Whale Identification Model based on Convolutional Neural Network

Ziyao Qiao qiao.z@husky.neu.edu, Yufei Gao gao.yu@husky.neu.edu
INFO 7390 Section 04 Spring 2018 Northeastern University

Abstract

In our final project, we aim to complete a competition held by Kaggle, utilize various convolutional neural networks to identify humpback whales from a series of photos of whales' tails.

Figure 1.1 Sample from our dataset

We reshaped pictures in the dataset with different size based on the model we use, and we have 2 different version: original photo and photo with highlighted tail. We applied different networks, including basic CNN, VGG19, ResNet50 and InceptionV3, on the dataset and cross those methods in some cases.
As the result, we found that VGG19 Model train with 'new-whale' class have the highest accuracy, but different models perform similar and there no obvious difference among them.
In this paper, we will introduce the background of deep learning as well as the competition at first. Then we will show related works that help us a lot in the second part of the report. After basic introduction, we will show the mechanism and implementation of our model. Finally, we will show the result of the project and draw a conclusion of our research.

1. Introduction

Under decades' development, Convolutional Neural Networks (CNNs) [2] have become one of the most popular approaches to solve computer vision problems, including object identification. Recently, this area developed rapidly because of two main reasons. Thanks to the rapid development of supervised learning, we could deploy various deep learning model on huge amounts of labeled photos. The wide-spread of advanced hardware such as GPUs also helps us to train a large neural network much faster than ever before.
Most of datasets in the competition comes from Happy Whale, a platform which already uses image process algorithms to category photos collected from public and various survey, and sorted by Kaggle. For now, the dataset contains one training set and one test set, and one csv file for labeling photos in the dataset.

2. Related works

Classifying breed of a specific whale is an example of a fine-grained image classification problem. Fine grained classification is challenging due to subtle differences among the images pertaining to each class. Previously, many researchers conduct research in this area, and they brought out many solutions. Those efforts which is being referred in our research project will be briefly discussed below:

2.1 Convolutional Neural Network

In 1966, Marvin Minsky at MIT asked his student to spend the summer linking a camera to the computer, and figure the disadvantage of Neural Network at that time. Due to the lack of advanced devices and computing ability, this research paused for a long time. However, in 1989, Yann LeCun created one of the very first

convolutional neural networks called LeNet[3] based on Back propagation theory, which could return feedback for model to improve. This designation had been used to picture recognition early and propelled the field of Deep Learning.

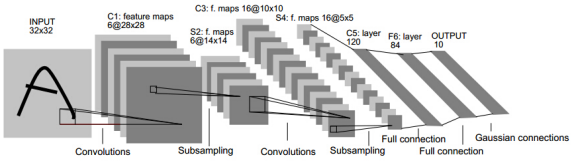


Figure 2.1 Architecture of LeNet

2.2 VGG-19[4]

In 2014, a team called Visual Geometry Group brought out the VGG Model, which contains deeper network and perform better on image identification problems than other models at that time.

Instead of using so many hyper parameters, all convolution layers in VGG-19 are 3 x 3 filter and 1 strides with same padding. Overall, VGG19 Model contains 16 convolutional layers and 3 pooling layers, and use "softmax" before return the result. The key that made VGG-19 successful is the depth of the network and the huge numbers of trainable parameters, which is about 138 million.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.2 Architecture of VGG Model

Like other VGG Model's features, VGG-19 also follows the pattern that when goes deeper into the network, the number of channel rises while the width and height of the volume reduces.

2.3 ResNet[5]

The revolution happened again in December 2015. ResNet have relatively simple ideas: feed the output of two successive convolutional layer, bypass the input to the next layers, improve and update information from residual value.

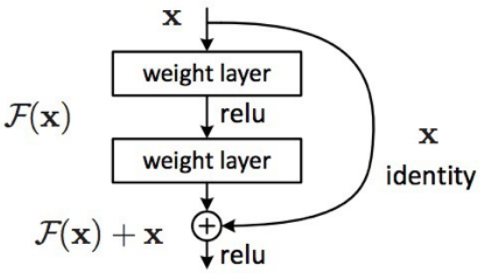


Figure 2.3 Basic Idea of ResNet Model

This is similar to older ideas like this one. But the new model bypass 2 layers and applied to large scales. Bypassing after 2 layers is a key intuition, as bypassing a single layer did not give much improvements. A 2-layers architecture can be thought as a small classifier, and the latter one inherits the result of the former one. This is also the very first time that a network has more than 1,000 layers was trained.

