Insect Classification With Heirarchical Deep Convolutional Neural Networks

Convolutional Neural Networks for Visual Recognition (CS231N) , Stanford University
FINAL REPORT, Team ID: 283
13 March 2016

Jeffrey Glick
MS Candidate, Management Science & Engineering
jdglick@stanford.edu

Katarina Miller
MS Candidate, Electrical Engineering
kmiller3@stanford.edu

Abstract

Classification of insect species is is a particularly difficult challenge because of the high degree of similarity of the appearance between distinct species. Given the visual commonalities across different levels of taxonomic hierarchies (i.e. class, order, family, etc.), insect classifcation provides a unique opportunity to apply emerging techniques used in Hierarchical Deep CNNs (HD-CNNs). Inspired by the success of recent work on HD-CNNs, our pipeline for designing, training and testing a hierarchical architecture achieves promising results considering the particularly messy nature of our data set. Working with 217,657 insect images from 277 unique classes, we achieved a top-5 misclassification rate of 22.54% *and a top-10 misclassification rate of* 14.01%

1. Introduction

There are approximately 1 million unique insects on Earth, making up 80% of all known species [4]. Insects are ubiquitous in and around human dwellings, farms and throughout nature. Certain insects are critical to the health of delicate ecosystems. Others are vectors for deadly diseases. There are numerous public health, safety, economic, educational and other practical benefits to a hypothetical mobile application which would allow someone to quickly and accurately classify an insect from the field using only their smart phone camera and a connection to the Internet.

We propose that a user could take a picture of an unknown species with her smart phone and then upload the image in a simple UI. [1] The user provided data would then be sent into the cloud to be passed through a pretrained hierarchically structured CNN. The application UI would then return images and names of the the top k likely predicted species along with a a few example photos of those top k insects species. The user would then be able to visually compare the target species in front of her to the k images to help positively identify the insect. At a minimum, the results would help to point the user in the right direction for further research.

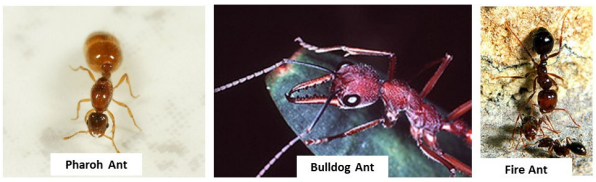

Figure 1. Example images from three distinct classes

This research explores the technical feasibility of classifying highly varied images of insects from a wide range of species. Once designed, built and trained, our predictive system could serve as a prototype 'back-end' for a deployable insect classification application. The goal of this project is to experiment with applying HD-CNN techniques to to this particularly challenging data set. Our approach borrows heavily from Yan et. al.'s 2015 work on HD-CNNs. They achieved state of the art results on the Imagenet-1000 data set (23.69 % Top-1, 6.7 % Top-5 misclassification errors [17, 18].

2. Background and Related Work

The idea of exploiting and/or learning naturally occurring category hierarchies for image classification problems has been the focus of a great deal of work in deep learning literature, particularly over the last 5 years. A 2012 survey on this specific area of research is

[1] The application UI might provide the user the opportunity to provide some additional information (in free text) if they know something about the insect (i.e. the user thinks the insect is an ant, but is unsure of which kind). Upon uploading, the user might also passively provide additional contextual information including temporal data (hour of the day, time of year). Geo-location data which could be helpful in increasing the accuracy of the classification.

well-summarized by Tousch [15].

In Deng's 2012 work on optimizing accuracy-specificity trade-offs in large scale visual recognition, he explored the concept of making predictions down a heirarchical class tree. The depth at which a prediction was made would correspond to the confidence of the prediction. For example, if uncertain about a particularly challenging image of an animal, the algorithm would predict a mammal instead of taking the risk of wrongly predicting a specific type of mammal [6]. They achieved 19% misclassification rates on leaf classes. In 2014, Deng et al. introduced another hierarchical approach creating a HEX graph[5]. This graph takes advantage of the fundamental characteristics of hierarchically organized categories and sub-categories of classes. For example, a Husky is a type of dog, but a dog should never be confused with a cat. This algorithm improved accuracy by about 8% over previous work. In 2010, Bengio et al. created an algorithm to create and train hierarchical trees of classes on (approximately) the whole tree's loss [3]. The primary objective of their reserach was speed, and they achieved a speed-up factor of approximately $85x$ with a slight gain in accuracy.

