Title
Towards Scalable and Accurate Fine-Grained Categorization /

Permalink
https://escholarship.org/uc/item/0v50q1rs

Author
Van Horn, Grant Richard

Publication Date
2014

Peer reviewed|Thesis/dissertation
UNIVERSITY OF CALIFORNIA, SAN DIEGO

Towards Scalable and Accurate Fine-Grained Categorization

A Thesis submitted in partial satisfaction of the requirements for the degree of Master of Science

in

Computer Science

by

Grant Richard Van Horn

Committee in charge:

Professor Serge Belongie, Chair
Professor Sanjoy Dasgupta
Professor David Kriegman

2014
The Thesis of Grant Richard Van Horn is approved and is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2014
TABLE OF CONTENTS

Signature Page ................................................................. iii
Table of Contents ................................................................ iv
List of Figures .................................................................... vi
List of Tables ...................................................................... vii
Acknowledgements .............................................................. viii
Abstract of the Thesis .......................................................... ix
Chapter 1 Introduction .......................................................... 1

Chapter 2 Vibe ................................................................. 3
  2.1 Introduction ................................................................. 3
  2.2 Related Work ............................................................... 5
  2.3 Design ............................................................ 7
    2.3.1 Label Hierarchy ...................................................... 8
    2.3.2 Image Management ............................................... 13
    2.3.3 Snapshots .......................................................... 15
    2.3.4 Cloud Implementation ........................................... 17
  2.4 Use Cases ................................................................. 18
  2.5 Future Work ............................................................... 20
  2.6 Conclusion ............................................................... 21

Chapter 3 Fine-Grained Visual Categorization ......................... 22
  3.1 Introduction ............................................................... 22
  3.2 Related Work ............................................................. 23
  3.3 Pose Normalization Schemes .......................................... 26
    3.3.1 Pose Normalization By Prototypical Regions .............. 27
    3.3.2 Learning Pose Prototypes ....................................... 28
  3.4 Deep Convolutional Features ......................................... 31
    3.4.1 Multi-Layer CNN Features For Different Alignment Models ... 31
    3.4.2 Training the Convolutional Neural Net ..................... 32
  3.5 Experiments ............................................................. 34
    3.5.1 Summary of Results and Comparison to Related Work ...... 35
    3.5.2 Comparing Feature Representations .......................... 38
    3.5.3 Comparing Part Localization Schemes ...................... 40
    3.5.4 Comparing CNN Learning Methods .......................... 43
  3.6 Conclusion ............................................................. 44
LIST OF FIGURES

Figure 2.1. Vibe Interface ................................................. 7
Figure 2.2. Vibe’s File System ........................................... 9
Figure 2.3. Label Hierarchy ............................................... 10
Figure 2.4. Label Chains .................................................. 11
Figure 2.5. Snapshot Interface .......................................... 16
Figure 2.6. Collaboration Example ...................................... 19
Figure 3.1. Warp Visualizations ......................................... 29
Figure 3.2. Feature Performance Comparison ......................... 38
Figure 3.3. Effect of CNN Layers For Different Regions .......... 39
Figure 3.4. Improvement From GT Parts ............................... 41
Figure 3.5. Effect of Fine-Tuning, GT Parts ......................... 41
Figure 3.6. Effect of Fine-Tuning, Pred Parts ....................... 42
LIST OF TABLES

Table 2.1. Example labels and their potential meaning to a curator. Note that labels can represent many different situations, not just dataset category names. .......................................................... 8
Table 2.2. Vibe Execution Times ....................................................... 13
Table 3.1. Wapring Functions .......................................................... 28
Table 3.2. Comparison to Related Work on CUB-200-2011 ................. 36
Table 3.3. Comparing Different strategies for combining multiple regions when part locations are given at test time. The number in parentheses indicates the number of regions used for each method. ............ 43
ACKNOWLEDGEMENTS

I would like to acknowledge Professor Serge Belongie for his incredible support during my time at UCSD. From undergrad to grad, his guidance has been invaluable.

Chapter 3, in part, has been submitted for publication of the material as it may appear in British Machine Vision Conference, 2014, Branson, Steve; Van Horn, Grant; Belongie, Serge; Perona, Pietro. The thesis author and Steve Branson contributed equally to this paper.
ABSTRACT OF THE THESIS

Towards Scalable and Accurate Fine-Grained Categorization

by

Grant Richard Van Horn

Master of Science in Computer Science

University of California, San Diego, 2014

Professor Serge Belongie, Chair

In this work I contribute two solutions towards increasing the utility and performance of fine-grained categorization. First, I present a software infrastructure designed to ease the burden of collecting and managing a fine-grained datasets. Second, I present a technique that significantly advances state-of-the-art performance on bird species categorization. These contributions provide the groundwork for expanding to other fine-grained domains.
Chapter 1

Introduction

Fine-grained categorization is a computer vision problem where the task is to distinguish subordinate-level categories from each other. Bird species recognition is a canonical example of this type of problem. In this setting, the computer knows that the image contains a bird, and the task is to predict the species of the bird. This type of task requires a great attention to the details of the category; the general appearances of the subcategories are most likely highly similar with only small interclass differences.

Conveniently, even though fine-grained tasks require an attention to detail not necessary at the coarse-grained level, the subcategories will often have the same general shape, made up of the same parts and/or attributes (e.g. birds all have the same parts: bill, throat, breast, tail, nape, etc.) or be in an environment that is constrained (e.g. plates of food). The computer can utilize this information to localize distinguishing features that are capable of separating the subcategories.

In order to train a computer to distinguish fine-grained categories, a dataset of category labels and images must be provided. Separate tasks may have further annotation requirements (e.g. birds may need part annotations, food may need segmentation annotations), but all of the tasks share the same basic requirement of needing images and labels. One of the difficulties of collecting this type of dataset is that most individuals do not have the knowledge to correctly divide the subordinate categories. For example,
in order to build a dataset to train a bird species classification system, you need to be able to categorize the species yourself. So, not only is it difficult to train a computer to recognize the differences between subordinate categories, it is difficult to even collect the data necessary to attempt the training.

In this work, I present solutions aimed at this dual problem of fine-grained categorization. First I present a set of tools aimed at easing the challenges of collecting a fine-grained dataset. Second I present a technique that significantly advances the state of the art in fine-grained bird species categorization.
Chapter 2

Vibe

2.1 Introduction

It goes without saying that datasets are a fundamental component of the computer vision field. Datasets not only allow researchers to test hypotheses and compare algorithms, but they also provide a way to challenge the field and measure progress. When transitioning new algorithms to the commercial setting, a dataset containing the domain specific visual concepts must be collected in order for the algorithm to be applicable. There is a fundamental reliance on datasets then, as they are needed to both progress research and to give utility to the results of research.

It is interesting that given the importance of datasets, and the potential influence they have on the field, there has been little work done to produce tools that are designed specifically for their creation. Our experiences with dataset creation and management usually start with grand ideas of reusable tools, and then quickly degrades to one-off tools that are hacked together to get the mundane data collection job finished. Similarly, given the cost of creating new datasets, it is interesting that there has not been more effort to organize the existing datasets so that they are easily accessible. While there may be a handful of very popular and accessible datasets, there exists many other datasets, perhaps used in one specific paper, that require non-trivial searching to locate and download.
The future of dataset creation using traditional techniques is looking bleak. As
the vision field has matured, there has been increasing pressure for larger and more
challenging datasets, witnessed in the evolution of popular datasets such as Caltech-
4/101/256, PASCAL VOC, MSRC, and ImageNet [23, 28, 21, 18] and the introduction
of specific fine-grained datasets for categories such as aircraft, birds, cars, dogs and
Stoneflies [48, 67, 39, 37, 47, 50]. This pressure for more challenging datasets only
amplifies the difficulty in collecting them. Simply working with large collections of
images becomes a burden as the data becomes unwieldy. As the field’s interest turns
towards more fine-grained category distinction, expert knowledge of the domain is
required to create the dataset categories, and either expert validation or creative crowd
sourcing is required to collect ground truth labels.