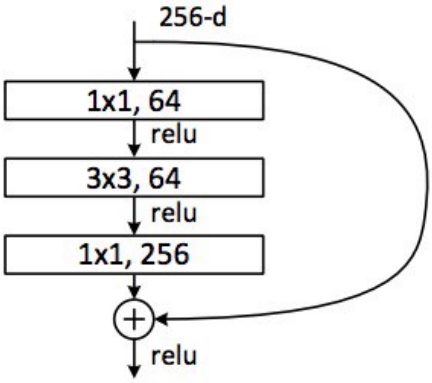


Figure 2.4 Residual block in ResNet50 Model

This layer reduces the number of features at each layer by first using a 1*1 convolution with a smaller output (usually 1/4 of the input), and then a 3*3 layer, and again a 1*1 convolution to a larger number of features. Like in the case of Inception modules, this allows to keep the computation low, while providing rich combination of features.

2.4 InceptionV3

In February 2015, close to the time ResNet created, Inception V2 was introduced to public for first time. Batch-normalization computes the mean and standard-deviation of all feature maps at the output of a layer, and normalizes their responses with these values.
This corresponds to 'whitening' the data, and thus making all the neural maps have responses in the same range, and with zero mean. This helps training as the next layer does not have to learn offsets in the input data, and can focus on how to best combine features.
Based on the principle, in December 2015, InceptionV3 had been created, with functions which could catch features more effectively.

3. Discussion

3.1 Lots of categories with limited training set

The training-set totally have 4251 categories, including 4250 unique and 1 ambiguous. However, we have only 9851 photos in the dataset, only 2-3 photo per category in average.

3.2 Unbalanced dataset distribution

As we described in midterm presentation, more than 3500 special ID in the dataset have less than 3 photos. And the "new_whale", one ambiguous category, have 810 photos. Those photos also make huge noise, so we would use specific balance methods to limit noise, and we also try remove them in some cases.

4. Implementation

4.1 Environment

We are running our models using GTX-1080ti with CUDA v9.0 and cuDNN 7.0. We are using Keras v2.1.3 and Tensorflow v1.8 as its backend. Most of our practice is made in Windows operating system and transferred the file path to fit Linux environment.

4.2 Preprocessing

Step1:
In the dataset, almost every photo has different size. In addition, some of them are black and white, but others are colored. Thus, we resize them before training. Besides, we also transfer all the photos with same channels to fit the models properly.

Step2:
Since tail is only a small part of photos, other objects such as sea-waves, ship and texts could make large noise in the training process. We designed an algorithm to try to highlight the tail area based on features of whale tail [6] to improve the accuracy. We enlarge the crack between different objects in the picture, find out those with large contour then figure out the edge of the shape with large contour angle.

Step3:
Since we have limited data in the training set, so we augment data for training and internal testing in our training process. We use the built in keras function to spin and reverse the photos, and try to figure out if the model catches the features correctly. If it returns a wrong answer, the residual could return some useful information to the model and help it to improve its performance.

4.3 Basic Convolutional Model

In the very beginning we constructed a CNN Model as baseline of the project. We cross validate the model with different elements including architecture, activation function, pool size, loss function, optimizer, batch size, epoch and step. Finally, we set up a model with relative high cost-interest ratio, with 3 convolutional layers, train 128 photos per step, iterate all photos per epoch and train for up to 100 epochs.

4.3.1 Network Architecture

Initially, our model has 3 convolutional layers, with several dropout layers, flatten layer and dense layer to reshape the data and catch features. We set batch-size as 128 and step value is total number of the training set divided by batch size, in this way we could make sure every photo in the dataset could be iterated once in a single epoch. Initially, we set the model run in small epochs like 10 to run in relative low costs. However, more epochs could be added if someone have better machine. In the end, we increase the epochs up to 100.
Besides, we change many different methods by using cross validation method. Compare the cost and accuracy, then find out the best one as part of our CNN model [12].