Bannour proposed a way to classify visual, conceptual, and contextual similarities between images based on their visual features, their WordNet descriptions, and the probability they appear together in an image [2]. In a following paper, Bannour et al. used these similarities in a bottom-up score fusion and improved accuracy over a flat model by 8.99 percentage points [1]. Yan's basic approach (which we seek to apply herein) strives to learn categorical hierarchies naturally, using clustering approaches to group classes together that might normally be confused as one another. Specialized parallel networks are then built to help improve prediction accuracy of these similar, hard-to-predict classes with hedged (weighted) prediction techniques. They achieved their top results with 84 overlapping class clusters to classify the ImageneNet-1000 classes [17, 18]. Many other recent papers have sought to learn heirarchical structures to categories in a natural way using a range of both top-down and bottom-up approaches. [16, 8, 12, 13]

Yan's 2015 HD-CNN uses a well-known 16-layer VGG architecture. This architecture has been used for a wide range of deep learning research over the last 2 years and has been convenient for its modularity and flexibility. The design of the VGG-16 architecture is described in detail by Simonyan et. al. [14]. When initially used on the Imagenet-1000 challenge, this architecture alone achieved an 8.1% top-5 misclassification rate, showing a great deal of promise. This 16-layer model was in part an evolution from another well-known 8-layer model which won the Imagenet challenge in 2012 with 37.5% top-5 accuracy. [10]

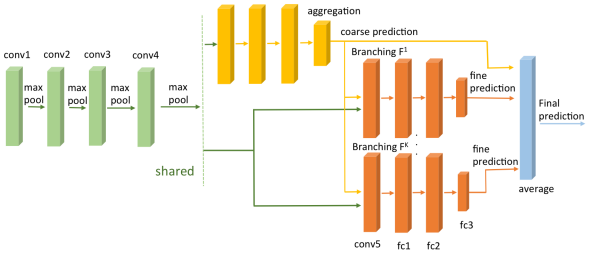

Figure 2. HD-CNN built on a VGG-16 foundation [17].

Figure 3. VGG-16 [11].

The current state of the art architecture for image classification was developed by Microsoft Research and won the ImageNet-1000 2015 competition [7]. Their model uses up to 128 layers with a residual loss function based only on the output of the previous layer for each layer. They achieved a top-5 error of 4.49%. These results are impressive, but the size of the model exceeds reasonable memory constraints for the purposes of our research.

Consequently, we utilized a the more manageable VGG-16 architecture for the insect classification problem outlined above. The primary intent of this research not to push the limit on state of the art accuracy, but rather to explore the benefits of various weighted (hedged) prediction techniques. .

3. Approach

3.1. Architecture

The VGG-16 architecutre is shown in figure 3. Each of the 5 orange groupings are *conv - ReLu - conv - Relu* followed by a *max pool* [2]. The ReLU non-linearity computes the element-wise maximum of 0 and the input x. The use of this function allows a deep network such as the VGG-16 to be trained more quickly, as its gradient is non-saturating. The discovery of this activation function was a key in deepening convolutional neural networks.

The 5 groupings of *conv* layers are are followed by 3 fully connected layers, each of which includes a 50% drop-out layer (the last 3 yellow boxes in Figure 2). The drop-out layers serve as regularizers in the learning process, preventing overfitting. The final output is a proba-

[2]Each of the *conv* layers use a *3x3* filter a a stride of 1. Each of the *max pool* layers use 2x2 pool size and a stride of 2

bility of prediction for each of the $C = 277$ classes, using SoftMax. The Softmax layer outputs, for each class:

$$\sigma\left(\sum_i w_i x_i + b\right)$$

This gives, for each class i, $P(y_i = 1; x_i, w)$. For each sample x, the class i with the maximum Softmax output is the predicted class for sample x.

We trained our networks using the Softmax loss function, which is defined here for each image i. f_{y_i} is defined as the value of the Softmax funcion for the correct class of the image, and the other f_i are the other values of the Softmax function.:

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

In our research, we only train the fully connected layers, fixing the rear convolution layers from a pre-trained network as feature extractors. In each iteration, we use back-propagation to calculate the gradient with respect to the loss at each fully-connected layer, then use an Adam policy to adapt the learning rate and change in each layer's weights over time. l is the learning rate we specify at that layer, and m and v are initialized to 0. The Adam algorithm updates the weights vector. This update takes into account the concept of momentum of the learning rate and an approximation of acceleration.

$$m = \beta_1 m + (1 - \beta_1)\frac{\partial w}{\partial L}$$

$$v = \beta_2 v + (1 - \beta_2)\left(\frac{\partial w}{\partial L}\right)^2$$

$$w = w - \frac{lm}{\sqrt{v} + \epsilon}$$

3.2. Methods

We first provide an overview of our training pipeline, with addition details follwing:

Step 1: Train and tune Building Block (BB) Net on train and validation sets.