For better or worse, many of the datasets in the computer vision field were
created by vision researchers themselves. Perhaps the lack of dataset collaboration
tools has prevented multidisciplinary efforts. Organizing and moving images between
collaborators is a difficult task, and without proper tools it can be frustrating to handle
evolving project requirements among larger teams, especially when the members are
distributed across institutions. This extra hassle added to the already mundane prospects
of collecting data seems to dissuade vision researchers from initiating these potentially
fruitful collaboration efforts.

Some parts of dataset construction are inherently project specific, or are the result
of current trends in the field. Pixel level annotations, such as bounding boxes, parts, and
segmentations, are entirely project specific and may become passé with future algorithms.
Tools to support these operations are better left as standalone tools that can accept inputs
of images. On the other hand, simply browsing and analyzing images are fundamental
requirements of any dataset project. Making the browsing and modification functions as
efficient and easy to use as possible allows for fast analysis and curation of the dataset.
Adding support for easy sharing of the data with collaborators further decreases the construction burden and opens the door to multidisciplinary collaborations. We have created Vibe (Visual Indexing Back End), a data management system that supports precisely these fundamental operations.

Vibe abstracts away the fundamental and mundane issues of working with image datasets. Entire datasets can be easily browsed, a label hierarchy can be created to organize the images, and data can be easily shared with other collaborators. The results of the organization process can be downloaded and utilized in a vision application. Vibe is our attempt to follow through with our grand ideas of reusable management tools.

Vibe comprises the following contributions. First, it moves dataset management into the cloud. Using the cloud will enable Vibe to scale with the increasing size of datasets, will serve as an accessible repository for datasets, allows for fast sharing of data, and enables images to be served to stand alone annotation tools. Second, it provides management operations specifically designed for the curation and analysis of image datasets. We have modified the standard file system to be more accommodating to dataset management and can support interactions that affect thousands of images. Third, it provides an straightforward method of sharing data between users to support collaborative efforts. Vibe will serve as an excellent starting point for multidisciplinary collaborations, allowing collaborators to define a hierarchy, split up curation tasks and merge the results back together.

2.2 Related Work

In many respects, Vibe is simply a tool for making other tools, a data curation workhorse that is built specifically for computer vision projects. Vibe’s closest relatives are probably personal photo organizing tools, such as Picasa and iPhoto, and file sharing applications such as Dropbox. Vibe’s label hierarchy is very similar to the tagging
and album creation ability found in personal photo organization tools. Vibe supports much more aggressive interactions with labels however, supporting powerful commands designed to move a large number of images to new or existing labels for efficient analysis and modification of datasets. Vibe’s dataset sharing capabilities are very similar to the way files and directories are shared among users in file sharing applications. Because Vibe’s scope is much more focused than general file sharing applications, we have sacrificed generality to allow Vibe’s users to quickly share large datasets with each other. Vibe currently runs only in the cloud, with no automatic syncing to a user’s local machine. When users want to process the images on their local machine they can manually download the images from Vibe.

Vibe should not be confused with image search. Rather, Vibe’s should be thought of more like a backend for image search systems, and is intended to help build and maintain them. Considering that some companies, such as Google, Flickr, Facebook, TinEye and Microsoft, already work with large, tagged image datasets, a tool similar to Vibe may already exist in industry. We were unable to find public versions of these tools.

Vibe should also not be confused with content based image retrieval. CBIR applies computer vision techniques to the problem of searching for images within a large database. Smeulders et al. [63] gives a good overview of the state of CBIR circa the year 2000. In this context, with its heavy use of databases and focus on interactivity, Vibe can be thought of as a back end for a CBIR system.

There have been photo management studies, particularly personal photo management, in the research literature. Kang et al. [35] give a good overview of approaches. Certain aspects of these studies, for example [26, 52, 60], and projects, such as PhotoFinder [36], appear in consumer photo organization software, such as Picasa and iPhoto. Vibe is similar to these projects due to the focus on interactivity and organizational capabilities. The PhotoFinder project is particularly similar to Vibe, but is more focused on browsing
Figure 2.1. This figure displays the basic interface to Vibe. On the left is the label hierarchy. It is visually similar to a typical file system interface. Users can arbitrarily drag and drop the directories, create new ones, duplicate them, delete them and perform dataset specific operations with them (merges, intersections, unions, etc.). Selecting a directory will display its contents in the center panel to the right of the label hierarchy. Users have the option of displaying the images found only in this directory, or can display all images found in the subtree rooted at this node. Images can be dragged individually or in groups to directories found in the label hierarchy. Directories that will be used often during a curation session can be added to the Short Term Memory panel, found at the bottom of the interface, for quick access. For experienced users, the interface contains many advanced interactions, beyond drag and drop, for faster data movement.

and searching. In general, these studies and projects are not geared towards manipulating large datasets and therefore lack useful and efficient data movement operations. Because of the focus on personal photo organization, a lot of this work does not introduce collaborative capabilities.

### 2.3 Design

This section will discuss the high level design of Vibe and the interactions and features that are available to users.
Table 2.1. Example labels and their potential meaning to a curator. Note that labels can represent many different situations, not just dataset category names.

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mallard</td>
<td>all images given this label contain a Mallard</td>
</tr>
<tr>
<td>John Doe</td>
<td>all images given this label were taken by John Doe</td>
</tr>
<tr>
<td>Instance Count 3</td>
<td>all images given this label contain 3 instances of the object</td>
</tr>
<tr>
<td>Pose Left</td>
<td>all images given this label have the object facing left</td>
</tr>
</tbody>
</table>

2.3.1 Label Hierarchy

The primary goal of Vibe is to make it easy to curate large datasets of images. Large, in this context, can mean thousands, hundreds of thousands or even millions of images. Curate is specifically ambiguous in this context. By curate, we simply mean group images by labels. The canonical example of a label is a particular category name in a dataset. However, a label is simply a textual artifact useful to the curator and is therefore flexible enough to represent many different situations. The source of the labels is inconsequential, the important part is that a curator can view the images that have a particular label and perform actions with those images. Table 1 provides examples of labels and their meanings.

To make the notion of labels more concrete, we have chosen think of them as nodes in a tree and to represent them as directories in a file system. We view the file system as a label hierarchy, with each node in the label hierarchy represented by a directory that can contain both images and other directories. One particular path in the hierarchy represents a label chain, with each node along the path adding specificity. Examples of label chains can be found in Figure 4.
Figure 2.2. This figure highlights the difference between a traditional file system and Vibe’s file system. In the traditional file system it is possible to have the same image, perhaps with a different filename, in multiple directories along the same directory path, or even duplicated within the same directory. With this representation it is the user’s responsibility to keep track of an image’s location in the hierarchy, to determine whether an image has already been curated, and to keep tabs on duplicate images. In Vibe, an image can only reside in one directory (or node) along a given path and exact duplicate images are managed automatically. Curation and analysis in Vibe is very efficient because the hierarchy itself maintains strict organization of the images. The location of an image is precisely defined and users can take advantage of this to manage the curation progress. For example, if the goal is to put all images in a leaf node, curators can quickly analyze internal nodes of the hierarchy (e.g. the Birds, Ducks and Hawks nodes in this example) to determine which images need further curation.
Figure 2.3. This is an example of a label hierarchy for a dataset project. The dataset categories are rooted at the Birds node. Extra meta data is stored with the images by creating the necessary label chains. In this example, the user is tracking the image photographers and the image sources. Results of annotation tasks can be used to generate more label hierarchies. In this example, the results of a bounding box annotation task was used to group the images by the number of birds they contain. Operating on the nodes allows users to perform visual queries. In this example, the user could quickly create a new node that contains all images from John Doe that contain one Wood Duck.
Figure 2.4. Label chains are simply paths in the label hierarchy, and are used to represent increasingly refined curation progress. Users can use the label chains to analyze the curation progress of their dataset. Operations between nodes in different label chains allow users to perform visual queries. For example, given these label chains, a user could create a new node that has images taken by John Doe of a single male Mallard that is facing left.
A particular image can be in multiple nodes of the hierarchy, but it can be in only one node along a given path. Because of our notion of label chains, having the image reside in an ancestor of a node would be redundant. This was implemented by turning the standard file system into a nested set structure. Each node contains images as well as children sets. The intersection of a node’s content with each of its ancestors’ content is the empty set. The difference between a typical file system and Vibe’s file system can be seen in Figure 2.