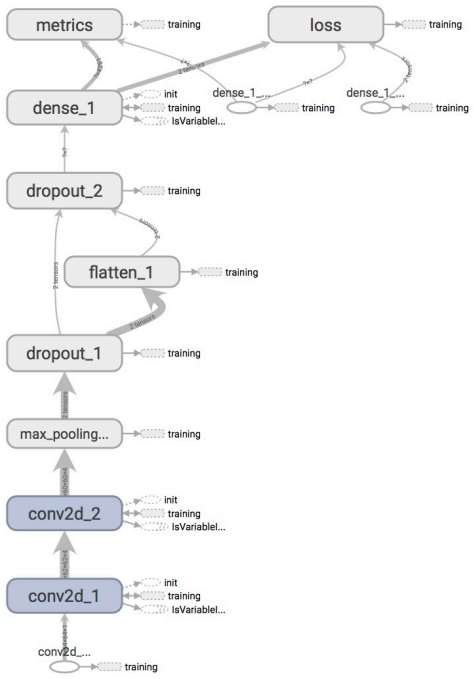


Figure 4.1 Basic Convolutional Model

4.3.2 Activation Function

Various activations functions are used for the convolution layers and the dense layers and the accuracy was calculated. We tried to change the activation function from Rectified Linear Unit (ReLU) and Sigmond to exponential linear unit (elu) and softplus. However, for this dataset, those two functions seem useless and the accuracy decrease drastically. Thus, we used ReLU and Sigmond as the activation in our CNN Model.

4.3.3 Cost Function

Just like activation function, we change the cost function from categorical-crossentropy to mean squared error. The substitution works bad for non-numerical categorical dataset, as it could not provide useful data for model to learn and improve, so the new function doesn't work well in our dataset. Hence, categorical-crossentropy used as the cost function for our model.

4.3.4 Kernel Initializer

Using various kernel initializers did not produce significant changes in the model accuracy or performance. Hence, we decided to use the default initializer to keep the performance of the model relatively good.

4.3.5 Optimizer Function

The model was trained with Adadelta Optimizer. We also tried Adamax and Momentum as optimizer function. The network plateaued normally when we used Adadelta Optimizer but overfitting quickly when we use others. At last we keep using Adadelta in our CNN Model.

4.3.6 Number of Epochs

Since in every single epoch of our model, all photos in our training set could be iterated for once, so we needn't consider epochs with the quantity of our dataset. At first, we just set our epoch as 1 to make sure our model could run normally, however, more epochs were added as we have better machine and in the end, we increase the epochs up to 100.

4.3.7 Learning rate

The model was trained with 0.001 as the training rate when we had 6 classes. But as the classes increased, the model took a lot of steps to plateau. Hence, we implemented decaying learning rate with the initial learning rate of 0.05 and the decay factor as 0.1, which decays every 50 epochs.

4.4 Pre-trained Models[10]

Keras have official VGG19[8]. ResNet50 and InceptionV3 implement [9], so we just applied those pretrained model to our dataset. As what we had done in the basic convolutional model before, we also need to augment data for training and internal testing in our pre-trained model training process.

At first, we tried ResNet50 Model. We set the input size as 224*224. Since the model is very complicated, we just set epochs as 10 so that we could get the result as quickly as possible.

Then we use InceptionV3 Model and keep the same input size. This model is a little easier with less layers, so we increase the epochs to 20. For this model, we set top 2 inception blocks in the architecture, hides some layer and leave others train, to figure out features more clearly and could be widely use in the dataset.

At last we tried VGG19 Model, and we change the input size as 299*299, corresponds to the default setting. This is the easiest pre-trained model we use for this project, thus we increase the epochs to 30, so that the model could receive more useful information as possible.

After the training process complete, we would save the trained model in .h5 format, then we use the well-trained model to a brand new test set from Kaggle, get the result and see the accuracy.

4.5 Code

https://github.com/ZiyaoQiao/INFO7390_FinalProject

The above is the link to the GitHub repository of this project containing all the code and documentation about this project.

https://github.com/ZiyaoQiao/INFO7390_FinalProject/blob/master/README.md

Referring to this portfolio will help better understand the structure and whole process this project.

5.Results

Generally speaking the result not correspond to our expection, this result lead by many reasons, but it's mainly because the unbalanced number of each class in the dataset and the low quality of photos. The detail of our analysis will be listed below.

5.1 Preprocessing Stage

5.1.1 Algorithms concerning detecting contour area doesn't work well

Our program aiming to figure out the tail basically based on "THRESH_BINARY" and "CANNY" function in cv2 package, two packages which could find angle and figure out the edge of the shape in the picture. However, we underestimated the noise made by sea waves and other contents in the pictures. In some photo which is relatively clean, the algorithm performs well. But in others with various noises, this program seems useless.