Step 2: Pass the full validation set through the trained BB Net. Compute a CxC confusion matrix **C**. Use eigenmap dimensionality reduction to establish K clusters of classes that tend to be confused as one another. Using confusion matrix **C**, establish a mechanism to map between a probability vector outputted by the BB Net and a normalized vector of likelihoods (weights) assigned to each of the clusters.

Step 3: Train and tune each cluster net the subset of $c \leq C$ classes assigned to it, using only relevant training and validation data. (For our implementation, $61 \leq c \leq 80$.)

Step 4: Test time: forward pass all images through the BB net and all K specialized networks. Use weighting techniques to combine the outputs and form a class prediction for the image. Compute top1, top5 and top10 accuracy based on predicted classes.

Step 1 - Train 'building block' (BB) net: First, we fine-tune a pre-trained VGG-16 network. The inputs are mini-batches of randomly cropped and flipped 224 x 224 x 3 images (taken from re-sized images of 256 x 256 x 3). We trained the VGG-16 network outlined below on the training data. The final output is a probability for each of the $C = 277$ classes. Throughout the paper, we refer to this as our 'building-block' (BB) network

Step 2 - Learn Hierarchical Clusters: Next, we use the full validation set (which has a distribution between the classes equal to the training set) and forward pass it through the BB net, generating softmax probabilities over all 277 classes. From this, produce a confusion matrix **C** of dimensions CxC. With **J** as a square matrix of all ones, we compute a distance matrix **D = J - C** and perform eigenmap dimensionality reduction on **D** as described below:

For each row i, select the top k columns (a paraemter we defined as 20) in **D** and set all other columns to 0. Intuitively, this limits the calculation to those other classes which are often confused with each other. Then, we calculate:

$$W_{ij} = e^{\mathbf{D}_{ij}/0.95}$$

$$\mathbf{D}_{ii} = \sum_{j=1}^{n} W_{ij}$$

$$\mathbf{D}_{ij} = 0 \text{ if } i \neq j$$

$$L = \mathbf{D} - W$$

Then, we find the eigenvalues λ and eigenvectors v of L such that:

$$L\mathbf{v} = \lambda\mathbf{D}\mathbf{v}$$

We use the second through $C+1$ eigenvector to form a matrix E. We then apply the K-means clustering algorithm on the matrix E and assign each of the $C = 277$ classes to one of K clusters. [3]. Intuitively, the concept is to generate clusters of classes then tend to be confused as one another.

Yan et. al. initially experimented with disjoint clusters of classes (by setting a a hyperparameter $\mu_t = 0$),

[3]Spectral clustering methods like affinity propagation clustering are popular choices for performing this clustering process, however, given the prsopect that the value of K would require the training of K separate networks, we wanted to have greater control over the number of clusters and instead used K-Means on this matrix E

where a given class could only appear in one coarse category. This one-to-one mapping is the natural output of a clustering algorithm like K-Means or Affinity Propogation. When $\mu_t = 1$, all C fine classes are mapped to all K coarse classes. They found that some overlap ($0 < \mu_t < 1$) achieved the best results (see Figure 4), allowing for a many-to-many mapping with 84 overlapping coarse clusters for 1000 class. Once **Step 2** is complete, each coarse cluster has a centroid computed and a set of fine classes assigned under it.

The BB net will output a prediction of one of the C classes, which exists in one of the K clusters. We'd like to add a class to particular cluster if a particular class is likely to be misclassified into a class which exists in some cluster (where a cluster can be thought of as a coarse category). So we use the information contained in the confusion matrix **C**. Using the notation from Yan et. al. we estimate the likelihood $u_k(j)$ that a particular image predicted to be some class j is misclassified into cluster k on the validation set which is used to generate the clusters. We have:

$$u_k(j) = \frac{1}{|S_j^f|} \sum_{i \in S_j^f} B_{ik}^d$$

B_{ik}^d is the probability that sample i is assigned to cluster k, using the information contained in the confusion matrix and the vector of probabilities produced by the BB net. A class is added to an additional cluster based on two conditions: $u_k(j)$ is larger than some minimum (tunable) threshold and the number of classes in a particular cluster cannot exceed a maximum number of classes. [4]. When we are finished with this expansion of the clusters, we are left with a many-to-many mapping between clusters and classes and a methodology for computing the weight assigned to each of the clusters for a given prediction. We have also established a mapping between a particular probability vector outputted by the BB Net and a vector of likelihoods for each of the K clusters (coarse classes).

