Monitoring Illegal Fishing through Image Classification

Jason Frost
jfrost@stanford.edu

Taylor Geisler
tgeisler@stanford.edu

Antariksh Mahajan
am95@stanford.edu

Abstract

In this paper we apply computer vision techniques to identify the species of fish caught on fishing boats. We compare several different methods of classification, from basic algorithms such as Naïve Bayes to pre-trained Convolutional Neural Networks (CNNs) to training our own CNN from scratch. After tuning various parameters to balance variance and bias, we find that training a CNN from scratch produced the highest validation accuracy.

1. Motivation

Overfishing presents a difficult problem for the sustainability of global ecosystems and food systems. Two-thirds of global fish stocks are currently over-exploited, making the protection of endangered and keystone fish species an urgent priority (The Economist, 2014).

This project aims to classify images of fish taken on fishing boats based on species. This will help organizations such as Nature Conservancy, which published our dataset, to monitor illegal fishing activity.

2. Related Work

Most modern sophisticated computer vision work uses Convolutional Neural Networks (CNNs). One study used a five convolutional and three fully-connected layers to achieve a maximum accuracy of 84.7% (Alex Krizhevsky, 2012), suggesting to us that CNNs would be the best-performing algorithm.

Researchers have used CNNs in several ways. While the study above trained a CNN from scratch, other studies have identified the value of using pre-trained CNNs to extract general features from images. Razavian, Azizpour, Sullivan, and Carlsson (2014) reported positive results using a pre-trained CNN to extract 4096 features from images, then applying a Support Vector Machine (SVM) classifier. This suggested to us that other than training a CNN from scratch, using a pre-trained model may also produce good classifiers.

3. Dataset

We used a dataset of 3777 images classified into 8 labels, published by Nature Conservancy on Kaggle. The labels included six species of fish as well as one “No fish” and one “Other” label.

From the outset, the data presented several key challenges. The images were all of different sizes and had all been taken at different times of day. Some contained more than one species of fish. The dataset was also small, containing only a few thousand images, while most CNNs in previous work had been trained on millions of images. Most significantly, the fish were only a tiny part of many of the images. Figure 1 shows two different species of fish that our classifier would eventually have to distinguish.

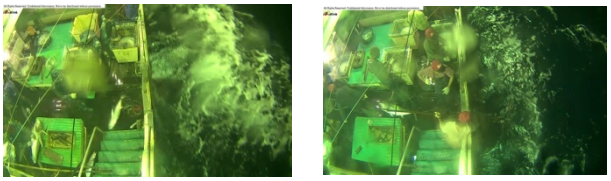

Figure 1. The left picture is labeled “ALB”, while the right picture is labeled “BET”.

4. Methods

The training set contained images of various sizes. Because neural networks require inputs of constant size, and for subsequent ease of processing, all images were scaled to a uniform 224 x 224 x 3 size. This results in an input of size 150,528. All original images originally contained integer RGB values from 0 to 255. During preprocessing, the dataset was rescaled and translated to have zero mean and a variance of 1 over all input variables. The given data was split randomly into 70% train and 30% groupings.

4.1 Naïve Bayes (NB)

We implemented a simplistic initial model by grouping colors into buckets and using NB to classify images according to the number of occurrences of each bucket.

4.2 Pre-Trained CNN Feature Generation

Convolutional neural networks (CNNs) are cutting-edge tools for numerous computer vision learning applications. CNNs resemble traditional fully-connected neural networks but integrate additional optimizations by assuming the inputs are images. Each layer of the CNN contains depth layers that act as filters on the output of the image coming from the previous layer learn to detect features such as edges, color grouping, or shapes. Because these networks learn to recognize these types of universally useful features it is useful and efficient to employ pre-trained CNNs as feature-generators. They are commonly used as starting points in training custom CNNs.

The CNN architectures employed in this report include the following layers:

1. 2D Convolutional Layer (Conv) - Computes the output of each neuron by convoluting the signal from a small volume of inputs from the previous layer. The convolution volume is defined by the filter-size and stride of the layer. Each depth layer is defined by a set of weight and bias parameters.

2. Rectified Linear Unit Activation Layer (ReLU) - Applies an activation function to each neuron output: f(x) = max(0,x).

3. Pooling Layer (Pool) - Downsamples the input along spatial dimensions, retaining depth dimension.

4. Dropout Layer - Regularization layer that randomly zeroes a fraction of input signals during training.

5. Fully Connected Layer (FC) - Traditional 1-dimensional NN layer in which every input is connected to every layer

6. Softmax - Logistic regression activation layer extension to handle n-dimensional classifiers.

To extract feature from these images, we employ the VGG-16 pre-trained CNN that has been trained on the ImageNet dataset. It reduces the order 100,000 input features to a 1000 feature vector. The network is tuned such that the 1000 outputted features are correlated with defining characteristics of the ImageNet library. We then use the 1000 feature CNN codes as inputs to various algorithms (random forest, SVM) to finish the classification.

