The Nature Conservancy: Fisheries Monitoring

Abhinav C, Abhinav M, Amit K, Atit S, Samir J, Sudipto B
Department of Computer Science
NC State University
Raleigh, US
{achoudh3, amedhek, akanwar2, akshetty, sjha4, sbiswas4}@ncsu.edu

Abstract:

This report deals with several techniques for classification of images containing fishes from 8 categories. The first technique is called as SIFT that extracts local features from images. The second technique is called as Transfer Learning using Convolutional Neural Networks. We have tried two CNN models: VGG and InceptionV3. We observed better results with InceptionV3. To improve the results further, we used 2 techniques: Bounding Box and Gradient Boosting Decision Trees. Our best result is a log loss of 1.879 using an ensemble of InceptionV3 and GBDT.

Keywords- convolutional neural net; deep learning; image classification; SIFT; gradient boosting; transfer learning; kaggle

I. INTRODUCTION

The Nature Conservancy is looking to the future by using automated learning algorithms to dramatically scale the monitoring of fishing activities and to fill critical science and compliance monitoring data gaps.

The input to our model are the images of fishes caught on cameras installed on boats. We have to classify these images into 8 categories viz. Albacore Tuna, Bigeye Tuna, Yellowfin Tuna, Mahi Mahi, Opah, Sharks, Other (fishes not belonging to previous categories) and No Fish(implying that the image consists of no fish). The main challenges in the identification are low quality images and size of interested feature (i.e. fish) in the image.

In our experiments, we have evaluated several techniques to accomplish the task. The first technique is called SIFT (Scale-invariant Feature Transform), which can detect features of interest (viz. fishes) in an image. The second technique is called Transfer Learning using CNN (Convolutional Neural Networks). We will use "pre-trained" CNN models, and modify them to detect features of interest in our data.

In the end, we will compare the results obtained from all the techniques, that will help us find best model to accomplish our task.

II. RELEVANT WORK

SIFT

Scalar Invariant Feature Transform (SIFT) is a feature extraction technique that provides us a vector of features which are unaffected by scaling, rotation or any other manipulation of the object. It is quite similar to how the human vision works.

SIFT employs a 4-stage filtering approach:-

1. Scale-Space Extrema Detection: This is employed to create the feature vector from all the critical points in the image which are usually edges or points of high contrast. To achieve this, we use a scale space function based on Laplace of Gaussian. Since Laplace of Gaussian is a difficult problem to solve, we employ difference of Gaussians to get the local minima and maxima (the keypoints we need). Each point is compared with 8 neighbors at same scale, 9 above and 9 below. Max or min value defines our critical points.
2. Key-Point Localization: The extrema detection will have all the edges which might not be needed in the final creation of features vector. This stage removes all the unwanted edges on the criteria of their intensity being lower than a particular threshold.
3. Orientation assignment: A histogram of 36 bins is created for including almost all the directions in 360 degrees. The maxima and points within 80% of the the maxima are used to decide the direction of the kepypoint. This makes SIFT resilient to orientation or rotational manipulations in the images[12].
4. A set of 16 histograms in 4x4 grid, for 8 directions with Gaussian weighing around center. This creates a 4x4x8 = 128 dimensional feature vector.

Convolutional Neural Networks (CNN)

CNNs are currently the state of the art for image classification models. We went through the recent high ranking submissions on Kaggle for image classification projects and saw that most of them use CNN models.

Layers:
Traditionally, a CNN model comprises of 3 primary types of layers:
(i) Convolutional Layer:
This layer does most of the computationally heavy work associated with CNNs. This layer uses a filter with defined weights to make a forward pass on the input in predefined strides.
(ii) Pooling layer:
Pooling layers are responsible for decreasing the size of the convolution output, by taking small blocks of pixels and compressing them to one pixel by averaging, taking max or min of the values in the block. Pooling layers do not have any associated weights and nothing is learned in these layers.
(iii) Fully Connected Layer:
This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from.