Step 3: Build, Train and Tune K 'coarse-to-fine' nets in parallel:

Next, we build and train a separate VGG-16 network associated with each of the K clusters, where each network is specialized in learning to predict the subset of C classes which exist in the cluster. We initiate the weights for each of these K specialized networks with the weights of the BB Net (except for the final fully connected layer *fc-8*, which is trained from scratch because the output dimension variaes). Since each of the K 'coarse-to-fine' nets are only providing predictions on

[4]The detailed implementation of this is available in our notebook. Yan et. al.'s HD-CNN paper also outlines the procedure in detail

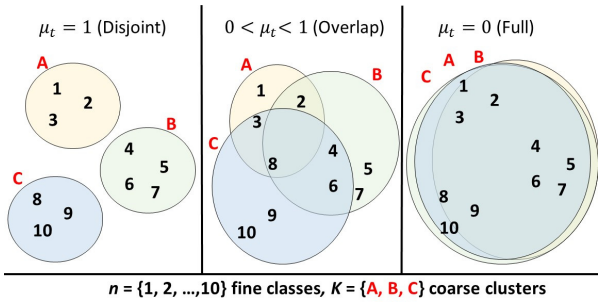

Figure 4. Clustering (and expansion of clusters) to learn a mapping from C fine classes to K coarse classes [17].

c classes where $c \leq C$, we only provide relevant training and validation data for those c classes. The BB Net and each of these networks both share the rear layers of the network. For our implementation, this included everything up to *conv-5*; everything from *fc-6* forward is uniquely trained to each of the K networks.

Step 4 - Perform classification with the BB Net and K Specialized Nets:

In Yan et. al., at this point, the HD-CNN architecture was fully assembled (ref. Figure 2), connecting the BB Net to the K specialized networks. Fine-tuning was conducted using a multinomial logistic loss function over the entire HD-CNN architecture. We don't take this final step (primary because of memory limitations associated with our computing resources). Instead, we execute a process to compute weighted probabilities using the trained building blocks of the HD-CNN on test data. We experiment with three different weighted prediction procedures.

The first approach uses weighted predictions using the BB net and all $K = 10$ clusters (which we refer to as the ALL CLUSTERS method). The second and third approaches (which we refer to as GATED and TOP CLUSTERS) only use a selected number of the K specialized nets to make predictions and then use the normalized cluster weights to make the final prediction.

In both approaches, we first pass a test image through the BB net and generate prediction probabilities over the C classes. From this probability vector, we compute a likelihood that the test should be routed to each of the K specialized networks based on mapping explained earlier (and exploiting the information contained in confusion matrix **C**). These likelihoods serve as the weights used for the predictions provided by the K specialized networks. For each image x_i with scores p output from the original BB Net, we create a matrix S such that $S_i = p$ and **C** as the confusion matrix defined above, we have:

$$F = \mathbf{C} \odot S$$

F_i is a Cx1 vector for each image. We followed three different procedures after computing this vector, called ALL CLUSTERS, GATED, and TOP CLUSTERS, giving SA, SG, and ST, respectively.

ALL CLUSTERS: For each cluster, select the scores from F_i for the classes output by that cluster and normalize those scores, creating a matrix M. Pass all test data through all K clusters. Calculate the final prediction as follows:

$$SA_i = \sum_{k=1}^{10} M_{ik} S_{ki}$$

GATED: For each image, let the top class predicted by the BB net be defined as class m. If cluster k didn't train on class m, set the weight of cluster k to 0 in M and all other weights to that in F, creating a matrix M. Re-normalize the cluster weights to sum to 1. Calculate the final prediction as follows for each image i:

$$SG_i = \sum_{k=1}^{K} M_{ik} S_{ki}$$

This procedure takes into account that each cluster is only trained on particular classes and therefore the output of other clusters should not be taken into consideration.

TOP CLUSTERS: Instead of considering the top prediction class by the BB net as in GATED, look at the cluster weights in F_i. Take the top t (such as t = 3 or t = 5) weights and re-normalize those, making a matrix M with the rest being 0. Calculate the final prediction as follows for each image i:

$$SG_i = \sum_{k=1}^{10} M_{ik} S_{ki}$$

The final classification for each image i and each method is the class for which the selected row i of the ouput matrix (from SA, SG, SI) is maximum. This is intuitively the class for which the probability is highest.