Vibe supports all the typical file system interactions. Users can create new nodes, modify the name and properties of nodes, duplicate nodes, move nodes, and delete nodes. When creating nodes, the user can perform set operations to initialize the contents of a node. This is implemented by allowing intersection, union and not operations on the nodes. These operations allow for very useful visual queries, and are essentially a GUI around database row operations. See Figures 3 and 4 for an example of a visual query use case. We have also implemented specific interactions that are particularly useful when multiple people are collaborating on a dataset. For example, there is a command to merge two nodes of the hierarchy. Descendant nodes of these two nodes will combine if their names match, otherwise the nodes will become siblings. This type of operation will allow multiple people, who initially started with the same hierarchy, to combine their work at a later point. See Figure 6 for an example of this operation.

Because we have turned the typical tree based file system hierarchy into a nested set hierarchy, we have to enforce the nested set rules after each modification by the user. In particular we have to enforce the rule that after each interaction, the intersection of a node’s content with its ancestors’ content is the empty set. Careful design considerations of Vibe’s database allows us to enforce these rules very efficiently.
Table 2.2. Execution times in seconds for various interactions on label hierarchies that contain 1,000 images and 500,000 images. Note that these times are only approximations as the load on the web servers plays a role. We expect these times to decrease as Vibe reaches a more mature and stable state.

<table>
<thead>
<tr>
<th>Operation</th>
<th>1,000 Images</th>
<th>500,000 Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate</td>
<td>&lt;1</td>
<td>&lt;120</td>
</tr>
<tr>
<td>Take Snapshot</td>
<td>&lt;1</td>
<td>&lt;120</td>
</tr>
<tr>
<td>Share Snapshot</td>
<td>&lt;1</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Load Snapshot</td>
<td>&lt;1</td>
<td>&lt;120</td>
</tr>
<tr>
<td>Remove</td>
<td>&lt;1</td>
<td>&lt;45</td>
</tr>
</tbody>
</table>

2.3.2 Image Management

Operations on the nodes of the hierarchy would be costly if the actual content of the nodes had to be modified as well. A simple interaction with the label hierarchy could easily affect thousands of images. In a traditional file system, an operation moving thousands of images would be prohibitively slow. We bypass this cost by utilizing a database layer in which the nodes store references to images rather than storing the actual images. For example, in order to duplicate a node, all we have to do is create a new node and copy the references contained in the original node into the new node. There is still a cost when duplicating a node that has many descendants, since there is the cost of duplicating the actual nodes and their properties, but this operation is much faster than if we had to duplicate the content as well. See Table 2 for the cost of different operations.

Having nodes store references to images means that for a particular image in Vibe, there is only one copy of that image, even across users. When an image is found in multiple nodes, or found in nodes of different users, the references all point to the same image resource. Therefore when an image is uploaded, Vibe will only store user agnostic meta data with the raw pixel data. We have designed Vibe so that any extra meta data a user wants to associate with an image is done through the label hierarchy. For example, to associate a photographer with an image, a user could create a “Photographer” node,
within that node they can create a “John Doe” node for the photographer, and then they could add the photo to the “John Doe” node. See Figure 3 for examples of using the label hierarchy to store user generated meta data.

The nested set rules can be easily broken by the presence of duplicate images. It is not difficult to think of cases where an image is uploaded to Vibe by many different people, or of cases where the same image is uploaded many times by the same user. Currently Vibe supports only exact duplicate detection. This method of duplicate detection is brittle, and in future work we will extend this functionality to include detection of near-duplicate images.

Adding images to Vibe can be accomplished in multiple ways. Images can be captured from websites and brought into Vibe through a drag and drop interaction. In this case, the original source of the image is stored along with the user agnostic image meta data. Images can also be uploaded from a user’s computer, or they can be selected from storage applications such as Dropbox and Google Drive. When an image is added, multiple sizes of the image are stored, allowing for adjustable gallery viewing when curating the dataset. Static urls are generated for the original images and the resized versions. These urls can be used by external websites and annotation tools to render the images.

For users that are starting a new dataset project, Vibe provides all the capabilities needed to define the hierarchy and to start collecting and curating images. However, if data collection and organization has already started, then we want users to be able to seamlessly transition into Vibe. We have attempted to reduce the cost of migrating to Vibe by accepting the upload of compressed folders. The directories contained in these compressed folders will turn into nodes of the hierarchy, and the content found in these directories will be added to those nodes. The nested set rules will be applied at this stage and will attempt to mitigate the presence of duplicate images. This type of upload is also
a very efficient way to programmatically create a complex label hierarchy that would be tedious to do by hand.

The actual curation of a dataset requires the movement of images among the nodes of the hierarchy. Vibe attempts to make this easy by supporting multiple options for moving the images. Simple drag and drop interactions allow one or multiple selected images to be moved to other nodes in the hierarchy. A Short Term Memory panel has been created that allows easy access to nodes that will be receiving a majority of the content during a curation session, see Figure 1. Movement operations can also be done simply by specifying the target node, and users can operate on all the content in a node rather than just selections. Interactions to move images up in the label chain are also supported, and the user can view multiple sizes of the images while they are curating. For a given image, a user can examine a subset of the label hierarchy that represents the nodes that contain the image. This allows for efficient modification of an image’s labels without having to browse the individual nodes.

### 2.3.3 Snapshots

In order to make Vibe collaborative we needed to allow users to share their label hierarchies. We considered two different approaches to enable this kind of interaction. We could allow a label hierarchy to be viewed and manipulated by multiple people at the same time. This type of interaction would be similar to Google Drive’s collaborative document editing system. Or we could implement a system where hierarchies can be sent to other users, modified, and recombined at a later time. This type of interaction is similar to a revision system, such as Git, where there is a master branch of the hierarchy maintained by one particular user which can accept modifications and additions from other users. We have opted for the second approach, and we support this interaction through the use of Snapshots.
Figure 2.5. This figure shows the Snapshot interface. A user’s Snapshots are displayed in a table, with information specifying the name of the Snapshot, the date it was taken, the source of the Snapshot, whether it is public or private, and some basic size stats. Snapshots can be loaded back into a user’s work space at any time. Users will also use this interface to do large uploads of images. In this screenshot, the Caltech-101 dataset is being uploaded.
Snapshots are Vibe’s solution for data archiving and sharing. See Figure 5 for the Snapshot interface. A user can take a Snapshot of a hierarchy at any time, capturing all the necessary information required to reproduce the hierarchy. The data is stored in a compressed, internal notation that can be archived and then later restored to a user’s workspace. This means that Snapshots serve as a basic revision tool, allowing the progression of a dataset to be captured. Snapshots can also be shared with other users, similar to the way a Dropbox directory can be shared. Users receiving a Snapshot can load it into their workspace and perform any actions necessary on the data. Operations such as content intersections and unions from multiple nodes as well as node merging augment the sharing process to make it efficient to split up curation tasks and recombine them at a later time. Modified versions of Snapshots are the main way data is pulled out of Vibe. This modified Snapshot converts the internal notation into a representation that is convenient for a user to utilize on their local machines. URLs to the images are provided so that the images can be downloaded.

Besides sharing a hierarchy with specific users, Snapshots can also be made public so that the entire community can utilize and build upon the works of others. Once a Snapshot is taken, it can be flagged as public and will then appear on the user’s profile. Other users can search for public Snapshots and can simply load them into their workspace. For vision researchers, this will provide easy access to a lot of data and will hopefully allow them to quickly test a new idea.