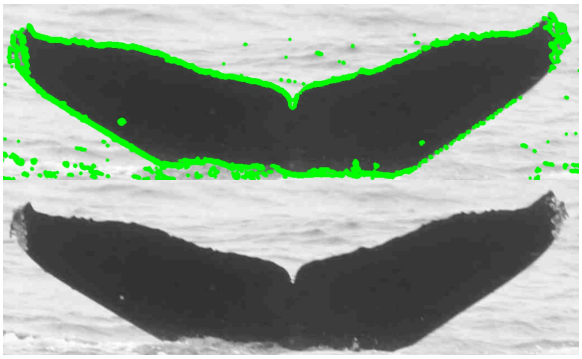


Figure 5.1 Success sample of contour-detect

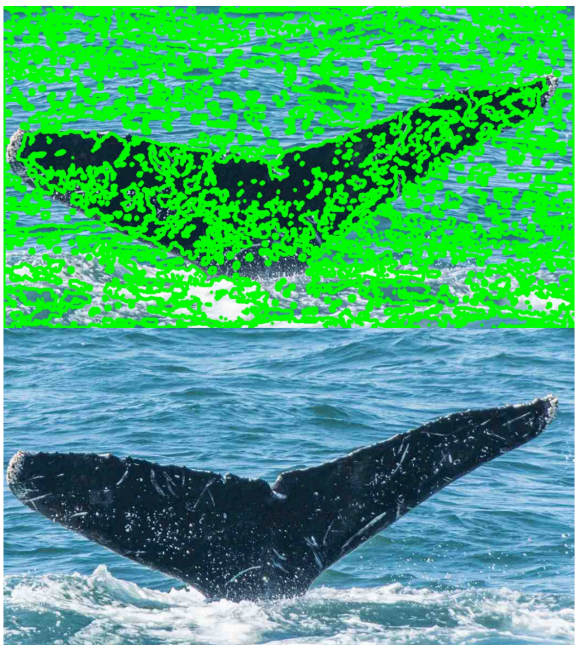


Figure 5.2 Failure example because of unexpected angles from spots in tail and waves

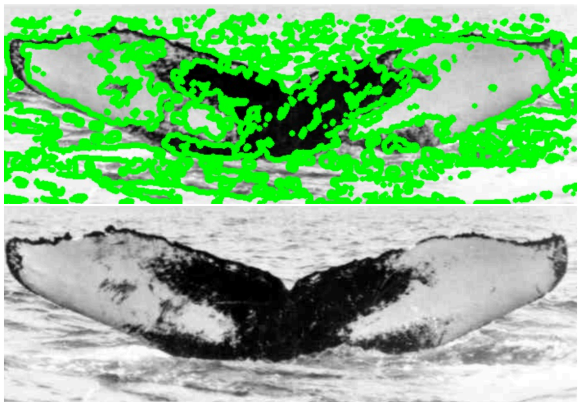


Figure 5.3 Another failure example generated by waves

5.1.2 Computing ability limited by GPU-Memory

Besides the model we choose and methods we used to tackle with photos based on our devices, computation power is also an important part of performances in such competitions. Since lack of advanced machines and computing ability of large models, we didn't train those models systematically, and we also unable to try any other aggregated models or tackle with photos with other advanced algorithms, which limited our performance. Furthermore, the average utility of our GPU(GTX-1080ti) always keep 20%-30%, which seldom fully used. As we analyzed, the bottleneck is the GPU Memory. The GPU could calculate a large batch of photo in one time but GPU-Memory limited the data stream [11].

Another point is that we could design some better algorithms in data augmentation. As we see, the test set in training process seems not work very well for the result, since we just rotate the photos in the training set simply and shift the size, which seems helpless for the model to predict photo from another dataset.

5.2 Models in the project and dataset

In conclusion, the basic CNN model seems too easy for the dataset and it could not

figure out too many features from the training set. Besides, since many class in the dataset have only one or two pictures, so the weight of features from single photo could be really large and have negative influence to the result.
As for pretrained model, VGG19 seems work best for this project. That's mainly because in the dataset, most of classes have only one or two photos, so a model with too many layers may work worse than a relative simple model. The architecture of VGG19 probably fits the dataset and could catch features better than other two models.

Method	Accuracy in test set	Epochs	Average time (s/per epoch)
ResNet50 without 'NewWhale'	32.482%	10	12478s
ResNet50 with 'NewWhale'	32.631%	10	12600s
VGG19 with 'NewWhale'	32.999%	30	2270s
inceptionV3 without 'NewWhale'	32.481%	20	4703s
CNN Model with contour detected	32.763%	100	215s
CNN Model without contour detected	32.875%	100	214s