Classes	277
# Images	217,657
Images Per Class	
min	205
25th pctl	374
median	727
75th pctl	1113
max	2386
Split of Data	
Training	90%
Validation	5%
Test	5%

Figure 5. Summary stats of data set

4. Data

We identified 277 distinct synsets that fall under the 'insect' synset in the ImageNet word tree. We only used distinct synsets that had 200+ images. The names for these 277 synsets are not scientific names for specific species. Rather, they are often the common name for the insect (or the grouping of two or three common names together). Manual inspections of the images revealed a significant degree of variance and mislabeling of the data in these insect classes. We used 90% of the images for training, 5% for validation (and to compute the confusion matrix for the clustering procedure), 5% held out for testing. We have some imbalance between classes, but this can be learned by the CNN. Before being packaged into `lmdb` files, the images were re-sized to 256 x 256 x 3. Additionally, the images were centered using the mean values from the training set. Our train, validation, and test sets were all shuffled and packaged into `lmdb`'s which can be easily read in as mini-batches by `Caffe`.

In addition to the data quality issues highlighted above, since many insects look alike, have limited variability in color and often are well-camouflaged with their backgrounds, we suspect that this will be quite a challenging data set to work with and our accuracies will be well under state of the art. [5]

5. Experiments, Results and Analysis

Our primary objectives in this research was to maximize top5 and top10 accuracy on the BB VGG-16 net, and to then maximize improvement using the clustering and weighted prediction techniques outlined above.

We provide a summary of our experimentation and results below. Additional details on our implementation and analysis in the compiled iPython Notebook attached to this report.

Step 1: Training 'building-block' net

We started with a pre-trained VGG-16 net used by Yan on the Imagenet-1000 challenge. Our first task was to fine-tune this building-block net that would take mini-batches of 25 centered and randomly cropped 224 x 224 x 3 images as input and then output softmax probabilities over the 277 classes as output. A few selected attempts of this training process are shown in the attached notebook.

Giving constraints on computing time, our general strategy was to fix *conv* layers (setting learning rate multipliers to 0) and training forward layers. Given that this network had been pre-trained on ImageNet images, we wanted to exploit the feature extraction capabilities of the rear layers without spending compute time to re-train them and instead focused on fine-tuning the fully connected layers. Our images are a subset of the ImageNet-1000 images, so we had confidence in the feature extrac-

[5] In ImageNet-1000, the relative difference between the classes is generally greater and the data set is cleaner. We noticed that the 1000 ImageNet Classes often cluster several ImageNet insect classes into one sysnset. For example bees, ants, moths, would all be a single class in ImageNet1000. We are keeping insect classes separate to strive for maximum granularity on the classification, which is important given our aforementioned objectives for this project.

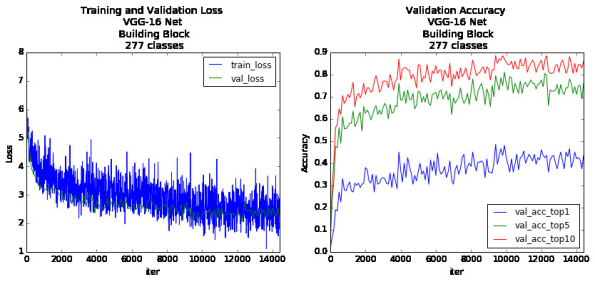

Figure 6. Training of Building Block (BB) Net

tion capabilities of the previously trained network.

Ultimately, we achieved our best results when fixing layers up through *pool5* and then training *fc-6* to the output. We tracked loss along with top1, top5 and top10 accuracy of the output. In all attempts, the *fc-8* layer was initiated from scratch (because the output dimensions differed from the downloaded network), but all other layers utilized the weights from the pre-trained VGG-16 as a starting point and fine-tuned from there.

Because of the large size of the VGG-16 and memory constraints, we were limited to a mini-batch size of 25, which undoubtedly led to instability in our gradients and certainly hindered our training process. Given the under-sized mini-batch size, we attempted to compensate with utilization of lower learning rates to slow down the learning process.

We used a coarse-to-fine randomized hyper-parameter search, tuning learning rate, initialization of weights in layer *fc-8* and weight decay, achieving best results with a learning rate of 1e-4, a weight decay of 5e-5 and a standard deviation on the weight initializations for fc-8 = $\sqrt{4096} = 0.15$.

We used an Adam update policy with the standard default momentum ($\beta_1 = 0.9$) and momentum2 ($\beta_2 = 0.99$) and delta ($\epsilon = 1e-7$) hyper parameters used in `Caffe`, borrowing best practices from existing literature working on image classification using a VGG-16 architecture [9]. We also experimented with Stochastic Gradient Descent and found greater stability in our loss convergence with Adam.

We experimented with different learning rate decay functions available in `Caffe` including `fixed`, `step`, `multistep`, `inv`. Ultimately, we used a step-policy to step down the base learning rate every epoch (7200 iterations) by a multiplier of 0.4.

After extensive experimenting, our final BB Net was trained for 2 epochs over full training data set (Figure 6).

