Fish Classification Using Deep Learning

Zach Sturgill
Southern Connecticut State University
Computer Science Department
New Haven, USA
zachsturg@gmail.com

***Abstract*—In this paper, I present a method to classify fish species using convolutional neural networks. An automated system which can classify different types of fish can help biologists who seek a better understanding of fish behavior in the natural environment. Also, aquaculture farmers can benefit from the same automated system in that it could assist with monitoring the stock of fish being kept. This allows for far less manual work being done to monitor these video recordings. Several challenges exist in designing such a system. For instance, in a natural habitat, variations in the background image, lighting changes and water quality can vary substantially. The method proposed in this paper uses Convolutional Neural Networks (CNN), a type of deep learning, to classify fish images collected from the WildFish dataset [11]. The best performing model achieved an accuracy of 92.86%.**

***Index Terms*—Deep Learning, Convolutional Neural Networks, Fish Image Classification, Computer Vision**

I. Introduction

The use of remote cameras is an efficient method to monitor fish activity in the wild. However, the recordings must be reviewed by a professional in order to classify the types of fish, or the number of fish, found in the recordings which takes time. The use of an automated classification algorithm would an effective way to lessen the supervision needed in this process, and allow researchers to focus on only abnormal cases which the algorithm fails to detect. In addition, such an algorithm could allow for the use of additional remote cameras and therefore more data collection.

There are challenges in creating such an algorithm. Natural underwater envoronments are difficult locations to apply computer vision techniques. This is due to several factors. One of which is the wide variations in backgrounds: images of the same fish could have a rocky background, a coral bed background, or the open ocean. In addition, changes in lighting due to time of day or cloud cover results in a wide variety of differences within the images even when taken from the same location. Lastly, water quality can vary from clear to cloudy depending on the turbulence of the water.

The technique proposed in this paper uses CNN's to accomplish this task of fish classification. CNN's are a form of deep learning which use convolutional layers to extract the features which are then used to make predictions in a more traditional fully connected neural network. These types of image classification algorithms have proven to be very effective and are perhaps the most popular choice for image classification problems in modern times.

The rest of this paper is divided into the following sections: Section II. contains related works for both CNN architectures and fish classification; Section III. is a review of CNN's in general; Section IV. Discusses the dataset used; Section V. pertains to the CNN architecture implemented; Section VI. contains a review of the results found; and Section VII. holds the conclusion and future works.

II. Related Works

The focus of this research is to build and train a convolutional neural network (CNN) capable of classifying fish species in the wild. A task which can be useful in both biological studies, and the fishery industry. The following are other works related to this topic. Both research into CNN architectures and fish classification is considered.

A. Network Architecture

Leon Bottou [3] introduced stochastic gradient descent (SGD) in this paper. SGD is an optimization algorithm which improves upon gradient descent in many applications. Whereas gradient descent uses the entire dataset to compute the gradient, SGD approximates this gradient using only a portion of the dataset, and then iteratively moves through the remaining data. SGD allows for faster training on large datasets due to the fewer number of calculations to be performed.

Diederik Kingma and Jimmy Lei Ba [5] introduced the Adam algorithm which was shown to improve upon the classical stochastic gradient descent algorithm: instead of using a single fixed learning rate, Adam applies a learning rate to each network parameter, and adapts the learning rates during training. The learning rate is a hyper-parameter to the network which is used during the backpropogation calculations, and it determines the speed at which gradient descent occurs. This can either slow down or speed up training times, albeit with other possible effects. The use of an adaptable learning rate allows for faster training times when farther from the target, and then slower but more precise training when closer. Adam is a popular optimization algorithm used in deep learning.

Sergey Ioffe and Christian Szegedy [4] introduced the process of batch normalization for deep learning. Batch normalization involves normalizing the inputs of each layer within the network, and thus, making normalization a part of the model, as opposed to a process done to data before inputted into the model. The researchers showed that, by using a higher learning rate, the models using batch normalization could reach baseline levels of accuracy in shorter amounts of time when compared to previous models, and then continue to achieve higher accuracy overall.

Nitish Srivastava et al. [9] introduced the process of Dropout in deep learning models. This technique helps models to avoid overfitting by occasionally removing nodes and their connections during training. Previous methods to reduce overfitting mainly included ensemble learning, which would use the average from multiple models consisting of different architectures. Using Dropout allows a single model to take on many different architectures and produces similar results to ensemble learning.

Björn Barz and Joachim Denzler[2] experimented with training CNN's from scratch using small datasets. The research was performed using several small datasets ranging from 20 – 60 images per class, and CIFAR-100 with 500 images per class. The authors found that using a cosine loss function outperformed the popular cross entropy loss function on the smaller datasets but performed worse than cross entropy on the larger CIFAR-100 dataset.