Parameters:
Activations: Activation functions are used to transform the activation level of a unit (neuron) into an output signal. We went through multiple papers which had a common conclusion that ReLu is currently the best[1] and the fastest[2] activation function.
Dropout: Dropout is an efficient technique to mitigate the risk of overfitting of CNN models by randomly dropping units (along with their connections) from the neural network during training. This prevents units from co-adapting too much.
Loss Function: We have used the *Cross-entropy error function* or *log loss* used to evaluate Kaggle submissions.
Log Loss is given by: $\mathbf{l(y,p) = -ylog(p) + (y-1)log(1-p)}$
Average log loss for N instances is $\mathbf{1/N \sum_i l(yi,pi)}$ (where y is the goal value and p is a predicted value)

Optimization using Stochastic Gradient Descent:
Several researches show that the best direction to change our weight vector is mathematically guaranteed to be the direction of the steepest descent of our loss function. Gradient Descent algorithm iteratively computes the gradient and performs a parameter update in the loop. SGD is a stochastic approximation of the gradient descent optimization method. There are several hyperparameters used to optimize this algorithm.
Learning Rate: Learning rate is roughly equivalent to the step size in the direction of gradient. Choosing this parameter carefully is important as a low step size makes progress very slow while a larger step size makes loss high due to overstepping.
Decay: Decay is a technique used for annealing the learning rate. This is adopted because slower learning rates tend to converge faster. Therefore, we have used step decay to gradually slow down the learning rate towards the end.
Momentum**:** This is a handy technique which is used for faster convergence of gradient descent algorithm. With Momentum update, the parameter vector will build up velocity in any direction that has consistent gradient favoring convergence in that direction.

VGG
The beauty of VGG lies in it's simplicty. It uses the insight that multiple 3*3 convolution layers stacked together can replicate relatively large receptive fields.

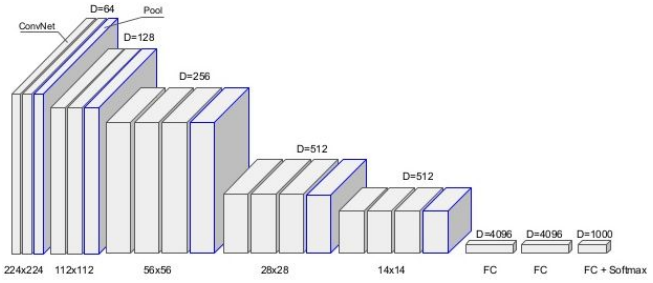


Figure: VGG Architecture

VGG stack consists of layers of convolutional layers with 3*3 filters. The stride is fixed to 1 and spatial padding is used to preserve input resolution after convolution. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers. Max-pooling is performed over a 2 × 2 pixel window, with stride 2.

Inception V3
The goal of the Inception module is to act as a "multi-level feature extractor" by computing 1×1, 3×3, and 5×5 convolutions within the same module of the network — the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network.

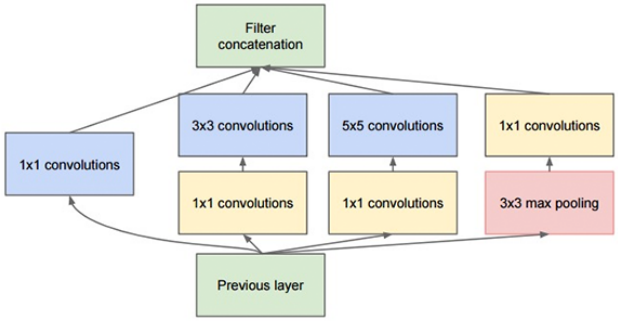


Figure: Inception Module

This architecture allows increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity. The ubiquitous use of

dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously.

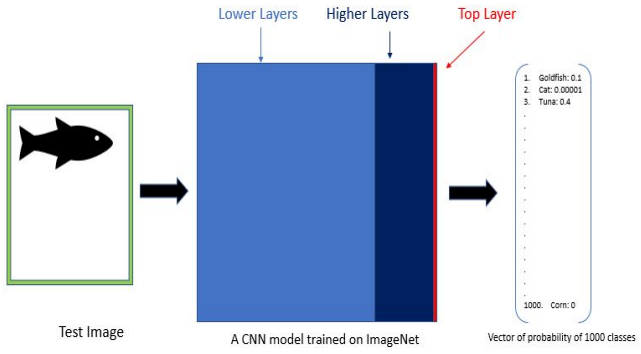
Transfer Learning

Deep learning frameworks like Keras, have inbuilt CNN models like VGG16, VGG19 and InceptionV3. We can use these models and train them on our data with random initializations. However, we seldom have large enough dataset to properly train such models, and the resulting model can have poor performance.