Step 2: Test Building Block Net; Use Confusion Matrix to Learn Hierarchical Clusters

Following the fine-tuning of the building block net, we used the entire validation batch of 11,017 images from the 277 classes (unseen during the training process). Performance by class over the entire validation set is then consolidated into a normalized confusion matrix (where row sums = 1). A visualization of the symmetric confusion matrix **C** is provided in Figure 7; dark blue represents a value ≈ 0 and whiter spots in the visualization represents values closer to 1. As described in Methods, we created matrix **F** from **C**.

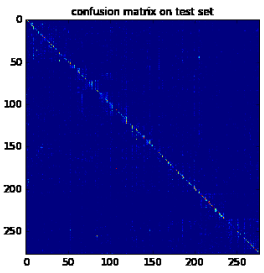

Figure 7. Normalized confusion matrix on validation data

We then used **F** to cluster classes into groupings where classes within the cluster tend to be confused with one another. We experimented with affinity propagation clustering as well as K-Means, ultimately using K-Means to assert more control the number of K clusters generated to a reasonable level (given our constraints on compute time). Yan et al. found that 84 clusters over the 1000 classes worked well, so the same ratio of total classes C to clusters K would have called for around 22 clusters C = 277 classes. However, we utilized just 10 clusters.

The K-Means procedure resulted in an assignment of classes to the $K = 10$ clusters (left plot, Figure 8. As expected, we observed classes close to each other in the ImageNet WordTree tended to be in the same cluster. For example, most butterflies were assigned to cluster 6 and most beetles were assigned to cluster 8.

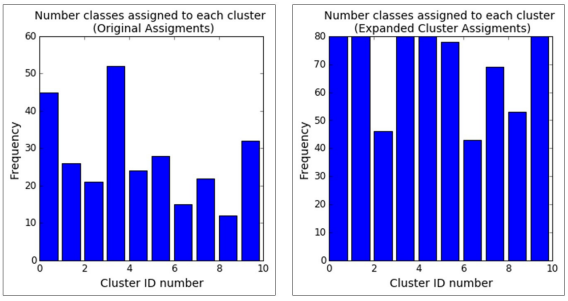

Figure 8. Classes per cluster (original assignments (L); expanded assignments (R)

We next expanded the clusters, allowing a class to be represented across multiple clusters. Two important hyper-parameters controlled this process: the maximum number of classes allowed into a given cluster and the minimum similarity score for a given class to a given cluster, which exploited the information contained in the confusion matrix to provide a sense of how often a particular class tended to be confused as the classes con-

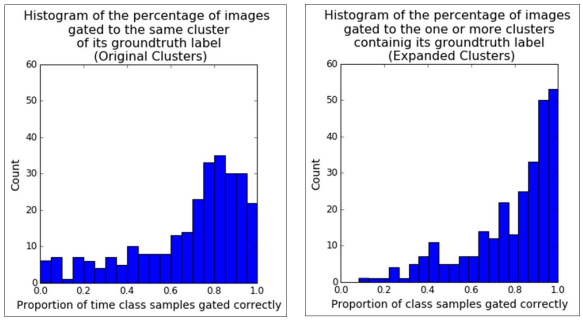

Figure 9. Proportion of time a class would be gated to proper cluster (L) original cluster assignments; (R) expanded cluster assignments

tained in a cluster. We utilized **max class** = 80 and **min similarity score** = $1/(2k) = 0.05$. The resulting number of classes per cluster after the expansion is the right plot of Figure 8.

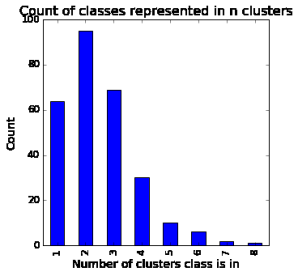

Figure 10. Histogram of clusters per class (expanded clusters)

We conducted extensive experimentation with these two hyperparameters, watching for two primary outcomes. First, given the top1 prediction made by the BB Net, what was the likelihood that the true class would exist in the cluster that the building block net suggested was 'best'? Under our hierarchical approach, we didn't want to doom our final predicted class estimation by having a particular test image incorrectly routed to a cluster which didn't even contain the true class. Given our data set, and the propensity for relatively weak predictions, we wanted to utilize a hedging strategy in which a class was represented in multiple clusters, increasing the probability that a test image could be routed to one or more clusters that was specially trained on the true class of that image. Using the information from the forward pass of the validation data, we retrospectively analyzed what would have happened to different classes (on average).