B. *Fish Classification*

M. Sarigül and M. Avci [8] experimented with different CNN architectures trained on the QUT Fish dataset. The researchers examined how network depth and filter size of convolutional kernels effected the models results in terms of both accuracy and performance. Their results showed that a three convolutional layer network with large filters performed the best on test data when compared to 4 and 5-layer networks with varying filter sizes.

Vaneeda Allken et al. [1] used CNN's to classify fish trained through the use of synthetic data. Images of fish were cropped and manipulated onto artificial backgrounds to create a larger dataset. By increasing the size of the dataset in this way, they were able to improve the models accuracy, acheiving 94.1%. Considering the small size of the dataset used in this research, applying this method of synthetic data could provide useful in creating a more robust model.

Peiquin Zhuang et al. [11] created and released the Wild-Fish dataset consisting of 1,000 classes and 54,459 images. The authors also performed several experiments with deep learning using this dataset including open-set learning and deep learning with pairwise fish text. Using ResNet50 CNN model, pretrained on the ImageNet dataset, they achieved 78.2% accuracy using 685 classes from the WildFish dataset.

Dhruv Rathi et al. [6] used a CNN to classify fish species from the Fish4Knowledge image dataset. To further enhance their model, they also implemented several preprocessing techniques on the images: Otsu's algorithm was applied to perform background subtraction, and then erosion and dilation is applied to that mask, which assists in removing noise. The CNN was then trained with the original RBG image along with preprocessed image being the inputs. They further explored the use of different activation functions, including Rectified Linear Unit (ReLU), Softmax, and tanh functions. Their best model, using the ReLU activation function, was able to achieve 96.29% accuracy in classifying 21 different fish species.

Ahmad Salman et al. [7] worked on classifying fish species using the LifeCLEF14 an LifeCLEF15 fish datasets. To accomplish this, they initially trained a CNN, then removed the final fully connected layer, and used the features found the in convolutional layers to train K Nearest Neighbor and Support Vector Machine models. This came about from the idea that the earlier convolutional layers produced features that were very important to fish classification, and yet were ignored in the traditional CNN architecture. Testing was done on different combinations of the two datasets, i.e. training on LifeClef 14 and testing on LifeClef 15. Of the different models and data combinations, their best model, CNN-SVM, achieved 97.41% accuracy.

III. Convolutional Nerual Networks

An artificial neural network (ANN) is a machine learning algorithm in which some form of input data, normally a set of features, is fed through a series of connected layers made up of a number of neurons, before being outputted as a prediction of what the input was. Each prediction the network makes is compared with the actual value, and then the prediction error is used to adjust numerical weights with the neurons. Repeating this process multiple times to minimize the prediction errors is the process of training a ANN.

CNN's are a specific type of ANN which are used primarily for computer vision applications. CNN's differ from ANN's in that they take a raw image as input, as opposed to a set of predefined features. The CNN interprets the image as a matrix representing each pixel value. Next, the image is passed through several network layers. In a CNN, these layers mainly consist of convolutional layers, pooling layers, and finally fully connected layers. These types of layers are explained in further detail in the following subsections.

A. *Convolutional Layers*

For any type of machine learning model, for the model to learn, it requires certain features which represent the important attributes of the data being used for training and testing. In a CNN, these features are gathered using convolutional layers within the model itself. That is, a user is not required to define a set of features manually, instead the model itself decides which features are the most valuable.

The input to these convolutional layers is an image of size $n \cdot n \cdot c$ were n is the size of the image in pixels, and c is the number of channels; an RGB color image has 3 channels. The convolutional layers themselves consist of a number of filters, also know as kernels. These filters are also of a size $m \cdot m \cdot d$ where m is less than n, the size of the original input, and d is less than or equal to the number of channels. The filters pass over the image at a specified stride, and at each location they produce a feature map. These feature may be passed through several other convolutional layers before finally being used in the fully connected layers as inputs.

B. Max-Pooling Layers

An issue that arises due to the convolutional layers is an increase in parameters. If the number of features maps produced by a convolutional layer is greater than the number of input channels to that layer, than the number of parameters will grow quickly. To many parameters will cause very long training times due to the increased number of calculations required by the model. A common approach to solve this problem is to use max-pooling layers.

Max-pooling layers again use several filters of a size defined by the user. These filters are passed over the feature maps produced by the previous convolutional layer. At each filter location, the maximum is taken from the values found under the filters. This reduces the resolution of each feature map, and thus reduces the number of parameters within the network.