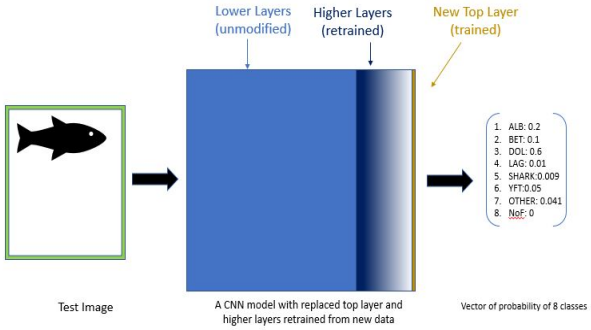
A solution to this problem is using pre trained models. These models have been trained on the large set of images available in databases like Imagenet and their weights have been saved for anyone to use. However, we cannot use these models as it is for our purpose. This is because the number and type of classes in our problem is very different from those used in Imagenet. We can use these models with some modifications to be trained on our dataset. This is known as transfer learning.

We can use any pre trained model in two ways:

a) Remove the top layer which is classifier layer and replace it with a linear classifier like *softmax*. We then train only this layer with our training data. Thus, we use the pre trained CNN model as feature extractor.



b) Instead of just training the last linear layer, we can train either the entire model or some top layers. This process is called fine tuning. The rationale behind this it is that the lower layers are used to detect basic features (like edges), and as we go higher, we detect complex features (like spirals). Thus, a CNN model detects more specific feature as we go from lower to higher layers.



III. Methodology and Implementations

SIFT with KNN

1. OpenCV was used to implement SIFT.
2. OpenCV has direct function calls to SIFT method which gives us a 128-dimensional vector.
3. We trained a KNN model on basis of feature vectors we got from training data (using SIFT).
4. This model gives the probability distribution of likelihood of image belonging to particular species.

Transfer Learning using CNNs

(a) The first step before we actually begin training and testing our model is to create training and validation data sets. Training records supplied by Kaggle has uneven distribution of classes. We need to create validation sets that have proportionate representation of each class. This can be accomplished using StratifiedShuffleSplit method from python package sklearn. We compared k-fold cross-validation split with StratifiedShuffleSplit method. We got improved results with k-fold, however, this iterative implementation is considerably slower than the package method. We have used StratifiedShuffleSplit method in our InceptionV3 model and k-flod split in our VGG models

(b) The second step is creating a new model that was pre trained on Imagenet weights. We have used the second Transfer Learning approach with InceptionV3 and VGG henceforth. We will remove the top layer and add few other layers as mentioned in the previous section. We have added different learning layers on top of the pre-trained models. The Dropout layer is used to control overfitting. GlobalAveragePooling layer is used to reduce the dimensions of output. Dense layer with Relu activation will add non-linearity.The last layer is a dense layer that uses softmax activation to generate 8 outputs, signifying probability of each class for an image. We then train only these added layers on our training data using SGD

optimizer. We trained the models with varying hyperparameters and found best accuracy and training time tradeoff with dropout of 0.5 and learning rate: 1e-2, decay: 1e-3 and momentum in the range of 0.9.

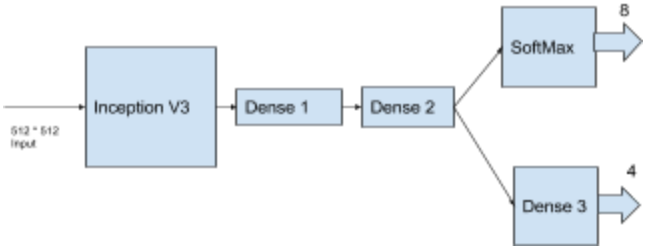
(c) The third step is fine-tuning. We train layers from 171 and above for InceptionV3, 20 and above for VGG19 and 16 and above for VGG16, keeping all the lower layers as untrainable. This helps us fit the model to our data set, increasing the performance of our model. We save the weights after this step.

(d) The last step is predicting the classes for test dataset. We load our model, with the weights saved after fine-tuning step. Now using this model, we can classify our test data, and save the output in a CSV.

A comparative analysis of results showed that InceptionV3 outperformed both the VGG models on loss and accuracy metrics. Also, due to its depth and number of fully-connected nodes, VGG is painfully slow to train.

Next, we use different strategies to improve the accuracy of InceptionV3 model further.