Using the 'original' cluster assignments generated by the K-Means procedure (where each class is assigned to only one cluster), we see a significant number of test images would have been routed to a cluster that didn't contain the true class less than more than 40% of the time (left plot, Figure 9). However, when expanding the clusters to allow particular classes to be in multiple clusters, we can see that the distribution shifts significantly. Many more classes are *always* routed to at least one cluster where the true image exists (right plot, Figure 9).

We also analyzed the resulting distribution of clusters per class, achieving a desirable outcome in which most classes are represented in 2 or 3 clusters. The classes assigned to only 1 cluster are classes that the BB Net is likely to predict with high confidence. Those classes which end up in 6, 7 or 8 clusters have relatively low confidence in the BB Net prediction and are confused as a wide range of different species.

Step 3: Build, Train and Tune K 'coarse-to-fine' nets in parallel

We next re-partitioned the training and validation data for each of the $K = 10$ clusters so as to only include images from the true class a particular cluster specializes in. We used the BB Net's weights to initiate and trained *fc-8* from scratch (since the number of classes being predicted was different).

We experimented with a range of hyperparamters for the training effort, and ultimately fine-tuned the same hyper-parameters utilized on the best training effort for the BB Net (base learning rate = 1e-4). The full results from the training efforts on each of the $K = 10$ specialized networks is included in the attached iPython notebook. [6]

Step 4 - Compute weighted predictions:

Test Set Results Summary (Accuracies)			
Prediction Method	top1	top5	top10
Building Block (BB) Prediction	42.14%	75.86%	84.93%
Weighted Prediction (ALL CLUSTERS)	43.83%	77.46%	85.99%
Weighted Prediction (GATED)	44.56%	77.82%	85.32%
Weighted Prediction (TOP 5 CLUSTERS)	44.03%	77.66%	86.13%

Figure 12. Results summary

We next utilized the three weighted prediction procedures to make predictions over the 11,107 test images as described in Methods (ALL CLUSTERS, GATED, TOP CLUSTERS). Results are summarized in Figure 12. The iPython notebook also includes a by-class summary of performance on the test set using just the BB net as well as all three weighted approaches. In earlier attempts, when our accuracy from the BB Net was not as low, we achieved a relatively greater boost from the weighted methods. Here, we achieve about a 2 % from the weighted probability techniques compared to the baseline of our BB Net accuracies. Generally, we noted that classes that had very strong performance on the BB Net actually decreased in performance slightly using the

[6] We know that it would have been beneficial to train these specialized nets slower, longer and deeper than we had trained the BB Net, but were constrained by computing resources. We cut off training after 1.5 epochs (1 epoch = 1 full pass through available training data).

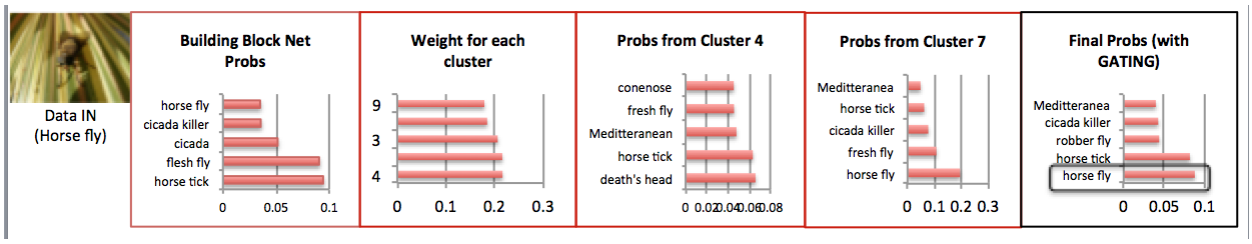

Figure 11. Process of predicted an image of a horsefly using the clusters

clustered method (eg. monarch butterfly). Those classes which performed very poorly on the BB Net improved classification accuracy significantly using one or all of the hedged prediction approaches (eg. biting midge; Figure 14).

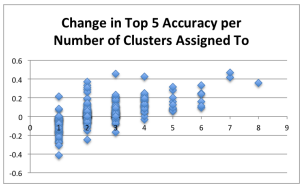

Figure 13. Improvement over BB-net increases as number of clusters a class is assigned to increases

The GATED procedure (similar to the Conditional Execution in the Yan paper) is certainly faster and requires fewer predictions. However, if the top prediction made by the Building Block network is wrong (and a class unlike the other classes in the cluster), we are likely to get a poor prediction. This is unlikely given the way that classes are assigned to clusters but still possible. The TOP CLUSTERS procedure is similarly faster and has the advantage of taking into account both the results of the BB net and the data from the confusion matrix. ALL CLUSTERS is the most hedged prediction that considers all of the predictions made by the K specialized nets; the relative confidence of those predictions are then proportional to the weight given to a particular cluster. The obvious disadvantage is a slow-down in computational time.