C. ReLU Acvtivation

In addition to max-pooling layers, the outputs from convolutional layers also pass through an activation function. Activation functions take the output of the layer and transforms the values before they enter the next layer. The purpose of such a function is to represent the activation of actual brain neurons. Several activation functions exist such as Hyperbolic Tangent function (tanh), Sigmoid, and Rectified Linear Unit (ReLU). Of these, ReLU has recently become a popular choice for CNN's and was found to outperform tanh and sigmoid functions for the purpose of fish classification [6].

D. Fully Connected Layers

Most CNN models contain a fully connected neural network as its final layers. These layers are specifically used to make predictions based off the features collected within the convolutional layers. The input to these layers is the output of the final convolutional or max-pooling layer. The number of outputs from the final layer is equal to the number of classes the model is attempting to predict.

IV. Dataset

The dataset used for this research comes from the WildFish dataset, which consists of 54,459 RGB images of 1,000 classes of different fish species. For the purposes of this research, only 4 classes were uses: Cephalopholis miniata, Dascyllus aruanus, Echeneis naucrates, and Genicanthus melanospilos. Of these classes, there are a total of 339 images. A split of 80% training data an 20% testing data was performed. Due to the small size of the dataset, both a random horizontal flip and random vertical flip where performed during training. The images were normalized before being inputted into the network. Examples of the data can be seen in figures 1, 2, 3, and 4.

V. Architecture

For the purposes of this research, several CNN models were created in Python using the Pytorch framework. Three models consisting of 3, 4, and 5 convolutional layers were trained and tested. [8] showed that the use of shallower networks is viable. Plus, a fewer number of layers decreases training time.

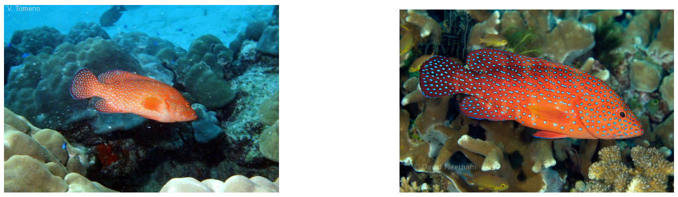
Fig. 1. Cephalopholis miniata

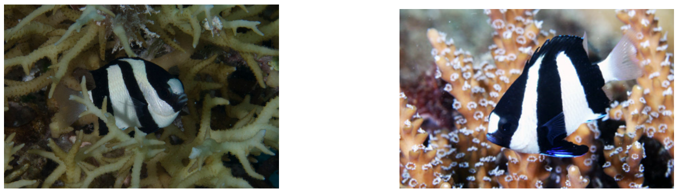
Fig. 2. Dascyllus aruanus

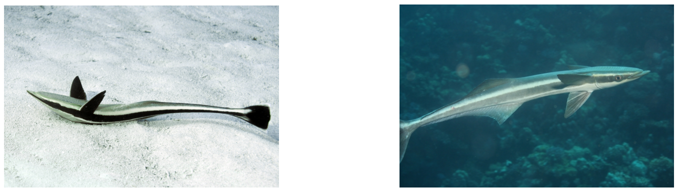
Fig. 3. Echeneis naucrates

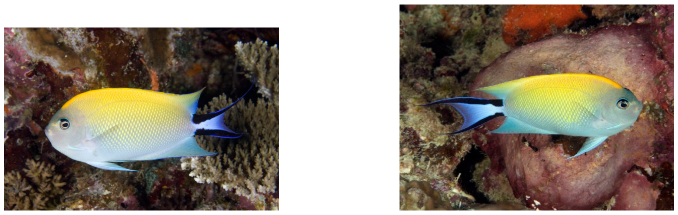
Fig. 4. Genicanthus melanospilos

The networks are trained on inputs of both 128x128x3 and 64x64x3 image sizes to compare differences. The following details the architecture of these model.

Each of the models has a similar structure of a series of convolutional layers followed by a ReLU activation function. Max-pooling is performed at certain intervals to reduce parameters. Variations to filter sizes are made for both convolutional filters and max-pooling filters depending on the number of layers and input size. Finally, each model ends with a two-layer fully connected neural network including a softmax classifier using cross-entropy loss.

Batch-normalization is implemented after each convolutional layer. Batch-normalization normalizes the outputs from each convolutional layer, rather than only normalizing the initial image. This has been shown to allow for higher learning rates, which decreases the training time, while maintaining state of the art results [4].

Stochastic gradient descent [3] was used as the learning algorithm for all models. Each model was also run with more advanced Adam algorithm [5], however, it was outperformed in all cases. Perhaps due to the limited training data.