4.2.1 Random Forest

A decision tree works by repeatedly choosing the most important features by which to split the data. During the training process, the tree initially chooses the feature that produces the best split in the data by maximizing the information gain:

$$Information\ gain = 1 - Entropy$$

$$Entropy = -\sum_{k=1}^{K} p_k log_2 p_k$$

where K is the number of categories. This is done recursively until the data is completely split. The random forest algorithm reduces overfitting that happens on normal decision trees by generating a number of decorrelated decision trees from random samples of the training set. Each tree then "votes" on the correct classification and the classification with the highest number of votes is chosen.

4.2.2 Support Vector Machines (SVMs)

SVMs work by creating a separating hyperplane to split data by maximizing the margin between the hyperplane and closest points from the various categories. We used a linear SVM as well as a Radial Basis Function (RBF) kernel:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{\sigma^2}\right)$$

4.3 CNN (Trained from Scratch)

We also employed the CIFAR-10 CNN architecture. The CIFAR-10 architecture was originally developed by Alex Krizhevsky to accurately categorize small images with 10 possible labels over his CIFAR-10 dataset. The final fully-connected layer with softmax optimization was modified to predict 8 labels. We trained the CIFAR network from a random initial state using our training data. The network was initialized with weights and biases normally distributed around zero. We explored a range of regularization techniques by adjusting the regularization strength and adding dropout layers.

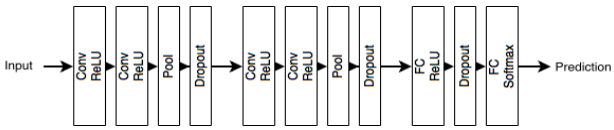

Figure 2. The CIFAR-10 architecture.

The categorical cross-entropy loss function

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) + \lambda R(W)$$

was minimized over the softmax activation of the final fully connected layer. During training, the loss value also includes the additional regularization terms penalizing the magnitudes of weights and biases. Network parameters were tuned using stochastic gradient descent with a decreasing learning rate. To accelerate loss minimization and improve convergence, Nesterov's accelerated momentum formulation was used.

Because of the small size of the available training data, we used data augmentation techniques to artificially increase the size of the training dataset. Ideally, the algorithms should be able to classify the fish species regardless of location, orientation, and size of the fish in the image. Images were randomly rotated, flipped, and translated during training to expand the dataset and prevent overfitting. This technique effectively increased the size of the training data by many times.

5. Results

5.1 Naïve Bayes

As expected, NB produced poor results of only 7% classification accuracy on the validation set. This was probably because there were many other colors in the images other than those of the fish, and because other factors such as shape were needed to distinguish between species.

5.2 Pre-trained CNNs

Upon running RF and SVMs on the data without tuning the algorithm parameters, it was evident that RF was performing better, producing around 80% validation accuracy compared to around 40% for SVMs. Hence we focused on tuning the parameters for RF.

We found that while changing the number of estimators and maximum depth of the tree did not affect the accuracy significantly, increasing the maximum number of features used affected the results significantly.

Eventually we settled on the following figures as optimal:

$$n_{estimators} = 15$$
$$\max \text{depth} = 40$$
$$\max \text{features} = 60$$

The results from pre-trained CNNs can be seen in Figure 3.

Model	Training accuracy	Testing accuracy
RF	99.5%	83.4%
SVM-RBF	44.3%	45.6%
SVM-Linear	54.1%	53.3%

Figure 3. Pre-trained CNN results.

5.3 Self-trained CNNs

CNNs trained from scratch were the most sophisticated learning algorithm we used and also had the most accurate results. We altered the hyperparameters of our model to see what model was most effective at learning how to classify the fish in the images we were provided. When we tested the CNNs, we split the data we were given, allocating 70% to the training set and 30% to the validation set.

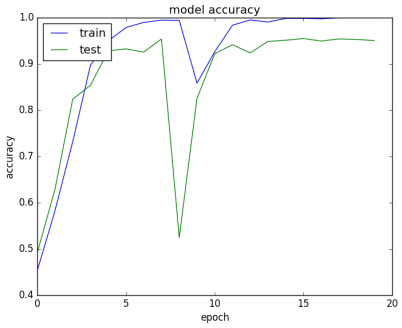

Figure 3a. No data augmentation, no dropout regularization, λ = 0.004.

One of the CNNs we ran directly took the training set images and learned how to classify them during 20 epochs of training (Figure 3a). This CNN, which had a regularization constant of 0.004 and did not use dropout, reached a validation accuracy of 95%, and a training accuracy of 99.96%. However, rather than spending most of our time improving this model, we decided to try a technique that could improve our generalization despite our small data set.

