Fluke_Net Project Report

Connor Barlow, Caelan Booker, Adicus Finkbeiner, Logan Pashby, Dylan Thompson

June 7, 2020

1 Overview

The objective of this project was to identify humpback whales from pictures of their flukes (e.g. their tails). The importance of this objective lies in conservation. Whale populations are currently monitored by by human scientists and volunteers who manually identifying individual whales. HappyWhale, a service that has been manually recording and tracking whales for the past 40 years, created a Kaggle challenge to see if machine learning could provide a less human-intensive alternative [4]. The dataset that HappyWhale provided had very few examples for most classes. Therefore, we took a Few-Shot Learning (FSL) approach to this problem, implementing a Prototypical Network [3] with Strength and Structure Training [5] image classifier to identify individual whales given few examples per whale.

2 Background

The following concepts are needed to understand our approach and results:

- **Few-shot learning:** Few-shot learning (FSL) is a category of deep learning designed for completing a task with very few examples. This is a growing area of research that attempts to address a shortcoming of deep-learning: its reliance on huge datasets [8]. Most FSL models train using training episodes (see below), though there is a growing literature on meta-learning and ensemble approaches that break this distinction.
- **Episodic training:** Episodic training is a common method of training FSL models. Episodes are typically described as N-shot K-way, where N is the number of examples per class and K is the number of classes per episode [8]. In our Prototypical-Learning model, N is further broken up into a query and support set, where the support is used to create the class prototype against which the query set is compared. [3]
- **Distractor classes:** A distractor class is a class of images that belong to no other class in a given dataset. These are used most regularly in anomaly detection, but one is present in our original dataset in the form of a "new_whale" class.

3 Prior Work

Generalizing from a Few Examples: A Survey on Few-Shot Learning (Wang et al.)
The core issue with FSL is that for a model h, empirical risk $\hat{R}(h)$ can no longer provide a reasonable estimate of ground-truth risk $R(h)$, and so techniques that rely solely on minimizing $\hat{R}(h)$(e.g. SGD) can not be used directly. Thus an important goal of FSL learning is to find ways to mitigate the effect of this unreliable empirical risk, and the paper gives 3 general approaches. A subset of these approaches, embedding learning, was the one we converged on for further research.

Learning Structure and Strength of CNN Filters (Keshari et al.)
A Structure and Strength Filtered CNN (SSF-CNN) attempts to reduce the risk of overfitting on a small sample size by reducing the number of parameters that need to be trained. SSF-CNN attempts to reduce the

number of parameters that need to be learned in a CNN. The filter's initial weights are learned separately with hierarchical dictionary learning. Instead of training every weight in the convolutional kernel, a single scalar "strength parameter" is learned for each of the kernel's filters.

Prototypical Networks for Few-shot Learning (Snell et al.)
This model addresses the problems of deep learning in FSL by defining learning as a problem of clustering in an embedding space. The structure of episodes described in the background section is used, where the average embedding of the support set is the "prototype" of a class in a single episode. The goal then is to minimize the distance of the query set's embedding to its corresponding class embedding, and maximize its distance to every other class.

Meta-Learning for Semi-Supervised Few-Shot Classification (Renet al.)
This paper describes an extension of Prototypical Nets that incorporates distractor classes. To each episode, unlabeled data is added, and the model uses this extra information to create new prototypes when unlabeled data is sufficiently close, and otherwise trains in a way to "push" those unlabeled examples further away from known class prototypes in the embedding space.

4 Methodology

We repeatedly culled our dataset to limit the scope of the problem to one which we could realistically tackle. Our limited dataset, in turn, informed our approach. Initially, our model included only 25,000 images for 5,000 classes. Of these 5,000 classes, over 2,000 included only a single example. One class was a distractor class, including all examples that did not fall into one of the labeled classes. We chose to ignore the distractor class and any classes containing fewer than 10 examples. This left us with a dataset of 5,000 images and 273 classes. We divided the remaining dataset into train/dev/test by shuffling and allocating 80% to training and 10% each to dev and test.

To work within the constraints presented by this dataset, we augmented a typical image classifier DNN to be more appropriate for the FSL task by (1) replacing the convolutional layers with SSF-CNN layers and (2) modifying the training pipeline to implement the Prototypical Network's classification & loss functions. We also augmented our data by applying a random combination of cropping, rotation, horizontal reflection, and hue/contrast/brightness jitter.