InceptionV3 with BoundingBox



In this part, we use training data which has been annotated with bounding boxes. These box locations are provided to us in the form of JSON files for each of the image. We now train our network to identify the type of fish and also to locate the fish in the image. Thus we have a two output network. The bounding box output is a normal dense layer output that predicts the bounding box location, while the class output is a Softmax output. Both outputs are trained using separate loss functions. The use of bounding boxes tends to focus the network on the the most significant part of the image .i.e the fish and thus helps in extracting significant features. This split of the CNN model into 2 outputs happens at the very last stage, so that we can ensure that the two outputs are dependent on each other.

InceptionV3 With GBDT

Gradient boosted decision trees (GBDT) is an ensemble classification approach based on the idea of boosting. GBDT works by growing an ensemble of decision trees, where the first tree is grown like a normal decision tree and subsequent trees are grown to fit the residuals from the tree before it. So each tree tries to 'rectify' the errors made by the previous tree.

We combine our InceptionV3 model with GBDT to improve the predictions. In order to achieve this in step d of previous section, we will drop the last two dense layers of our InceptionV3 model. The new output from inner layer is a 2048 vector called embeddings. Embeddings from inner layers of a deep net can form a representation of images that is almost linearly class separable.

We capture embeddings for all the training, validation and test images and save them in train.csv, validation.csv and output.csv respectively. These CSVs are then fed to GBM. The GBM has predefined number of trees of some max depth. We use StratifiedKFold method to train and grow the forest. We then predict the 8 classes, for all the test data in output.csv.

IV. Experiments and Results

Data Description

Training data consisted of a total of 3777 images. The validation set of 500 images sampled from the training data using Stratified split was used for hyperparameter tuning.

The test data consisted of 13513 images provided by Kaggle. Class distribution is not disclosed for this test data, but rather a final log-loss score is shown based on the output uploaded.

Programming Language

Python was chosen as the de facto language for implementation due to large support for deep learning libraries like Keras and availability of scientific packages like numpy, sklearn, etc.

Setup

SIFT

1. OpenCV was used to implement SIFT.

CNN

1. Deep learning library Keras employing Theano backend was used to generate neural networks.
2. In order to increase speed of training and testing of NNs, we used GPU support on Nvidia GeForce GTX TitanX in NC State ARC cluster.
3. The GPU has 12GB of memory with 3072 CUDA cores and 1000MHz base clock.
4. Microsoft's LightGBM gradient boosting machine to increase classification accuracy.

Data setup

1. In order to train CNN model, we created a stratified split consisting of 500 validation records from training records.
2. The rest were used for training.
3. All the images will be scaled to 512x512, before being fed to CNN model.

Parameters

VGG16 FCN

1. Layers added in sequence:
   Dropout(0.5)
   Conv2D(256,(3,3),activation='relu')
   Conv2D(8,(3,3),activation='relu')
   GlobalAveragePooling2D()
   Activation('softmax')
2. Optimizer = SGD(learning rate : 0.001)
3. Batch Size = 64

VGG19

1. Initialization: Pooling='avg'
2. Layers added in sequence:
   Dropout(0.5)
   Dense(1024, activation='relu')
   Dropout(0.5)
   Dense(1024, activation='relu')
   Dense(8, activation='softmax')
3. Optimizer: SGD
   (learningrate=1e-3,decay=1e-4,momentum=0.9)
4. Batch size = 64

InceptionV3

1. Layers added in sequence:
   Dropout(0.5)
   GlobalAveragePooling2D()
   Dense(1024, activation = 'relu')
   Dense(8, activation = 'softmax')
2. Optimizer = SGD (learning rate : 0.01)
3. Batch Size = 64

GBM

1. boosting_type = "gbdt"
2. num_trees = 200
3. learning_rate = 0.01
4. max_depth = 5
5. num_leaves = 60
6. colsample_by_tree = 0.8
7. validation split = 25%

Bounding Boxes

Input:
1. JSON File: X, Y, Height, Width

Output:
1. Bounding Box Output: Loss function = MSE
2. Class Output: Loss function = Categorical Cross Entropy

Results

The accuracies obtained for the various stand-alone CNN models was as following:

Model	Validation-set accuracy
InceptionV3	**0.962**
VGG19	0.91
VGG16-FCN	0.84

The following results were based on log-loss used by Kaggle. The function used is Categorical Cross-entropy.

Approach	Log-loss
SIFT + KNN	1.93562
InceptionV3 TL + GBM	**1.87961**
InceptionV3 TL + BB	2.4

It can be seen that transfer learning with InceptionV3 combined with GBM outperformed all other approaches.