2.3.4 Cloud Implementation

Vibe has been built as a web application that runs in the Amazon Web Services cloud infrastructure. Amazon EC2 provides the machines that run the various servers. Amazon RDS provides the database instances, and Amazon S3 provides storage for all of the content. Other AWS resources are utilized to bring performance and reliability to
Vibe.

Like any project, sustainability of Vibe will be contingent on its utility. We have funding to support a certain number of users and a certain amount of operational and storage costs during the next 3-5 years. Depending on the reception of Vibe by the vision community and by other potential users, e.g. application developers or non-vision researchers who deal with large collections of images, more long term funding sources will be sought.

2.4 Use Cases

We predict several different types of use cases for Vibe. As with most tools, the intended usage is not always the most prevalent usage, and we hope that Vibe’s flexibility provides a solution to situations we have not considered.

The most basic interaction with Vibe involves using it as cloud based image repository. Since each image is given a static url and a unique identifier, Vibe can be used as a simple, browsable image hosting service. Existing web based annotation tools, such as those built for Amazon Mechanical Turk, can use these urls to display the images and associate the annotations with the images’ identifiers.

Dataset curation is the intended usage of Vibe. We anticipate project managers creating the initial dataset hierarchy, loading in a set of images, and then collaborating with other members of the project to curate the images. See Figure 6 for an example of the collaboration process. There may be an easy way to programmatically jumpstart the curation process (e.g. from initial tag meta data), in which case the managers can upload a compressed file with the label hierarchy partially created. After initial curation of the images, some categories may need additional images. Project members can add new images to the system and share them with each other. As annotation tasks are performed on the images, additional label hierarchies can be created that allow for more analysis.
Figure 2.6. This figure shows an example of the collaboration process in Vibe. In Stage 1, the two collaborators have agreed upon a label hierarchy and begin collecting and curating images. Stage 2 represents curation progress and changing project requirements. New categories have been added, and an original category has been further refined. The label hierarchies of the two collaborators no longer match. In Stage 3, the two collaborators decide that it is time to sync back up. Collaborator 1 shares a Snapshot of their work with Collaborator 2 who then merges it into their label hierarchy. Vibe handles the dirty work of merging the two hierarchies with a single command. Collaborator 2 can then share this merged label hierarchy with Collaborator 1. The label hierarchies of the two collaborators now match and the collaborators are in Stage 1 again.
of the images. Project specific requirements can be enforced through this additional analysis, perhaps filtering out images, and the dataset can converge towards a cleaner version. This interaction will continue until the project requirements are met and each category has an acceptable number of images. A Snapshot can then be downloaded and given to a vision researcher or an engineer, who then has the necessary information to download the dataset, and look up image labels.

Vibe may prove to be a valuable addition to the workflow of vision researchers. Because data movement is so easy in Vibe and because it is easy to grab public Snapshots, a researcher could easily use Vibe to create a modified dataset that is relevant for their work. For example, entire categories could be removed from a dataset, or two independent datasets could be merged. Or a particular category in a dataset could be further refined. These modified datasets can then be re-shared with the community. Visualizing the results of an annotation task indirectly through the Vibe label hierarchy may also be a valuable tool for researchers. For example, if parts are collected for an image set, then pose labels could be generated and reviewed in Vibe. When testing a classification algorithm, the results can be uploaded and easily browsed in Vibe. If the images are already in Vibe, then these interactions are very fast.

2.5 Future Work

The initial implementation of Vibe was designed specifically for the curation of data by either a single person or few members of a collaborative team. This may seem very limited, but we have determined this to be a basic requirement of a computer vision project. We wanted a robust foundation to build upon and the current version of Vibe provides enough power and flexibility to begin creating and managing datasets. Large scale labeling relies on crowd sourced workers, and while the results of crowd sourced labeling tasks can be uploaded to Vibe for review right now, future additions to Vibe
will support creating and issuing crowd sourced curation tasks. A public facing API that can be accessed by annotation tools is also in the works. We imagine annotation tools querying Vibe for specific images to annotate based off of the label hierarchy.

### 2.6 Conclusion

We have developed Vibe to be a tool for anyone working on a project that utilizes image datasets. Vibe tackles the mundane, yet fundamental, tasks associated with dataset creation and maintenance, and has attempted to minimize their impact on a computer vision project. Vibe has modified the standard file system interactions to support efficient dataset operations allowing users to curate, analyze and modify large collections of images. Vibe supports collaborations by allowing data to be effortlessly shared through Snapshots, and because Vibe is hosted in the cloud, it can reliably store and scale with the increasing size of datasets.

Given that datasets are a fundamental part of computer vision, we hope that Vibe increases the accessibility and scope of computer vision by reducing the cost of dataset management. Hosting these datasets in a centralized location will allow the data to be efficiently utilized. The results for the vision community can only be positive. More datasets, along with the tools to interact with them, will reveal new, challenging problems whose solutions will ultimately advance the field.
Chapter 3

Fine-Grained Visual Categorization

3.1 Introduction

Fine-grained categorization, also known as subcategory recognition, is a rapidly growing subfield in object recognition. Applications include distinguishing different types of flowers [53, 54], plants [41, 2], insects [51, 42], birds [13, 22, 65, 73, 6, 74, 15, 43], dogs [56, 45, 55, 38], vehicles [64], shoes [5], or architectural styles [49]. Each of these domains individually is of particular importance to its constituent enthusiasts; moreover, it has been shown that the mistakes of state-of-the-art recognition algorithms on the ImageNet Challenge usually pertain to distinguishing related subcategories [62]. Developing algorithms that perform well within specific fine-grained domains can provide valuable insight into what types of models, representations, learning algorithms, and annotation types might be necessary to solve visual recognition at performance levels that enable practical applications.

Within fine-grained categorization, bird species recognition has emerged as one of the most widely studied areas (if not the most) in the last few years, in part due to the release of CUB-200 [68] and CUB-200-2011 [66] as standard datasets. Performance improvements on the CUB datasets over the last few years have been remarkable, with early methods achieving 10 – 20% 200-way classification accuracy [13, 66, 68, 71], and
recent methods achieving 55–65% accuracy [6, 25, 15, 74, 12, 19]. Here we report further accuracy gains up to 75.7%. Our improvements were greatly facilitated by the recent success of new deep convolutional neural network features [40, 19, 27, 34], and we study their integration with part-level alignment models and adaptation to fine-grained problems.

### 3.2 Related Work

Work on fine-grained categorization over the past 5 years has been extensive. Areas explored include feature representations that better preserve fine-grained information [71, 70, 51, 69], segmentation-based approaches [54, 14, 16, 15, 1, 25] that facilitate extraction of purer features, and part/pose normalized feature spaces [22, 6, 7, 65, 56, 73, 74, 45, 55, 29]. Among this large body of work, we will focus on analyzing some recent, state-of-the-art approaches [74, 12, 6, 29, 15, 25, 19].

Zhang et al. [74] utilize a weakly supervised deformable part model and then leverage semantic annotations at training time to model the correspondences between the latent parts of different components. The authors can then pool features across the predicted part boxes to create a global pose normalized model. They extract kernel descriptor features from 8 parts, stack them and use them for the final classification. The authors achieve a performance of 51% (bounding boxes are utilized at test time).

Branson et al. [12] use a strongly supervised deformable parts model to localize parts and a mixture model to handle different poses. Fisher features are extracted from the predicted part locations, stacked, and used for the final classification. The authors achieve a performance of 56.5% by utilizing a per class model with an attribute model (no extra information is used at test time).

Berg et al. [6] tackle fine-grained categorization by building a library of Part-base One-vs-One Features (POOFs). Each POOF utilizes the location of two parts, which are
transformed into a normalized position, and then a discriminative template is extracted around the parts. At test time, all POOFs are extracted from the given test image, and the resulting scores are stacked into a feature vector which is used for the final classification. Berg and Belhumeaur use the (non-parametric) part detection techniques from [3], where parts are localized using a Bayseian inference that combines the output of local detectors with a prior of the object shape. The authors achieve a performance of 56.8% (bounding boxes are utilized at test time).

