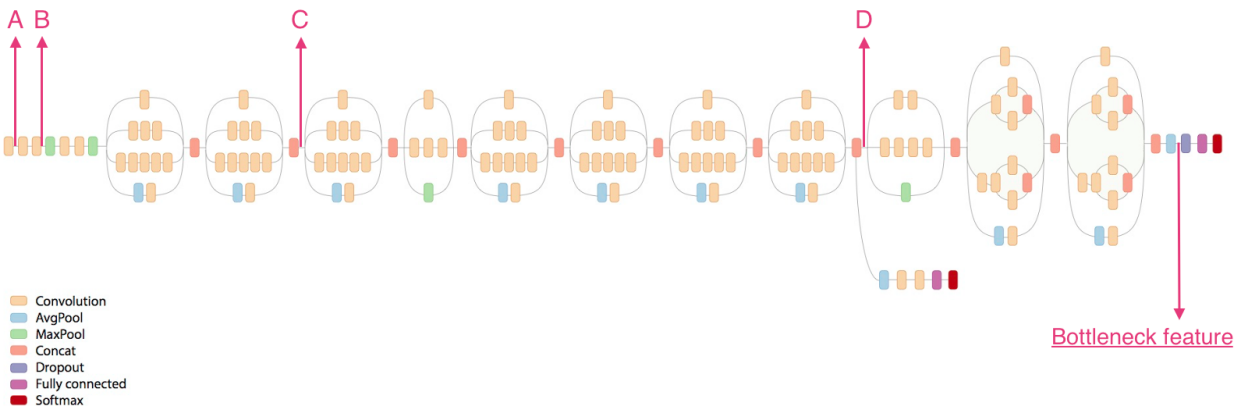


Figure 1Image from https://code.oursky.com/TensorFlow-svm-image-classifications-engine/

Bottlenecks are saved to disk as often the most time-consuming part of actually training CNN models is this feature extraction. As we tune our models, we can reuse these bottlenecks to avoid having to repeat the same procedure.

CNN:

Since this was the first time I was training a neural network, I followed the TensorFlow documentation closely when training my CNN. I employed the function retrain.py [3]to train my CNN and then I adapted those outputs to create the correct TensorFlow graphs and classify a further 4320 images. I chose the default Parameter, i.e learning rate of 0.01 and the 500

3

https://github.com/TensorFlow/TensorFlow/blob/master/TensorFlow/examples/image_retraining/retrain.py

learning steps. However, the retrain function did not allow me to run as 30 samples per class were too few. There were simply not enough samples in there for me to use as a test set and a validation set, while still being able to get a good fit.

```
Traceback (most recent call last):
  File "retrain.py", line 1062, in <module>
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/platform/app.py
", line 44, in run
    _sys.exit(main(_sys.argv[:1] + flags_passthrough))
  File "retrain.py", line 866, in main
    bottleneck_tensor))
  File "retrain.py", line 484, in get_random_cached_bottlenecks
    image_dir, category)
  File "retrain.py", line 213, in get_image_path
    mod_index = index % len(category_list)
ZeroDivisionError: integer division or modulo by zero
ubuntu@ip-172-31-52-210:~/tf_files$
```

However, due to the feature of bottlenecks being written to disk (as noted above), all this time was not wasted. I then realized that I would have to augment my data.

Data Augmentation: My first instinct was to simply copy all the images in my dataset and train my CNN on them to keep a baseline for my model. After doing so I was able to reach an Accuracy score of 63% on my test set. This was a promising baseline to compare other efforts to. I knew that simply copying my samples would lead to overfitting my model to the images in my training set, so I researched many data augmentation techniques. According to researchers Wong, Gatt, and Stamatescu[4], data augmentation in the data space, transformations to the images themselves were much better than any feature engineering or synthetic minority oversampling with images.

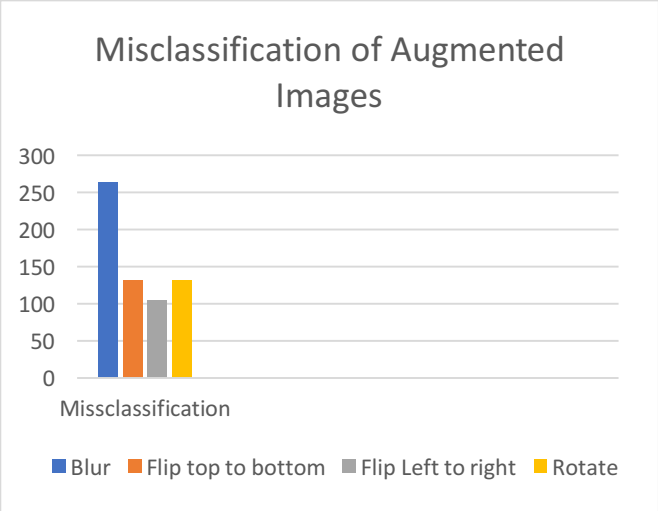
I therefore wrote a script that employed the Pillow library to quintuple my dataset. I added the following transformations to my data:

- Gaussian blur
- Rotation of 180°
- Mirror image Front to Back
- Mirror Image Top to Bottom

Refer to the appendix for examples of images. I then re-ran my CNN with the default Hyper-Parameters and achieved an accuracy score of 55%. I was disheartened to see a lower score than my baseline prediction. It clearly showed that my image augmentation was counterproductive as my CNN was not able to identify the features that identified the images clearly. I printed the images that were misclassified in the hopes to see which transformation was the worst. It was clear that the Gaussian Blur transformation, instead of preventing overfitting like I hypothesized, actually made my algorithm worse. Indeed, it was 'underfitting' my data.

---

[4] Understanding data augmentation for classification: when to warp? (Wong, Gatt, and Stamatescu)
https://arxiv.org/pdf/1609.08764.pdf



I therefore removed the blurred images and then retrained my models. This time, I received a scored of 66%. We were finally making significant progress. Excellent.

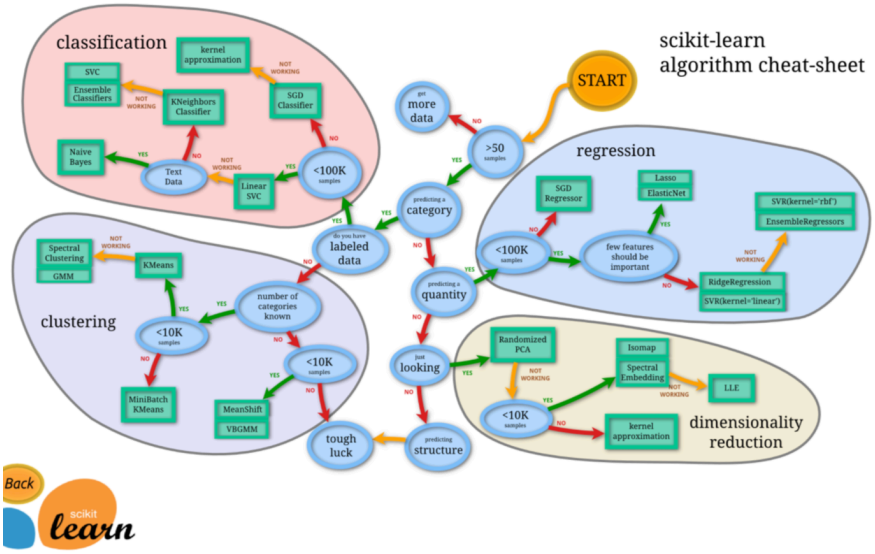
Tuning Hyper-Parameters: In order to improve the accuracy of my results, I started to experiment with the hyper-parameters of my CNN. I researched how best to tune my Hyper parameters and found a research paper[5] by Dr Yoshua Bengio of Université de Montréal. I saw that he recommended that I fine tune the learning rate first as it had the highest effect on model performance. Conventional wisdom would show that a slower learning rate would lead to a better model, however I saw only marginally better results of 68% in my test set when I set small learning steps. I tried 0.01 and even 0.001 but the model seemed to be getting worse along with performance. I then tried a step size of 0.75 and was ecstatic to find a 75% accuracy score. Looks like my intuition was dead wrong! I then re-ran models with the parameters of 0.8, 0.85, 0.9 and 0.95. I also saw that the Cross-entropy of these models, which the Softmax regression employed by the CNN was employing was reaching convergence much faster so I reduced the amount of steps. I then used a naïve ensemble of the results produced by my models by taking the mode of all the predictions by my CNNs. The final result on my test set (20% of my full data) was 82%. A good result in my opinion!