We implemented the SSF-CNN layers in two parts. First, we used SKLearn's dictionary learning library to generate a sparse represenation of the entire dataset, which we then used to initialize the SSF-CNN kernel weights. Second, we implemented a custom "Strength Convolutional Layer" that prevented the kernel weights from being updated during backpropagation and instead scaled each filter in the kernel by a single trainable scalar parameter.

A prototypical network can be thought of as an embedding function that computes a representation of each input in an embedding space $f_\phi : \mathbb{R}^{\text{input}} \to \mathbb{R}^{\text{embed}}$ with learnable parameters ϕ. To implement f_ϕ, we simply modified the final fully-connected layer in our image-classifier CNN to output an *embed*-dimensional vector. In each training episode (see Background), the model calculates the average of the embeddings for the support set of each class to yield class prototypes. Thus, for class k and its associated support set S_k, the prototype p_k is found by:

$$p_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i)$$

After it has calculated the prototypes for an episode, the model calculates the negative Euclidean-distance between the query set embeddings and all class prototypes. From here, all we had to do was calculate cross-entropy loss with respect to these distances and their true classes, since the goal is to minimize the distance of a query embedding and its true class's prototype while maximizing its distance to every other class prototype.

In order to interpret our results, we implemented several baselines. Four were different random sampling strategies, two of which (uniform and stratified) weighted classes equally while the other two (prior and frequency) were biased toward classes with more examples. We included k-Nearest-Neighbors with k=1, since, like a prototypical network, it is a clustering function. For a more direct comparison with our model,

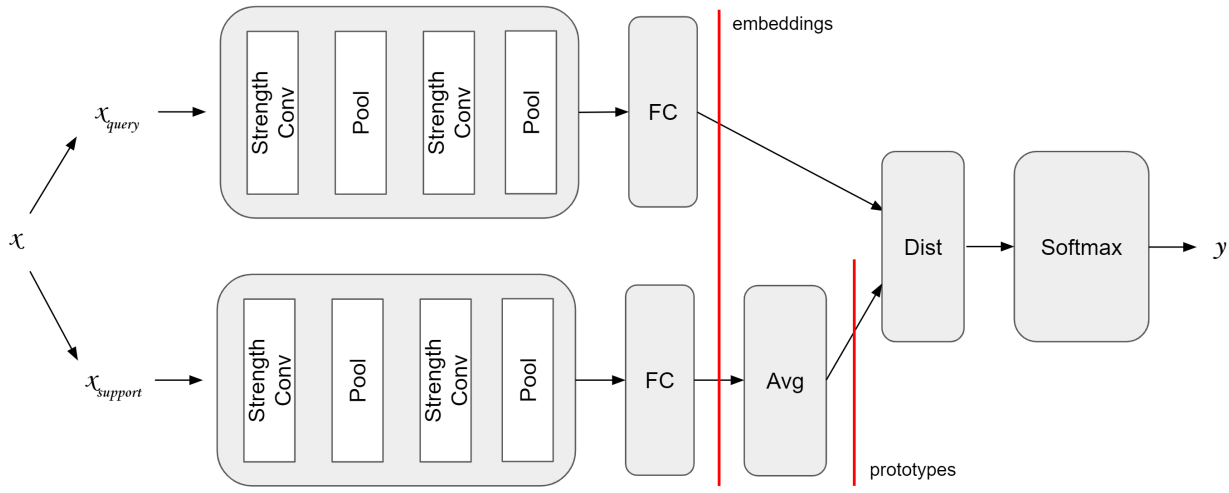

Figure 1: fluke_net architecture. Embeddings are generated for both the query and support set, then averaged over each class in the support set to generate class prototypes. Euclidean distance between each query embedding and all prototypes is then calculated and the model's prediction for each element of the query set is yielded by softmax.

we trained a vanilla CNN image classifier with the same architecture as our final model, except with an additional fully-connected layer after the embedding layer. As a final baseline, we created a survey to test human performance on this task. The survey simulated a 5-Way training episode with 3 support examples per class that they would memorize, followed by a test with 3 query examples per class. We had 31 participants.

5 Experimental Results

We initially implemented a Bayesian hyperparameter search using Chocolate[2] that traversed a search space including: the number of convolutional layers, size/stride of convolutional & pooling layers, episode size (the K in N-shot K-way, see Background), and several other hyperparameters. Unfortunately, this search space was too large and failed to converge in a reasonable amount of time given the constraints on our use of the cluster, so we instead pivoted to a simpler approach we had seen in [6].