Goring et al. [29] also utilize a non-parametric part detection technique that maps training image parts onto a test image. A HOG feature representation from the bounding box allows the authors to search the training images for similar examples. The parts from the k nearest examples are mapped onto the testing image, and used for feature extraction points. The authors achieve a performance of 57.8% (bounding boxes are utilized at test time).

Chai et al. [15] use a symbiotic part detection and segmentation system to extract pose normalized features. The authors link the GrabCut [61] segmentation algorithm with a multicomponent deformable part model [24] through a spatial saliency coupling. The resulting part locations and segmentation mask provide them with regions to extract features from. Fisher feature vectors are stacked together and used in the final classification. The authors achieve a performance of 61.0% (bounding boxes are utilized at test time).

Gavves et al. [25] utilize segmentation masks to generate a rough object shape that they use for alignment. Given the bounding box for the bird, the authors use GrabCut [61] to generate a segmentation mask and they fit an ellipse to the pixels of the segmentation mask. Using the gravity vector assumption [57] to produce a local frame of reference, the authors extract Fisher feature vectors from a fixed number of segments. The resulting feature vectors are stacked together and used for the final classification. The authors
achieve a performance of 62.7% (bounding boxes are utilized at test time).

Donahue et al. [19] use the same technique as Zhang et al. [74], the only difference being that they extract Decaf features rather than KDES. The authors achieve a performance of 65% (bounding boxes are utilized at test time).

There is a common thread that runs through the previous techniques. Namely, all of the authors follow a paradigm of detect → align → represent → classify. Parts of the object are first detected and then used to either warp the image itself or to guide the construction of a pose invariant feature vector. Finally, the features themselves are extracted and then classification is performed. We analyze the steps below:

1. Detection – There appears to be two general approaches for part detection. One approach uses DPMs [24] to learn a part model that can be used to detect the parts in a testing image [74, 12, 15, 19]. Another approach uses non-parametric techniques and utilizes the training data at test time to search for the most likely part configuration in the test image [6, 29].

2. Alignment – Once parts are localized they can be used in essentially two different ways. The parts can be used to warp the image itself into a normalized view from which fixed sized features can be extracted [6]. Alternatively, the parts can also be used to guide the construction of a pose invariant feature vector, where the vector is composed of individual feature vectors stacked in a specified order [74, 12, 29, 15, 25, 19].

3. Representation – The reason that parts are detected in the first place is that they should provide discriminative locations that are useful in distinguishing between categories of the same basic shape. Representing these locations in a suitable feature space requires transforming the pixel values, and the field has progressively created richer feature extractors. Simple histograms have given way to SIFT [46]
and HOG [17] features, which have given way to Fisher encoded vectors [59] and now CNN features [27, 19] appear to be the latest generation of features.

4. Classification – Once a suitable feature vector has been constructed, it appears that most authors use a simple linear SVM or logistic regression model for the classification task.

Table 3.2 summarizes the related work in this space and their choice of components.

As found in [27, 19], deep convolutional features appear to be incredibly rich feature representations. The impressive performance of deep convolutional networks [44] (CNNs) on large scale visual recognition challenges, ignited by [40], has motivated researchers to adapt CNNs that were pre-trained on ImageNet to other domains and datasets, including Caltech-101 [72], Caltech-256 [72], VOC detection [27], and VOC classification [72]. Donahue et al. [19] extracted CNN features from part regions detected using a DPM, obtaining state-of-the-art results in bird species classification. Our work is inspired by these results, and we improve on them by combining ideas inspired from fine-grained recognition and CNN research. In particular, we find that different layers of the CNN are appropriate for different levels of alignment. Secondly, we explore different methods for fine-tuning CNN weights on the CUB-200-2011 training set, inspired by techniques from Girshick et al. [27].

3.3 Pose Normalization Schemes

In this section, we define a class of pose normalization schemes based on aligning detected keypoints to the corresponding keypoints in a prototype image. In Section 3.3.2, we introduce an algorithm for learning a set of prototypes that minimizes the pixel-wise alignment error of keypoint annotations in a training set and works for arbitrary warping
3.3.1 Pose Normalization By Prototypical Regions

Let \( \{(X_i, Y_i)\}_{i=1}^n \) be a training set of \( n \) images and ground truth part annotations, where each annotation \( Y_i = \{y_{ij}\}_{j=1}^K \) labels the pixel location and visibility of \( K \) 2D keypoints in the image \( X_i \). Due to its simplicity and ease of collection, this style of 2D keypoint annotations is widely used (e.g, for birds [66], dogs [45], faces [32], and humans [10]).

Let \( \Psi(X, Y) = [\psi_p(X, Y)]_{p=1}^P \) be a feature vector that is obtained by concatenating \( P \) pose normalized feature spaces, where each \( \psi_p(X, Y) \) may correspond to a different part or region of an object and can be estimated using some subset of keypoints in \( Y \).

We consider a simple definition of \( \psi_p(X, Y) \) based on prototypical examples. Let the \( p \)-th prototype \( R_p = \{i_p, b_p, S_p\} \) consist of a reference image \( i_p \), a rectangle \( b_p \) defining a region of interest in \( X_{i_p} \) for feature extraction, and a set of keypoint indices \( S_p \). Given a test image \( X_t \) with detected keypoints \( Y_t \), we solve for the transformation \( W(y_{ij}, w) \) in some class of warping functions \( \mathcal{W} \) that best aligns the corresponding keypoints in \( Y_t \) to \( Y_{i_p} \):

\[
w_{i_p}^* = \arg \min_{w \in \mathcal{W}} \sum_{j \in S_p} E(y_{ij}, R_p, w), \quad \text{where} \quad E(y_{ij}, R_p, w) = \|\hat{y}_{i_p j} - W(y_{ij}, w)\|^2 \quad (3.1)
\]

where \( \| \cdot \| \) indicates Euclidean distance, and \( \hat{y}_{i_p j} \) is a version of \( y_{i_p j} \) after normalizing by the bounding box \( b_p \) (by subtracting the upper-left coordinate and dividing by the width/height). The induced pose normalized feature space \( \psi_p(X, Y) = \phi(X(w_{i_p}^*)) \) is obtained by applying this warp to the image \( X_t \) and then extracting some base feature \( \phi(X) \), where \( X(w) \) is a warped version of image \( X \).

In Table 3.1, we define how Eq 3.1 can be computed for many different warping functions.
Table 3.1. Computation of warping function \( W(y, w) \) from detected points \( Y_t[S] \) to a prototype \( Y_i[S] \) for different warping families. In the above notation, let \( M_t \) and \( M_i \) be \( 2 \times |S| \) matrices obtained by stacking points in \( Y_t \) and \( Y_i \), and \( \mu_t \) and \( \mu_i \) be their means. Let \( \bar{M}_t \) and \( \bar{M}_i \) denote mean subtracted versions of these matrices, and the superscript \( h \) denote points in homogeneous coordinates. Let \( C = U\Sigma V^\top \) be the SVD of \( C = \bar{M}_t \bar{M}_i^\top \).

<table>
<thead>
<tr>
<th>Name</th>
<th>( W(y, w) )</th>
<th>Solve ( w_{ip}^* )</th>
<th># Pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation</td>
<td>( y = y_t + T )</td>
<td>( T = \mu_i - \mu_t )</td>
<td>(</td>
</tr>
<tr>
<td>2D Similarity</td>
<td>( y = sRy_t + t )</td>
<td>( s = \frac{\text{tr}(\bar{M}_i^\top R\bar{M}_t)}{\text{tr}(\bar{M}_i^\top \bar{M}_t)} )</td>
<td>(</td>
</tr>
<tr>
<td>2D Affine</td>
<td>( y = Ay_t^h )</td>
<td>( A = M_iM_t^{h\top} (M_i^hM_t^{h\top})^{-1} )</td>
<td>(</td>
</tr>
<tr>
<td>Homography</td>
<td>( y^h = Hy_t^h )</td>
<td>4 point algorithm [30]</td>
<td>(</td>
</tr>
<tr>
<td>Thin-plate-spline</td>
<td>( y = \sum_j c_j \varphi(|y_{ij} - y_{ij}|) )</td>
<td>solve for ( c_i ) as in [9]</td>
<td>( S = S_p )</td>
</tr>
</tbody>
</table>

families, including simple translations, similarity transformations (2D rotation, scale, and translation), and affine transformations. The extension to other families such as homographies and thin-plate-splines [9, 4] is straightforward. The above transformations have simple closed form solutions and have well understood properties in approximating projective geometry of 3D objects. In each case, the applicable warping function is only well-defined if the number of points available \( |S| \) is sufficiently large. Let \( S \subseteq S_p \) be the subset of points in \( S_p \) that are visible as determined by detected keypoints \( Y_t \). If \( |S| \) falls below the applicable minimum threshold, we set the induced feature vector \( \psi_p(X, Y) \) to zero.