- SIFT was not able to achieve lower log-loss score due to presence of low-contrast images in the testing data.
- Stand-alone VGG models achieved lower acurracy than the deeper InceptionV3 model.
- InceptionV3 with bounding boxes was able to generate an acuracy of more than 97 % on our validation set (better than with only Inception), however it did badly on the Kaggle's test data set. We believe the reason for this is that the model suffered from overfitting on the training data . The images in test data had new boats and new locations of fishes and the over trainig prevented the model from detecting such changes.
- It is also important to note the time for training and predictions.
- Since, SIFT compares the test image with all the vectors obtained from training data, it takes a long time predict all the images.
- Training a CNN model takes considerable time, as we have to run many epochs to achieve desired log-loss.
- However, predicting an image is much faster in CNN.

V. Conclusions and future directions

It is evident that CNNs are better at classification task for images. Even a vanilla model of CNN pre trained on ImageNet can give above average accuracy. Due to presence of deep learning library like Keras, it is easy to implement models for image classification. There is also very less pre-processing required. CNNs combined with other classification techniques have the potential to generate state of the art models that can excel at image classification.

However, it is evident from our experience that there is still a lot of work that can be done. We can use SURF to improve accuracy for prediction of low contrast images. CNNs can be trained to improve their performance in localization task, eg. bounding boxes.

VI. Acknowledgment

This project would not have been possible without our learning from the course Automated Learning and Data Analysis taught by Prof. Ranga Raju Vatsavai. We appreciate the continuous support and feedback of Prof. Ranga Raju Vatsavai and the course TAs throughout the timeline of the project.

VII. References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet classification with deep convolutional neural networks" NIPS, 2012

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, S. Jonathon, and Z. Wojna. "Rethinking the inception architecture for computer vision." arXiv preprint arXiv:1512.00567, 2015

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. "Going deeper with convolutions", CoRR, abs/1409.4842, 2014.

[4] Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. CoRR, abs/1411.4038, 2014

[5] Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. "CNN Features off-the-shelf: an Astounding Baseline for Recognition", CoRR, abs/1403.6382, 2014.

[6] Karpathy, A., Johnson, J, Li, F., Convolutional Neural networks for visual recognition, Stanford university course CS 231

[7] Szegedy, C., Vanhoucke, V., Iofe, S., Alemi, A., Inception-v4,Inception-ResNet and the Impact of Residual Connections on Learning

[8] Szegedy, C., Wei, L., Yangqing, J., Sermanet, P., Reed S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., Going deeper with convolutions

[9] Rosebrock, A., ,ImageNet: VGGNet, ResNet, Inception, and Xception with Keras

[10] Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In International conference on machine learning.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, pages 1929–1958, 2014.

[12]http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MURRAY/SIFT.html

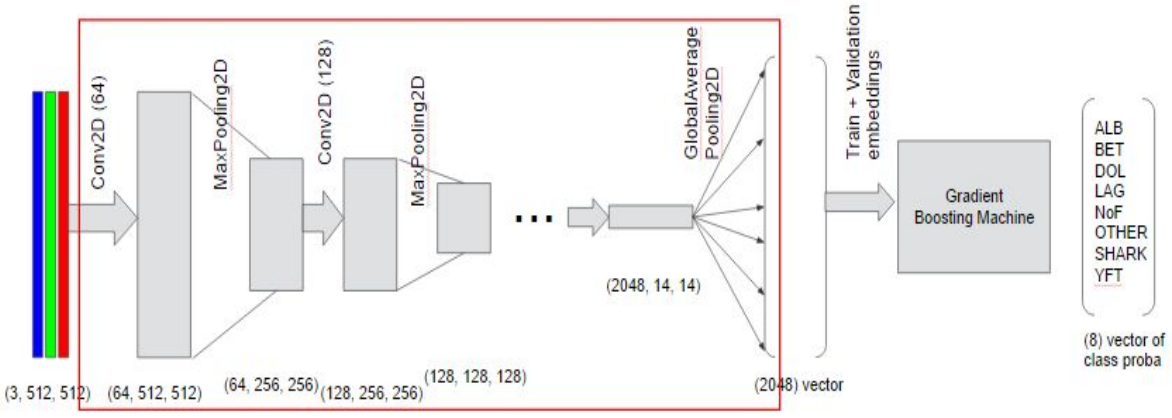


Figure: Representational diagram (simplified) for our model with InceptionV3 and GBM