Learning Rate Sweep	
LR	mAP@1
1e-4	0.11
1e-3	0.15
1e-2	**0.34**
1e-1	0.26
1e0	0.24

Episode Size	
Size	mAP@1
4	0.28
8	0.27
16	0.23
32	**0.20**

Pretrained / Strength Filters			
mAP@1		Pretrained	
		Yes	**No**
Strength	Yes	0.22	—
	No	0.25	**0.28**

Epoch Sweep	
Epochs	mAP@1
40	0.28
80	0.31
120	**0.38**
160	0.38

Tables 1-4: Hyperparameter sweep. We conducted a grid search across each individual hyperparameter while holding the others constant, at each step picking the value that produced the best validation set accuracy.

In this alternative approach, we selected a model architecture that had been shown to converge during manual tuning, which eliminated a number of hyperparameters related to the architecture, and we further restricted the set of hyperparameters to just learning rate, episode size, whether to use either/both pretraining and strength filters, and number of epochs. Then, we conducted a grid search across each individual

hyperparameter (in the order listed above) while holding the others constant, at each step picking the value that produced the best validation set accuracy (see Tables 1-4).

We evaluated our model's performance using the mean average precision (mAP) metric. During hyperparameter tuning, we were concerned with optimizing the model's best guess, so we based our tuning decisions on mAP@1. When evaluating our model on the test set, however, we wanted a metric that reflected how often the true class was in our model's top few predictions even when its best guess was incorrect, so we calculated mAP@1, mAP@5, and mAP@20.

Comparison with Baselines	
Approach	Test mAP@1
Random (Uniform)	0.005
Random (Stratified)	0.005
Random (Prior)	0.097
Random (Frequency)	0.097
kNN (k=1, 8 examples per class)	0.019
Vanilla CNN	0.022
Human ID	**0.549**
fluke_net	0.331

Table 5: Comparison with baselines. All mAP@1 values (except from the human survey) were evaluated on the test set.

The automated baselines had a test set mAP@1 near the expected value for random selection. Human recognition from our informal survey was, unsurprisingly, much better with an mAP@1 around 0.55 on their 5-way classification task. With optimal hyperparameters, our model's test set performance fell somewhere in the middle with mAP@1 of 0.33, mAP@5 of 0.44, and mAP@20 of 0.46. Our model performs significantly better than both random selection and vanilla CNN image classification, but under-performs compared to human classification.

6 Conclusions and Future Work

Our goal was to train a model that would learn to distinguish unique whales given few examples per whale. Our solution was to use a prototypical network with data augmentation and SSF-CNN layers to reduce the number of trainable parameters. After a hyperparameter search, we found that pre-trained filters did not result in an increase in dev set accuracy, possibly because the sparse representation produced by dictionary learning was a poor choice of initial weights for the kernels. With our final prototypical network architecture, we achieved a mAP@1 performance of 33% on the test set, which, while falling significantly short of our tested human accuracy of 55%, was also a huge improvement over our baselines.

We believe the greatest strength of our model is the relative simplicity of our final architecture, which demonstrates the power of prototypical networks as an approach to FSL. The greatest weakness of our approach and results was our inability to work with the entire dataset as given, and this would be the primary focus of future work. The semi-supervised extension of prototypical networks would be a promising alternative, as it would allow the inclusion of the new_whale distractor class in training [7]. However, this approach is complex and would still require the exclusion of our most sparse classes, so to address this, an alternative would be a Siamese Neural-Net [1]. This network was designed for 1-shot image training, but allows for classes with greater numbers of examples to be utilized by carefully choosing pairs of images to compare.

References

[1] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. *Siamese Neural Networks for One-Shot Image Recognition*. 2015.

[2] *Chocolate Documentation*. 2016. URL: `https://chocolate.readthedocs.io/en/latest/index.html`.

[3] Jake Snell, Kevin Swersky, and Richard S. Zemel. *Prototypical Networks for Few-shot Learning*. 2017. arXiv: `1703.05175 [cs.LG]`.

[4] *Humpback Whale Identification*. Nov. 2018. URL: `https://www.kaggle.com/c/humpback-whale-identification`.

[5] Rohit Keshari et al. *Learning Structure and Strength of CNN Filters for Small Sample Size Training*. 2018. arXiv: `1803.11405 [cs.CV]`.

[6] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002v2 [cs.CV].

[7] Mengye Ren et al. "Meta-Learning for Semi-Supervised Few-Shot Classification". In: *CoRR* abs/1803.00676 (2018). arXiv: 1803.00676. URL: http://arxiv.org/abs/1803.00676.

[8] Yaqing Wang et al. *Generalizing from a Few Examples: A Survey on Few-Shot Learning*. 2019. arXiv: 1904.05046 [cs.LG].