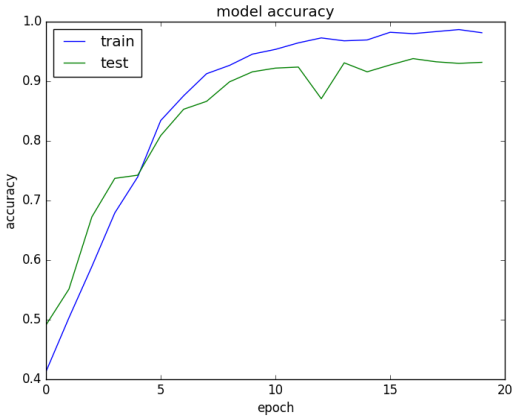

Figure 3b. Data augmentation, no dropout regularization, λ = 0.004.

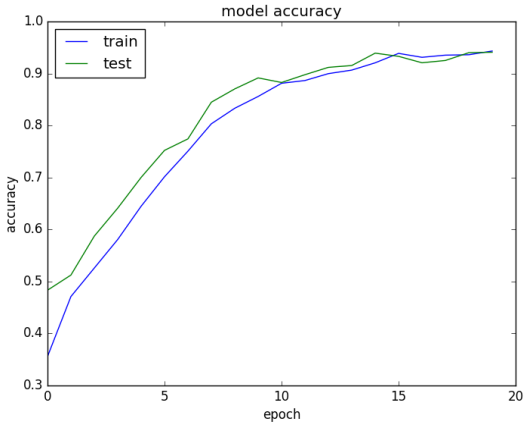

Figure 3c. Data augmentation, dropout regularization.

The rest of the CNNs we trained used data augmentation, a process in which the training images were randomly flipped horizontally and vertically in real time, so that the training set changed between epochs. We expected these results to appear worse, even on the validation set, because the images provided to us have similarities in the background that

the previous model was likely to take advantage of (see the conclusion below.) The data augmentation models were still fairly successful. When we trained the exact model (0.004 regularization, no dropout) with data augmentation, the CNN classified 98.15% of the training samples and 93.19% of the validation samples correctly (see Figure 3b). Adding dropout at three different points to this CNN gave 94.32% training accuracy and 94.09% validation accuracy (see Figure 3c). Dropout certainly reduced overfitting, but appears to have made it significantly more difficult for the model to learn how to classify the images.

6. Conclusion

Altogether, we were very pleased with the accuracy found by many of the algorithms we tried. The CNNs, in particular, were able to correctly classify almost all of the validation set images. The added complexity was evidently valuable in approaching a less standard image classification problem.

The problem of classifying fish species from images taken from a significant distance away is rather challenging. The vast majority of the pixels in the images are part of the background, which often includes fishing boats, the ocean, and people, among other objects present on the ships. Some of the images were taken while the people and fish in the pictures were moving, which makes them a little blurry. Additionally, due to computational limitations, we reduced the image quality before training our CNNs, making it even more difficult to distinguish the features of the fish.

We had to grapple with another challenge because of the data set we were using. Quickly glancing at the images contained in the training set provided to us reveals that there are only a handful of different fishing boats that appear in the images. Since the cameras taking the images are normally fixed in a certain position on the boats, the backgrounds of images taken on the same boat can be extremely similar. We tried the 1-nearest neighbor algorithm on the data to try to get a sense for how problematic this effect is, and it was able to correctly classify nearly 95% of the images correctly. Due to the small number of pixels occupied by the fish, this can only mean that the algorithm was typically able to classify the type of fish by comparing the boats in the background. That would suggest that the data set is somewhat flawed, as in the real world many different boats fish for the same species.

One way to get around this problem with the data is to learn how to find the fish in the pictures. This would be our next step in the project given more time. Finding the pixels that depict the fish would allow an algorithm to ignore the background pixels, which should provide no useful information. It would also allow us to not compress the images, thus preserving information that actually can be useful. The best algorithms for solving this problem would almost certainly need to locate the fish within each image.

Another future direction to take is increasing the training set size. CNNs that have the depth of CIFAR-10 are normally trained on millions of images (Alex Krizhevsky, 2012), while even with data augmentation, our training set was only tens of thousands of images large. Moreover, as Nature Conservancy transitions from using onboard cameras to using drone photography, adding images from drones to the training set would help the CNN become more accurate.

7. References

Alex Krizhevsky, I. S. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* , (pp. 1097-1105).

Karen Simonyan, A. Z. (2014, September 4). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs.CV]*.

Karpathy, A. (2016). *Module 2: Convolutional Neural Networks*. Retrieved from CS231n Convolutional Neural Networks for Visual Recognition: http://cs231n.github.io/

Krizhevsky, A. (2009, April 8). *Learning Multiple Layers of Features from Tiny Images*. Retrieved from University of Toronto Computer Science: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN Features off-the-shelf: an Astounding Baseline for Recognition. *eprint arXiv:1403.6382*.

The Economist. (2014, February 22). *The tragedy of the high seas*. Retrieved from The Economist: http://www.economist.com/news/leaders/21596942-new-management-needed-planets-most-important-common-resource-tragedy-high