### 3.3.2 Learning Pose Prototypes

In this section, we introduce an algorithm for learning pose prototypes from training images with keypoint annotations. The approach has a similar objective to a poselet learning algorithm [10]. The main difference is that our approach generalizes to arbitrary warping schemes while explicitly optimizing pixel-wise alignment error in the induced feature space.
Figure 3.1. The top row visualizes different prototypes, each of which defines a region of interest and multiple keypoints that are used to estimate a warping. The bottom rows show the resulting warped regions $X(w_{tp}^*)$ when 5 images are aligned with each prototype. The 4 groupings of warped regions represent 4 baseline experiments analyzed in Table 3.3, which includes 1) Hand-Defined head or body regions, 2) the 1st 3 prototypes learned using our method from Section 3.3.2, 3) Rand-Pairs, which simulates [6], 4) CUB-Keypoints, which simulates [12]. In general, we see that a similarity transform captures the scale/orientation of an object better than a translation, while an affine transformation sometimes overly distorts the image. Using more points to estimate the warping allows for non-visible keypoints and ambiguous image flipping problems to be handled consistently.
Recall that $E(y_{tj}, R_p, w_{t,p}^*)$ is the squared pixel alignment error in the induced pose-normalized feature space when matching image $t$ to prototype $R_p$. We attempt to learn a set of prototypes $\{R_p\}_{p=1}^P$ that minimizes the alignment error under the constraint that each keypoint $y_{tj}$ in the training set must be aligned with low error to at least one prototype. The intuitive justification is that all portions of an object – each of which might contain important discriminative information – should be consistently aligned in at least one component of the feature space $\Psi(X, Y) = [\psi_p(X, Y)]_{p=1}^P$. Our goal is to choose a set of prototypes $R^*$ that optimizes the objective

$$
R^* = \arg \min_R \lambda P + \frac{1}{nK} \sum_{t=1}^n \sum_{j=1}^K \min_{p} E(y_{tj}, R_p, w_{t,p}^*)
$$

(3.2)

where the first term penalizes the number of prototypes selected and the second term minimizes pixel-wise alignment error, with $\lambda$ being a tweakable tradeoff parameter. The optimization problem can be complex due to the possibility of complex warping functions $w_{t,p}^*$ (Eq 3.1) and prototype definitions. To make the problem tractible, we consider candidate prototypes anchored by a keypoint in the training set ($nK$ candidates in total). Given an anchor $y_{ik}$ representing the $k$-th keypoint in image $i$, we define the candidate prototype $R_{ik} = \{i, S_{ik}, b_{ik}\}$ in terms of a set $S_{ik}$ of the $M$-nearest neighbors in $Y_i$ to $y_{ik}$, and $b_{ik}$ as an expanded bounding box around those keypoints.

We can solve Eq. 3.2 as a non-metric facility location problem [20]. Given predefined costs $c_{lm}$ of connecting city $l$ to facility $m$ and costs $o_m$ of opening facilities, the goal is to open a subset of facilities with minimum cost such that each city must be connected to one facility. Eq 3.2 reduces to a facility location problem, where each anchor point $y_{ik}$ is a candidate facility with open cost $\lambda$, and each keypoint $y_{tj}$ is a city that can be connected to a facility with cost $c_{tj,ik} = E(y_{tj}, R_{ik}, w_{t,ik}^*)$. A nice property of facility location problems is that, unlike some clustering algorithms like k-means,
a fast greedy algorithm [33] has good approximation guarantees (1.61 [33] when the
city-facility costs are metric, 1 + lnP [31] for non-metric costs). This algorithm requires
precomputing pairwise costs $c_{i,j,k}$ and sorting them. Examples of learned prototypes can
be seen in Fig 3.1.

3.4 Deep Convolutional Features

Our pose-warped image regions $\{X(w_{ip})\}_p$ are each fed into a feature extractor
$\phi(X)$, where $\phi(X)$ is the output of one or more layers of a deep convolutional neural
network (CNN) [40]. We use the network structure from Krizhevsky et al. [40].

In brief, a fixed input image region of size 224x224 is forward propagated
through 5 convolutional layers, followed by 3 fully connected layers. The last fully
connected layer outputs data to an N-way softmax function, where N is the number of
classes. Whereas a rush of papers have begun to adapt CNNs to different computer
vision problems with very compelling results [72, 19, 27, 75], many aspects of their
performance and training are still not well understood. In the next two sections, we
provide an empirical investigation of a couple of different aspects of CNN features that
to our knowledge have not been extensively studied yet.

3.4.1 Multi-Layer CNN Features For Different Alignment Models

The progression through the 8-layer CNN network can be thought of as a progres-
sion from low to mid to high-level features. The later layers aggregate more complex
structural information across larger scales–sequences of convolutional layers interleaved
with max-pooling are capable of capturing deformable parts, and fully connected layers
can capture complex co-occurrence statistics. On the other hand, later layers preserve less
and less spatial information, as max-pooling between each convolutional layer succes-
sively reduces the resolution of the convolutional output, and fully connected layers drop
semantics of spatial location. We thus hypothesize (and verify empirically in Section 3.5), that different layers of this pipeline are more appropriate for different alignment models, and combining multiple levels of alignment can yield superior performance. For example, an image-level alignment scheme, where features are extracted from the entire image, will work best with the last fully connected layers (as empirically verified in [40]). On the other hand, a pose-aligned region may work better with the earlier convolutional layers. Furthermore, combining multiple levels of alignment can yield superior performance. Image-level features include greater contextual information and remain unchanged when part detection is noisy. By contrast, pose-aligned features preserve finer-grained discriminative information but only work if part detection succeeds. Let $R_p$ be a region as defined in Section 3.3, and assume we associate $R_p$ with a feature layer $l_p$. We denote the corresponding feature vectors $\phi_{l_p}(X(w^*_r))$ as the output of the $l_p$-th layer of the CNN given the appropriate pose-warped image $X(w^*_r)$ as an input.

Our final feature space concatenates features from multiple regions and layers, and one-vs-all linear SVMs are used to learn weights on each feature. The use of an SVM (instead of the multiclass logistic loss used by CNNs) is primarily for technical convenience when combining multiple regions. To handle layers with different scales of magnitude, each CNN layer output is normalized independently during feature extraction. In the next section, we explore a few different approaches for training the internal weights of the CNN.

3.4.2 Training the Convolutional Neural Net

We consider 4 training/initialization methods:

**Pre-Trained ImageNet Model:** This corresponds to the methodology explored in [19], where the CNN is pre-trained on the 1.2 million image ImageNet dataset and used directly
as a feature extractor.

**Fine-Tuning the ImageNet Model:** This corresponds to the methodology explored in [27]. Here, the final 1000-class ImageNet output layer is chopped off and replaced by a 200-class CUB-200-2011 output layer. The weights of the new layer are initialized randomly, and stochastic gradient descent (SGD) and back propagation are used to train the entire network jointly with a small learning rate. Because the last layer is new and its weights are random, its weights are likely much further from convergence than the pre-trained ImageNet layers. Consequently, its learning is increased by a factor of 10.