The use of Dropout [9] was also implemented in the models. The dropout layer was implemented between the final two fully connected layers. The dropout probability was set to 0.5%. This too had a negative effect on all models, and was removed.

Each model was trained for 10 Epochs. Beyond this, there was signs of overfitting likely due to the limited amount of data per class. SGD was used for the training algorithm with a learning rate of 0.0001. The architectures of each network can be found in tables I, II, III.

A. Google Colaboratory

The training of a deep neural network is a very computationally heavy task and can take a long amount of time to complete. Using a graphics processing unit (GPU) can drastically speed up the computations and therefore training times. Google Colaboratory is a remote interactive python notebook which allows for the use of a Nvidia Tesla K80 for no charge. All models in this research were trained using Colaboratory GPUs.

TABLE I
5 CONVOLUTIONAL LAYERS

Layer No.	128x128 Input	64x64 Input
1	6x6 Convolution	6x6 Convolution
2	3x3 Convolution	3x3 Convolution
3	4x4 Max-pool	4x4 Max-pool
4	3x3 Convolution	3x3 Convolution
5	6x6 Convolution	6x6 Convolution
6	4x4 Max-pool	4x4 Max-pool
7	3x3 Convolution	3x3 Convolution
8	4x4 Max-pool	4x4 Max-pool
9	4096 Neuron Fully Connected	4096 Neuron Fully Connected
10	4 Neuron Softmax Output	4 Neuron Softmax Output

TABLE II
4 CONVOLUTIONAL LAYERS

Layer No.	**128x128 Input**	**64x64 Input**
1	6x6 Convolution	3x3 Convolution
2	4x4 Max-pool	2x2 Max-pool
3	3x3 Convolution	3x3 Convolution
4	4x4 Max-pool	2x2 Max-pool
5	3x3 Convolution	3x3 Convolution
6	4x4 Max-pool	2x2 Max-pool
7	6x6 Convolution	3x3 Convolution
8	4096 Neuron Fully Connected	4096 Neuron Fully Connected
9	4 Neuron Softmax Output	4 Neuron Softmax Output

VI. RESULTS

Testing of each model was performed on previously unseen testing data. For both input sizes, 128x128 and 64x64, the 4 convolutional layer model performed the best, achieving an overall testing accuracy of 92.86% and 90.00% respectively.

TABLE III
3 CONVOLUTIONAL LAYERS

Layer No.	**128x128 Input**	**64x64 Input**
1	5x5 Convolution	5x5 Convolution
2	6x6 Max-pool	6x6 Max-pool
3	5x5 Convolution	5x5 Convolution
4	6x6 Max-pool	6x6 Max-pool
5	5x5 Convolution	5x5 Convolution
6	4096 Neuron Fully Connected	4096 Neuron Fully Connected
7	4 Neuron Softmax Output	4 Neuron Softmax Output

However, all models performed relatively well, considering the small amount of data to train from. Evaluation metrics included training accuracy, testing accuracy, precision, and recall. The results from all models can be found in tables IV and V. The loss values generated by the models during training can be found in Fig. 5 and 6.

Due to the small size of the networks, training is completed very quickly. Utilizing Google Colaboratory free GPU service, which allows use of a Nvidia Tesla K80 GPU, training the networks was completed in a matter of minutes. The exact training times of each network is impossible to track using this remote service; variations of up to 20 seconds can be seen when training the network at different times. Nevertheless, the time results of each network is shown in fig. VI. This shows that the networks can be quickly trained on new data to allow for classification of different species.

TABLE IV
128X128 IMAGE SIZE RESULTS

Model	**Training Acc.**	**Testing Acc.**	**Precision**	**Recall**
5 conv	97.01%	91.43%	92%	91%
4 conv	92.54%	92.86%	93%	93%
3 conv	94.40%	91.43%	93%	91%

TABLE V
64X64 IMAGE SIZE RESULTS

Model	**Training Acc.**	**Testing Acc.**	**Precision**	**Recall**
5 conv	95.15%	87.14%	88%	87%
4 conv	88.80%	90.00%	91%	90%
3 conv	91.41%	87.14%	90%	87%

TABLE VI
TRAINING TIMES

Model	**Time (seconds)**
128x128 5 Conv.	73.38
128x128 4 Conv.	57.78
128x128 3 Conv.	53.05
64x64 5 Conv.	48.92
64x64 4 Conv.	47.60
64x64 3 Conv.	47.63