SVMs:

According to Sci-Kit Learn, the best method to represent traditional machine learning in this scenario models was the Support Vector Classifier. I used a one vs many approach to the SVMs and used the GridSearch module of sklearn to find the optimum parameters for my classifier. I used 10-fold cross-validation as well to prevent overfitting my data and explored

[5] Practical Recommendations for Gradient-Based Training of Deep Architectures https://arxiv.org/pdf/1206.5533v2.pdf

between the rbf and linear kernel. All the permutations of my GridSearch are visible in the appendix



I optimized the code to employ the power of 16 CPUs on my AWS instance to parrellize this process. After 132 minutes, my model reached a precision of 72%. Details of this can be referred to in the appendix. The GridSearch function showed me that my model had arrived at the following parameters as the optimal.

```
Best parameters set:
{'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
```

Conclusion:

As we can see in our experiment, the use of transfer learning in our models is very powerful, especially when it comes to complex data sets like images. It can perform feature selection that can make both deep learning and traditional machine learning models powerful. In the end, we can see that SVMs can perform well with transfer learning, achieving a success rate of 72% with a small dataset of 30 images per class. Indeed, the CNN model recommended by Google does not even run with 20 images per class and at 30 samples per class gave errors that could only be circumvented by deploying less images into the test set, something I was not willing to do due to overfitting. However, with some experimentation and data augmentation, we are able to see that the CNN is able to perform well when there are more than 100 samples per class. As a next step, I would like to explore the dropout concept as I believe that my model was overfitting due to it showing a very high training testing accuracy and a much lower validation accuracy. I just need to be sure not to use a Gaussian blur like before!

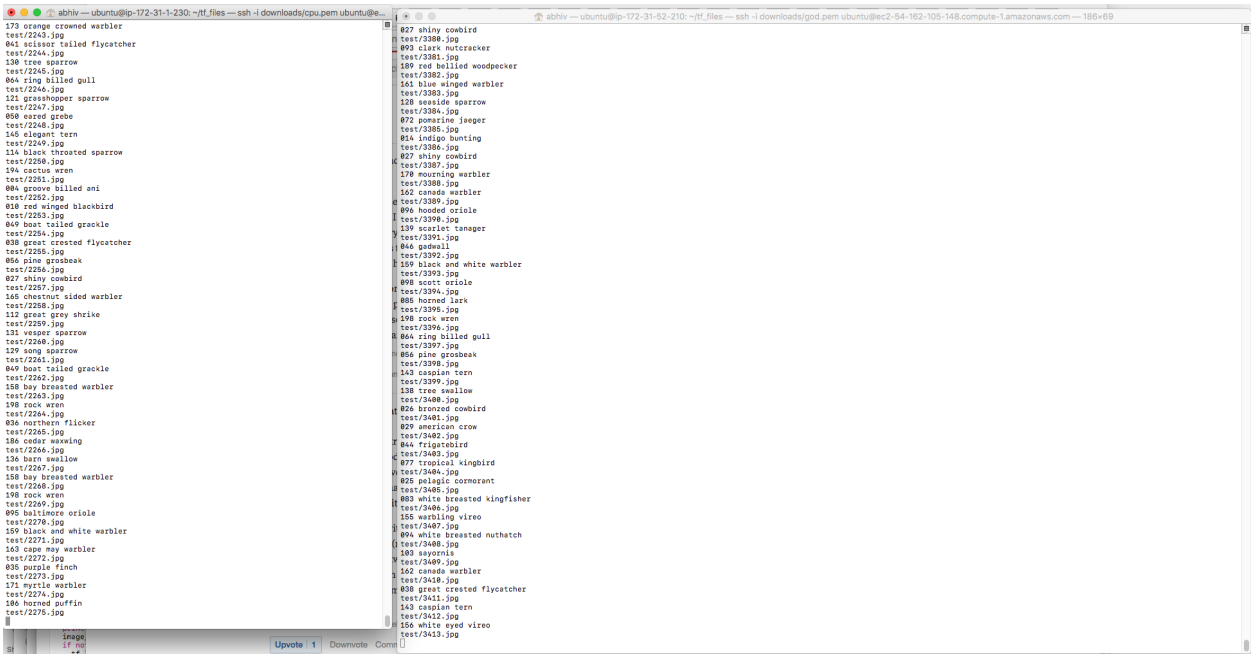
Appendix:

Data Augmentation examples (including Gaussian blur)

GridSeach Parameters for SVM.

```
param = [
    {
        "kernel": ["linear"],
        "C": [1, 10, 100, 1000]
    },
    {
        "kernel": ["rbf"],
        "C": [1, 10, 100, 1000],
        "gamma": [1e-2, 1e-3, 1e-4, 1e-5]
    }
```

Running Multiple models in AWS at the same time:



SVM classification report.