Figure 11 shows how a single image would be classified using the weighted approach using the outputs of the BB Net, mapped probabilities for each cluster, and class predictions for 2 (of the possible 10) clusters. The horse fly images in the test set were all misclassified using the BB Net alone, but achieved a top-5 accuracy of 72.2% using the TOP CLUSTERS. The image has few broad features to distinguish it from other flies. The outputs of the two clusters with the highest scores, clusters 4 and 7, are shown in the figure as well. We note that cluster 4, with the highest score, does not assign horse fly the highest probability. However, cluster 7 gives horsefly the highest score by far, and in the end this image is classified correctly as a horse fly.

Figure 13 confirms that our approach was most effective on classes assigned to many clusters. A class is assigned to many clusters if it is confused with other classes often (and the confidence of predictions from the BB Net are weak). This makes sense intuitively: for a class only assigned to one cluster, the cluster net would not make as accurate of predictions as it was trained for fewer iterations. The power of the hierarchical approach is the increased learning that comes from training only with similar images.

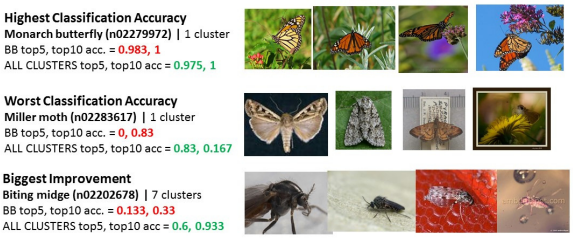

Figure 14. Sample class performance

6. Conclusion

Our final results show an improvement of roughly $2-3$ percentage points in accuracy over the building block net, resulting in a top-5 accuracy of 23.03%. We know that further improvement could be achieved by training the specialized nets deeper and longer. Better quality assurance of the data in the classes would have certainly also helped increase performance. We could also implemented a different network architecture for the building block and cluster nets, such as that used by Microsoft Research in the 2015 competition, likely achieving stronger results overall. However, these initial results show great promise for the deployment of HD-CNN techniques for boosting classification accuracy on particularly challenging data sets. Additionally, we are encouraged by the initial accuracy achieved on the insect classification problem, even with a minimal amount of training. We believe that there would be tremendous benefit to an insect identification application and hope that this work will provide a starting point for further work on such a technology.

References

[1] H. Bannour and C. Hudelot. *Building semantic hierarchies faithful to image semantics*. Springer, 2012.

[2] H. Bannour and C. Hudelot. Hierarchical image annotation using semantic hierarchies. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2431–2434. ACM, 2012.

[3] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, pages 163–171, 2010.

[4] A. D. Chapman et al. Numbers of living species in australia and the world. 2009.

[5] J. Deng, N. Ding, Y. Jia, A. Frome, K. Murphy, S. Bengio, Y. Li, H. Neven, and H. Adam. Large-scale object classification using label relation graphs. In *Computer Vision–ECCV 2014*, pages 48–64. Springer, 2014.

[6] J. Deng, J. Krause, A. C. Berg, and L. Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3450–3457. IEEE, 2012.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[8] Y. Jia, J. T. Abbott, J. Austerweil, T. Griffiths, and T. Darrell. Visual concept learning: Combining machine vision and bayesian generalization on concept hierarchies. In *Advances in Neural Information Processing Systems*, pages 1842–1850, 2013.

[9] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] F.-F. Li, A. Karpathy, and J. Johnson. Cs231n course lecture slides. 2016), publisher = Stanford University.

[12] L.-J. Li, C. Wang, Y. Lim, D. M. Blei, and L. Fei-Fei. Building and using a semantivisual image hierarchy. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3336–3343. IEEE, 2010.

[13] M. Marszałek and C. Schmid. Constructing category hierarchies for visual recognition. In *Computer Vision–ECCV 2008*, pages 479–491. Springer, 2008.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[15] A.-M. Tousch, S. Herbin, and J.-Y. Audibert. Semantic hierarchies for image annotation: A survey. *Pattern Recognition*, 45(1):333–345, 2012.

[16] N. Verma, D. Mahajan, S. Sellamanickam, and V. Nair. Learning hierarchical similarity metrics. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2280–2287. IEEE, 2012.

[17] Z. Yan, V. Jagadeesh, D. DeCoste, W. Di, and R. Piramuthu. Hd-cnn: hierarchical deep convolutional neural network for image classification. *CoRR, abs/1410.0736*, 2014.

[18] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. Hd-cnn: Hierarchical deep convolutional neural network for large scale visual recognition. In *ICCV'15: Proc. IEEE 15th International Conf. on Computer Vision*, 2015.