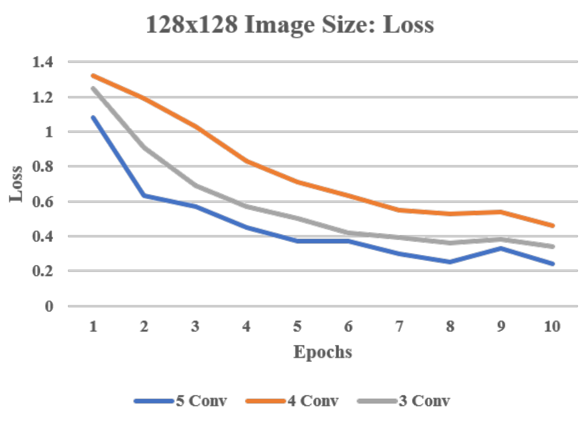

Fig. 5. 128x128 Training Loss

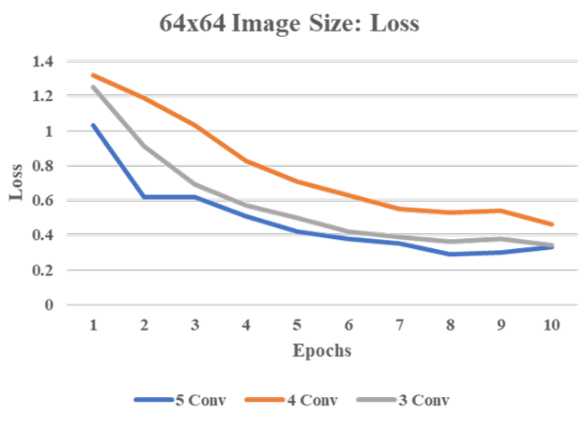

Fig. 6. 128x128 Training Loss

VII. Conclusion and Future Work

There exists a need by researchers and aquaculture farmers for a way to autonomisly monitor fish species in the wild. In this paper, I present a way to use convolutional neural networks to classify fish species. CNN models consisting of 3, 4 and 5 convolutional layers were trained and tested on both 128x128 and 64x64 input image sizes. The highest accuracy obtained, 92.86%, was through the 4 convolutional layer network with 128x128 input image size. These CNN models have been shown to be capable of being quickly trained on relatively small amounts of data, and yet still obtain high performance.

Further finetuning of the networks is always a possibility for future improvements. Experimenting with different loss functions for such a dataset, such as using a cosine loss function, which was found outperform the cross entropy loss function on smaller datasets [2].

To address the issue of this datasets size, work could be done to create synthetic data. The fish could be cropped from images, an inserted into a synthetic background consisting of varying shapes and textures such as in [10], [1]. This could allow for more capable models being created.

References

[1] Vaneeda Allken et al. “Fish species identification using a convolutional neural network trained on synthetic data”. In: *ICES Journal of Marine Science* 76 (Jan. 2019), pp. 342–349. DOI: 10.1093/icesjms/fsy147.

[2] Björn Barz and Joachim Denzler. “Deep Learning on Small Datasets without Pre-Training using Cosine Loss”. In: *CoRR* abs/1901.09054 (2019). arXiv: 1901.09054. URL: http://arxiv.org/abs/1901.09054.

[3] Léon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *Proc. of COMPSTAT* (Jan. 2010). DOI: 10.1007/978-3-7908-2604-3_16.

[4] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[5] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[6] Dhruv Rathi, Sushant Jain, and Dr. S. Indu. *Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning*. 2018. arXiv: 1805.10106 [cs.CV].

[7] Ahmad Salman et al. “Fish species classification in unconstrained underwater environments based on deep learning”. In: *Limnology and Oceanography: Methods* 14.9 (2016), pp. 570–585. DOI: 10.1002/lom3.10113. eprint: https://aslopubs.onlinelibrary.wiley.com/doi/pdf/10.1002/lom3.10113. URL: https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.1002/lom3.10113.

[8] Mehmet Sarigul. “Comparison of Different Deep Structures for Fish Classification”. In: *International Journal of Computer Theory and Engineering* 9 (Oct. 2017). DOI: 10.7763/IJCTE.2017.V9.1167.

[9] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[10] Jonathan Tremblay et al. “Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization”. In: *CoRR* abs/1804.06516 (2018). arXiv: 1804.06516. URL: http://arxiv.org/abs/1804.06516.

[11] Peiqin Zhuang, Yali Wang, and Yu Qiao. “WildFish: A Large Benchmark for Fish Recognition in the Wild”. In: *Proceedings of the 26th ACM International Conference on Multimedia*. MM ’18. Seoul, Republic of Korea: ACM, 2018, pp. 1301–1309. ISBN: 978-1-4503-5665-7. DOI: 10.1145/3240508.3240616. URL: http://doi.acm.org/10.1145/3240508.3240616.