Classification report:

	precision	recall	f1-score	support
American_Crow	0.20	0.33	0.25	3
American_Goldfinch	0.83	0.83	0.83	6
American_Pipit	0.60	0.75	0.67	4
American_Redstart	1.00	0.50	0.67	4
Anna_Hummingbird	0.67	0.75	0.71	8
Baltimore_Oriole	1.00	1.00	1.00	4
Barn_Swallow	0.73	1.00	0.84	8
Bay_Breasted_Warbler	0.57	0.57	0.57	7
Belted_Kingfisher	0.57	0.57	0.57	7
Bewick_Wren	0.00	0.00	0.00	3
Black_And_White_Warbler	0.75	0.75	0.75	4
Black_Billed_Cuckoo	1.00	0.86	0.92	7
Black_Footed_Albatross	0.86	1.00	0.92	6
Black_Tern	0.50	0.33	0.40	6
Black_Throated_Sparrow	0.43	1.00	0.60	3
Blue_Grosbeak	1.00	0.40	0.57	5
Blue_Headed_Vireo	0.67	0.80	0.73	5
Blue_Jay	1.00	1.00	1.00	10
Blue_Winged_Warbler	0.67	0.44	0.53	9
Boat_Tailed_Grackle	0.60	0.50	0.55	6
Bobolink	1.00	0.50	0.67	4
Bohemian_Waxwing	0.60	0.50	0.55	6
Bronzed_Cowbird	1.00	0.50	0.67	8
Brown_Pelican	1.00	1.00	1.00	4
Cactus_Wren	0.64	0.78	0.70	9
California_Gull	0.33	0.60	0.43	5
Canada_Warbler	0.60	1.00	0.75	3
Cape_Glossy_Starling	0.78	1.00	0.88	7
Cape_May_Warbler	0.57	0.80	0.67	5
Carolina_Wren	1.00	0.33	0.50	6
Caspian_Tern	0.50	0.40	0.44	5
Cedar_Waxwing	0.88	0.88	0.88	8
Cerulean_Warbler	1.00	0.67	0.80	9
Chestnut_Sided_Warbler	0.80	0.57	0.67	7

Chipping_Sparrow	0.50	0.22	0.31	9
Clark_Nutcracker	0.60	1.00	0.75	6
Cliff_Swallow	0.12	0.25	0.17	4
Common_Tern	0.33	0.33	0.33	6
Common_Yellowthroat	0.56	0.83	0.67	6
Dark_Eyed_Junco	0.88	1.00	0.93	7
Downy_Woodpecker	1.00	0.88	0.93	8
Eared_Grebe	0.67	0.57	0.62	7
Eastern_Towhee	0.88	0.88	0.88	8
Elegant_Tern	0.67	1.00	0.80	2
European_Goldfinch	1.00	0.88	0.93	8
Evening_Grosbeak	0.83	0.71	0.77	7
Fish_Crow	0.40	0.50	0.44	4
Florida_Jay	0.89	1.00	0.94	8
Forsters_Tern	0.44	0.57	0.50	7
Fox_Sparrow	0.50	0.43	0.46	7
Frigatebird	0.75	0.75	0.75	4
Gadwall	0.75	0.75	0.75	8
Geococcyx	0.80	0.80	0.80	5
Grasshopper_Sparrow	0.67	0.50	0.57	8
Great_Crested_Flycatcher	0.00	0.00	0.00	3
Great_Grey_Shrike	0.25	0.50	0.33	4
Green_Kingfisher	0.75	0.86	0.80	7
Green_Tailed_Towhee	0.75	0.60	0.67	5
Green_Violetear	0.80	1.00	0.89	4
Groove_Billed_Ani	0.33	1.00	0.50	2
Harris_Sparrow	0.88	1.00	0.93	7
Heermann_Gull	0.57	0.67	0.62	6
Henslow_Sparrow	0.60	0.43	0.50	7
Herring_Gull	0.57	0.50	0.53	8
Hooded_Merganser	1.00	0.75	0.86	8
Hooded_Oriole	0.60	0.43	0.50	7
Hooded_Warbler	0.75	0.67	0.71	9
Horned_Grebe	0.67	0.86	0.75	7
Horned_Lark	1.00	0.88	0.93	8
Horned_Puffin	0.60	1.00	0.75	3
House_Sparrow	0.71	0.71	0.71	7
Indigo_Bunting	0.60	1.00	0.75	3
Ivory_Gull	0.86	1.00	0.92	6
Laysan_Albatross	1.00	0.86	0.92	7
Least_Tern	0.50	0.67	0.57	6
Loggerhead_Shrike	0.57	0.44	0.50	9
Long_Tailed_Jaeger	0.33	0.17	0.22	6
Louisiana_Waterthrush	0.56	0.71	0.63	7
Mallard	1.00	0.67	0.80	6
Marsh_Wren	0.58	1.00	0.74	7
Mockingbird	0.25	0.20	0.22	5
Mourning_Warbler	0.67	0.75	0.71	8
Myrtle_Warbler	0.57	1.00	0.73	8
Nashville_Warbler	0.80	0.44	0.57	9
Nighthawk	1.00	0.80	0.89	5
Northern_Flicker	0.62	0.71	0.67	7
Northern_Fulmar	0.80	0.50	0.62	8
Northern_Waterthrush	0.40	0.33	0.36	6
Olive_Sided_Flycatcher	0.50	0.75	0.60	4
Orange_Crowned_Warbler	0.62	0.62	0.62	8
Ovenbird	0.67	0.80	0.73	5
Pacific_Loon	0.80	0.80	0.80	5
Palm_Warbler	0.33	0.40	0.36	5
Pelagic_Cormorant	0.75	0.75	0.75	4
Pied_Billed_Grebe	0.60	0.75	0.67	4
Pied_Kingfisher	0.83	0.83	0.83	6
Pileated_Woodpecker	1.00	0.86	0.92	7
Pine_Grosbeak	0.80	1.00	0.89	4
Pine_Warbler	0.33	0.50	0.40	4