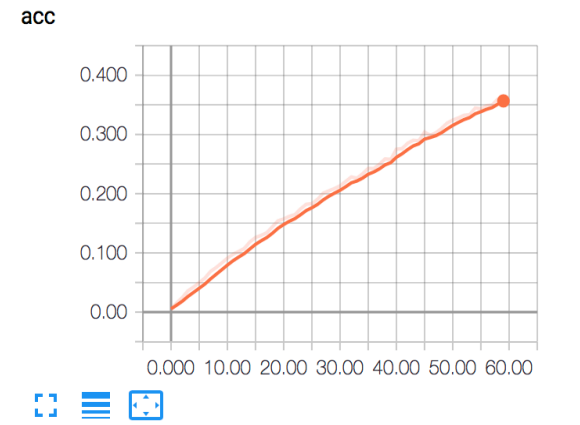
Figure 5.4 Overall result of different models in our projects

However, even different models we implemented have tiny difference, totally they performed similar for this dataset and the score in the test set not have large difference. As we summarized, that's mainly because of the low quality of the dataset: nearly every picture has different size, some photos are colorful but others not, and the photos obtained from different direction, different distance and environment, which is also considered and captured as a part of features by the models.
All implementation concerning the project could be found on our GitHub repo https://github.com/ZiyaoQiao/INFO7390_FinalProject

Reference

[1] https://www.kaggle.com/c/whale-categorization-playground
[2] https://www.coursera.org/learn/neural-networks-deep-learning
[3] https://github.com/BVLC/caffe/blob/master/examples/mnist/lenet_train_test.prototxt
[4] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[5] https://arxiv.org/abs/1512.03385
[6] https://en.wikipedia.org/wiki/Humpback_whale
[7] https://cs231n.github.io
[8] https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8
[9] https://www.learnopencv.com/keras-tutorial-transfer-learning-using-pre-trained-models/
[10] https://www.tensorflow.org/tutorials/image_retraining
[11] https://github.com/spejss/Keras-Transfer-Learning-Tutorial
[12] https://www.kaggle.com/gimunu/data-augmentation-with-keras-into-cnn

APPENDIX

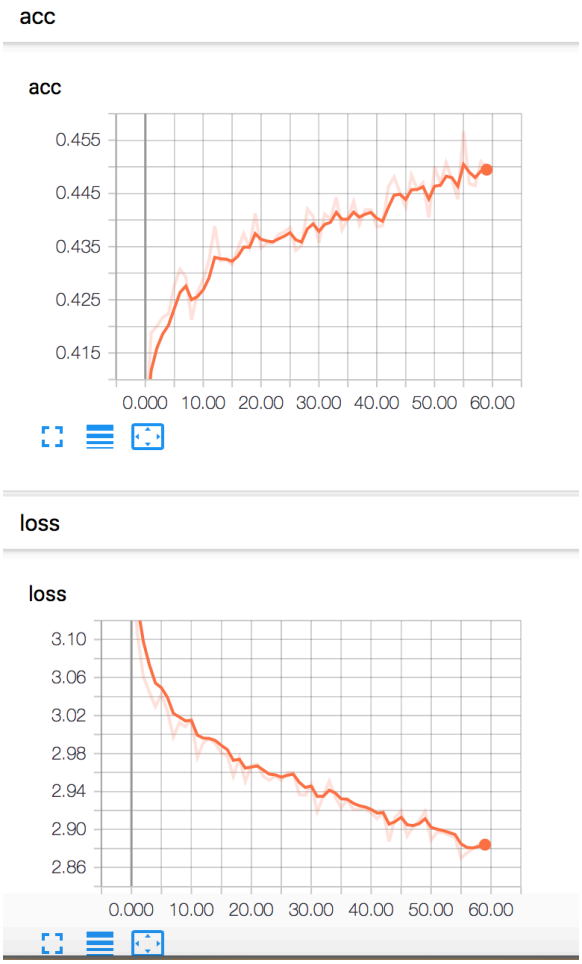


performance of InceptionV3 in training set in one epoch

The horizontal axis is the steps of training and vertical axis is accuracy and loss in two graphs.
This model takes almost 1h20min to finish one epoch's training, the loss drop and the accuracy increase stably, but the highest accuracy (the accuracy in final step) still relatively low.

performance of VGG19 in training set in one epoch

The horizontal axis is the steps of training and vertical axis is accuracy and loss in two graphs.
This model takes about 40 minutes for one epoch. It is relatively simple, so it seems perform worse than InceptionV3 in the training set. But in the test set, it would benefit from this feature because many classes in the dataset have a few samples.



performance of ResNet50 in training set in one epoch

The horizontal axis is the steps of training and vertical axis is accuracy and loss in two graphs.
This model performs best in the training process, it also takes the longest time – more than 3 hours per epoch. However, it seems extreme sensitive to loss and it volatile constantly, thus it would be influenced by irrelevant noises and not performs well in the test set.