**Two Step Fine-Tuning Method:** We explore a 2nd possible fine-tuning method that aims to avoid using unbalanced learning rates for different layers. Here, we use the same network structure as for the previous method. We use a two step process. In the first step, we fix the weights of the old ImageNet layers and learn the weights of the new 200-class output layer—this is equivalent to training a multiclass logistic regression model using the pre-trained ImageNet model as a feature extractor. It is a fast, convex optimization problem. This fixes the problem of initializing the new layer. SGD and back propagation are then used to jointly train all weights of the entire network, where each layer is given the same learning rate. To our knowledge, this initialization scheme has not yet been explored in earlier work.

**Training From Scratch:** The earlier three approaches can be seen as an application of transfer learning, where information from the ImageNet dataset has been used to train a better classifier on a different set of classes/images. To help differentiate between gains from more training data and the network structure of the CNN, we investigate training the CNN without ImageNet initialization. Weights are initialized randomly before training with SGD.
3.5 Experiments

We evaluate performance on the CUB-200-2011 dataset [66], a challenging dataset of 200 bird species and 11,788 images. The dataset includes annotations of 15 semantic keypoint locations. We use the standard train/test split and report results in terms of classification accuracy. Although we believe our methods will generalize to other fine-grained datasets, we forgo experiments on other datasets in favor of performing more extensive empirical studies and analysis of the most important factors to achieving good performance on CUB-200-2011. Specifically, we analyze the effect of different types of features, alignment models, and CNN learning methods. We believe that the results of these experiments will be informative and useful to researchers who work on object recognition in general.

Implementation Details: We used the DPM implementation from [11], which outputs predicted 2D locations and visibility of 13 semantic part keypoints. To learn pose prototype regions, we chose $\lambda = 8^2$, which means that a new prototype should be added if it reduces the average keypoint alignment error by 8 pixels. For our best classifier, we concatenated features extracted from each prototype region with features extracted from the entire image.

We used the Caffe code base from Jia [34] to extract, train, and fine-tune the CNN with the default structure and parameter settings. When extracting feature outputs from different CNN layers, we use the names $conv3$, $conv4$, $conv5$, $fc6$, and $fc7$, where $conv$ denotes a convolutional layer, $fc$ denotes a fully connected layer, and the number indicates the layer number in the full CNN. We appended these names with the suffix -ft to denote features extracted on a CNN that was fine-tuned on CUB-200-2011. To fine-tune the CNN, we set the base learning rate to 0.001.
3.5.1 Summary of Results and Comparison to Related Work

Table 3.2 summarizes our main results and comparison to related work. Our fully automatic approach achieves a classification accuracy of 75.7%, a 30% reduction in error from the highest performing (to our knowledge) existing method [19]. We note that our method does not assume ground truth object bounding boxes are provided at test time (unlike many/most methods). If we assume ground truth part locations are provided at test time, accuracy is boosted to 85.4%. These results were obtained using prototype learning using a similarity warping function computed using 5 keypoints per region, CNN fine-tuning, and concatenating features from all layers of the CNN for each region.

We attempt to categorize each related method according to part localization scheme, features used, and learning method. See the caption of Table 3.2 for details. The major factors that we believe explain performance trends and improvements are summarized below:

1. Choice of features caused the most significant jumps in performance. The earliest methods that used bag-of-words features achieved performance in the 10 – 30% range [66, 73]. Recently methods that employed more modern features like POOF [6], Fisher-encoded SIFT and color descriptors [59], and Kernel Descriptors (KDES) [8] significantly boosted performance into the 50 – 62% range [6, 15, 25, 74, 12]. CNN features [40] have helped yield a second major jump in performance to 65 – 76%.

2. Incorporating a stronger localization/alignment model is also important. Among alignment models, a similarity transformation model fairly significantly outperformed a simpler translation-based model. Using more keypoints to estimate warpings and learning pose regions yielded minor improvements in performance.

3. When using CNN features, fine-tuning the weights of the network and extracting
Table 3.2. Our method significantly outperforms all earlier methods to our knowledge, both in terms of fully automatic classification accuracy (top grouping), and classification accuracy if part locations are provided at test time (bottom grouping). We categorize each method according to 4 axes which we believe significantly affect performance: 1) Level of automation, where column 2-3 indicate whether or not parts or object bounding boxes are assumed to be given at test time, 2) Part localization scheme (column 4), using the naming scheme Transformation-X-Y, where Transformation indicates the image warping function used (see Table 3.1), X indicates the number of keypoints/base-parts used to warp each region, and Y indicates the number of pose regions used, 3) Type of features (column 5), and 4) Learning algorithm (column 6), where CNN-FT is short for CNN fine-tuning.
<table>
<thead>
<tr>
<th>Method</th>
<th>Oracle Parts</th>
<th>Oracle BBox</th>
<th>Part Scheme</th>
<th>Features</th>
<th>Learning</th>
<th>% Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>POOF [6]</td>
<td>✓</td>
<td></td>
<td>Sim-2-131</td>
<td>POOF</td>
<td>SVM</td>
<td>56.8</td>
</tr>
<tr>
<td>Symbiotic [15]</td>
<td>✓</td>
<td></td>
<td>Trans-1-1</td>
<td>Fisher</td>
<td>SVM</td>
<td>61.0</td>
</tr>
<tr>
<td>DPD [74]</td>
<td>✓</td>
<td></td>
<td>Trans-1-8</td>
<td>KDES</td>
<td>SVM</td>
<td>51.0</td>
</tr>
<tr>
<td>Decaf [19]</td>
<td>✓</td>
<td></td>
<td>Trans-1-8</td>
<td>CNN</td>
<td>Logistic Regr.</td>
<td>65.0</td>
</tr>
<tr>
<td>NPT [29]</td>
<td>✓</td>
<td></td>
<td>Trans-1-15</td>
<td>Fisher</td>
<td>SVM</td>
<td>57.8</td>
</tr>
<tr>
<td>CUB [66]</td>
<td>✓</td>
<td></td>
<td>Trans-1-15</td>
<td>BoW</td>
<td>SVM</td>
<td>10.3</td>
</tr>
<tr>
<td>Visipedia [12]</td>
<td></td>
<td></td>
<td>Trans-1-13</td>
<td>Fisher</td>
<td>SVM</td>
<td>56.5</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td></td>
<td></td>
<td>Sim-5-6</td>
<td>CNN</td>
<td>SVM+CNN-FT</td>
<td><strong>75.7</strong></td>
</tr>
<tr>
<td>CUB Loc. [66]</td>
<td>✓</td>
<td>✓</td>
<td>Trans-1-15</td>
<td>BoW</td>
<td>SVM</td>
<td>17.3</td>
</tr>
<tr>
<td>POOF Loc. [6]</td>
<td>✓</td>
<td>✓</td>
<td>Sim-2-131</td>
<td>POOF</td>
<td>SVM</td>
<td>73.3</td>
</tr>
<tr>
<td><strong>Ours Loc.</strong></td>
<td>✓</td>
<td>✓</td>
<td>Sim-5-6</td>
<td>CNN</td>
<td>SVM+CNN-FT</td>
<td><strong>85.4</strong></td>
</tr>
</tbody>
</table>
Figure 3.2. CNN features significantly outperform HOG and Fisher features for all levels of alignment (image, bounding box, head).

features from mid-level layers yielded substantial improvements in performance beyond what had been explored in [19].

We support these conclusions by performing lesion studies in the next 3 sections.