Pomarine_Jaeger	0.43	0.60	0.50	5
Prairie_Warbler	0.50	0.60	0.55	5
Prothonotary_Warbler	0.67	0.50	0.57	4
Purple_Finch	1.00	1.00	1.00	8
Red_Bellied_Woodpecker	1.00	0.83	0.91	6
Red_Breasted_Merganser	0.71	1.00	0.83	5
Red_Eyed_Vireo	0.71	0.71	0.71	7
Red_Headed_Woodpecker	1.00	1.00	1.00	4
Red_Winged_Blackbird	0.67	1.00	0.80	2
Ring_Billed_Gull	0.67	0.80	0.73	5
Ringed_Kingfisher	0.75	0.75	0.75	4
Rock_Wren	1.00	0.64	0.78	11
Rose_Breasted_Grosbeak	0.86	0.86	0.86	7
Ruby_Throated_Hummingbird	1.00	0.33	0.50	6
Rufous_Hummingbird	0.62	0.83	0.71	6
Rusty_Blackbird	0.50	0.17	0.25	6
Sage_Thrasher	0.75	0.50	0.60	6
Savannah_Sparrow	0.30	0.33	0.32	9
Sayornis	0.17	0.33	0.22	3
Scarlet_Tanager	1.00	0.90	0.95	10
Scissor_Tailed_Flycatcher	1.00	0.43	0.60	7
Scott_Oriole	0.56	0.56	0.56	9
Seaside_Sparrow	0.80	0.80	0.80	5
Shiny_Cowbird	1.00	0.20	0.33	5
Song_Sparrow	0.17	0.33	0.22	3
Summer_Tanager	1.00	0.75	0.86	4
Tree_Sparrow	0.25	0.25	0.25	4
Tree_Swallow	1.00	0.60	0.75	5
Tropical_Kingbird	0.50	0.50	0.50	2
Vermilion_Flycatcher	0.67	1.00	0.80	6
Vesper_Sparrow	0.50	0.40	0.44	10
Warbling_Vireo	0.60	0.60	0.60	5
Western_Grebe	1.00	1.00	1.00	5
Western_Gull	1.00	0.43	0.60	7
Western_Meadowlark	1.00	0.60	0.75	5
Western_Wood_Pewee	0.50	0.33	0.40	6
White_Breasted_Kingfisher	1.00	1.00	1.00	6
White_Breasted_Nuthatch	0.83	0.62	0.71	8
White_Crowned_Sparrow	0.50	1.00	0.67	3
White_Eyed_Vireo	0.60	0.43	0.50	7
White_Necked_Raven	1.00	0.80	0.89	5
White_Throated_Sparrow	0.67	0.67	0.67	9
Wilson_Warbler	0.83	0.83	0.83	6
Winter_Wren	1.00	0.88	0.93	8
Yellow_Warbler	0.29	0.50	0.36	4
avg / total	0.72	0.67	0.67	864