3.5.2 Comparing Feature Representations

We performed experiments to quantify the effect of different image features and their performance properties with different alignment models. We compare CNN features from different layers to HOG [17] and Fisher-encoded [59] color and SIFT features while controlling for other aspects of our algorithms. HOG is widely used as a good feature for localized models, whereas Fisher-encoded SIFT is widely used on CUB-200-2011 with state-of-the-art results [15, 25, 12]. For HOG, we use the implementation/parameter settings of [24] and induce a $16 \times 16 \times 31$ descriptor for each region type. For Fisher features, we use the implementation and parameter settings from [12]. We summarize the results below:
Figure 3.3. Comparing classification performance for different CNN layers and regions if we assume ground truth part locations are known at test time, we see that 1) features extracted from the head (yellow tube) significantly outperform other regions, 2) The later fully connected layers (fc6 & fc7) significantly outperform earlier layers when a crude alignment model is used (image-level alignment), whereas convolutional layers (conv5) begin to dominate performance as we move to a stronger alignment model (from image → bbox → body → head), 3) Using a similarity warping model significantly outperforms a translation model (width of the red and yellow tubes), and slightly outperforms an affine model, 4) Using more points (from 1 to 5) to estimate the warping improves performance for the body, whereas 2 points is sufficient for the head.
CNN features significantly improve performance: In Fig 3.2, we see that CNN features significantly outperform other features for all levels of alignment, 57.3% vs. 28.2% for image-level features, and 78.4% vs. 58.1% for a similarity-aligned head. HOG performs well only for aligned regions (the head), while Fisher features perform fairly well across different levels of alignment.

Different layers of the CNN are appropriate for different alignment models: In Fig. 3.3, we see that the later fully connected layers of the CNN (fc6 & fc7) significantly outperform earlier layers when a crude alignment model is used (57.3% vs 42.4% for image-level alignment), whereas convolutional layers (conv5) begin to dominate performance as we move to a stronger alignment model (from image → bbox → body → head).

3.5.3 Comparing Part Localization Schemes

We next perform experiments to quantify the effect of our pose normalization scheme, including the effect of the type of warping function used, a comparison of different methods of combining multiple pose regions, and the effect of imperfect part detection.

A similarity alignment model works best: In Fig. 3.3, we compare the effect of different choices of warping functions (translation, similarity, and affine) and the number of keypoints used to estimate them. We see that a similarity warping model significantly outperforms a translation model and slightly outperforms an affine model (on the head region, 74.8% for similarity vs. 65.2% for translation vs. 73.3% for affine). Secondly, we see that using more points (from 1 to 5) to estimate the warping improves performance for the body, whereas 2 points is sufficient for the head.
Figure 3.4. If ground truth parts were available at test time or part detection could be improved, performance would be improved significantly (width of red/yellow tubes).

Figure 3.5. Fine-tuning significantly improves performance for all alignment levels (width of each tube). Improvements occur for all CNN layers; however, the effect is largest for fully connected layers.
Figure 3.6. The same fine-tuning effect holds for automated part prediction.

**Combining multiple regions improves performance:** In Table 3.3, we compare different strategies for combining multiple pose regions. We note that combining multiple regions improves performance over the best single region: 85.4% vs. 78.4% for the head. We compare to several different baseline methods for inducing a multi-region feature space while keeping our feature implementation fixed. The Proto-Learn method employs our pose learning scheme from Section 3.3.2 using a similarity warping model and slightly outperforms other methods while being compact. Rand-Pairs simulates the alignment method used by POOF [6], where random pairs of keypoints induce similarity-aligned regions. CUB-Keypoints simulates the method used by [12] (among others), where each detected keypoint directly induces a surrounding pose region. Head-Body represents a baseline of expert-defined regions, and concatenates hand-defined similarity-aligned head and body regions with image and bounding box features.
Table 3.3. Comparing Different strategies for combining multiple regions when part locations are given at test time. The number in parentheses indicates the number of regions used for each method.

<table>
<thead>
<tr>
<th>Head Body (2)</th>
<th>Proto-Learn (6)</th>
<th>Rand-Pairs (6)</th>
<th>Rand-Pairs (30)</th>
<th>CUB-KP (13)</th>
</tr>
</thead>
<tbody>
<tr>
<td>83.7</td>
<td>85.4</td>
<td>83.2</td>
<td>84.1</td>
<td>79.6</td>
</tr>
</tbody>
</table>

**Imperfect part detection causes a significant but manageable drop in performance:**

In Fig. 3.4, we visualize the drop in performance caused by using detected parts vs. ground truth parts for different regions, which ultimately results in a drop in performance from 85.4% to 75.7%. This is a sizeable drop in performance that we hope to reduce in future work by improving our part detection method; however, this gap is also surprisingly small, in large part due to the excellent performance of CNN features on image-level features.

### 3.5.4 Comparing CNN Learning Methods

In this section, we compare different strategies for learning the internal weights of the CNN.

**Fine-tuning CNN weights consistently improves performance:** In Fig. 3.5-3.6, we compare performance when using the pre-trained ImageNet model as a feature extractor vs. fine-tuning the ImageNet model on the CUB-200-2011 dataset (see Section 3.4.2 for details). We see that fine-tuning improves performance by 2 – 10%, and improvements occur for all region types (image, bounding box, head, body), all CNN layers, and both on predicted and ground truth parts.

**ImageNet pre-training is essential:** The default CNN implementation was pre-trained on ImageNet and performance improvements come in part from this additional training data. We tried training the same network structure from scratch on the CUB-200-2011
dataset over 5 trials with random initialization. Performance was significantly worse, with 10.9% and 54.7% accuracy on image-level and similarity-aligned head regions, respectively (compared to 57.0% and 78.6% performance with pre-training). The problem relates to overfitting—the CNN model has 60 million learnable parameters [40] and the CUB-200-2011 dataset has < 6000 training images. Learning converged to near zero training error for both fine-tuning and training from scratch.

The two step fine-tuning method yields more reliable improvements: Over 5 random trials, our proposed 2-step fine-tuning method improved average accuracy on both the image and head regions by about 2% compared to the method used in [27] (57.0% and 78.6% compared to 55.1% and 76.9%).

3.6 Conclusion

In this paper, we reduced the error rate on CUB-200-2011 by 30% compared to previous state-of-the-art methods, and analyzed which design decisions were most important to achieving good performance. Our method is based on part detection and extracting CNN features from multiple pose-normalized regions. Performance improvements resulted in large part from 1) using CNN features that were fine-tuned on CUB-200-2011 for each region, 2) using different CNN layers for different types of alignment levels, 3) using a similarity-based warping function that is estimated using larger numbers of detected keypoints. We also introduced a novel method for learning a set of pose regions that explicitly minimizes pixel alignment error and works for complex pose warping functions. In future work, we hope to apply our methods to other fine-grained datasets and explore customized CNN network structures and their training.

This chapter, in part, has been submitted for publication of the material as it may appear in British Machine Vision Conference, 2014, Branson, Steve; Van Horn, Grant;
Belongie, Serge; Perona, Pietro. The thesis author and Steve Branson contributed equally to this paper.
Chapter 4

Conclusion

The task of fine-grained categorization poses many challenges for vision researchers. Not only is the algorithmic task of separating highly similar subordinate categories difficult, it is challenging to simply collect the necessary data to start working on the algorithmic task. In this work I have presented two solutions to this dual problem of fine-grained categorization.

Vibe is a software infrastructure designed to ease the collection and management of image datasets. It was created to be used by taxon communities to build the label hierarchy and host the exemplar images needed to train the computer. Vibe utilizes a unique file system interface to make interacting with the label hierarchy easier, and provides a set of interactions and tools to make collaborating on a dataset easy and efficient. Data can then be exported from Vibe and given to researchers so that they can begin to work on the algorithmic task.

Once the data has been collected, there are many ways to train a computer to perform the categorization task. In this work I have present a new approach that utilizes pose normalized deep convolutional nets. This technique has significantly advanced the state-of-the-art on the CUB-200-2011 dataset [66], and closes the gap between expert human recognition and machine recognition.

The contributions of this work form the basic building blocks of a generic, scalable
fine-grained categorization system. In future work, these pieces will play a major role in the development of Visipedia [58]. With Visipedia, I hope to expand to many community-driven fine-grained categorization tasks, and ultimately bring the utility of computer vision to a broader audience.
Bibliography


[62] Olga Russakovsky, Jia Deng, Zhiheng Huang, Alexander C Berg, Li Fei-Fei, and UNC Chapel Hill. Detecting avocados to zucchinis: what have we done, and where are we going?


