The "What"-"Where" Network: A Tool for One-Shot Image Recognition and Localization

Daniel true Hurlburt
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Physical Sciences and Mathematics Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/9366

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.
The “What”-“Where” Network: A Tool for One-Shot Image Recognition and Localization

Daniel Hurlburt

A thesis submitted to the faculty of Brigham Young University in partial fulfillment of the requirements for the degree of Master of Science

David Wingate, Chair
Jacob Crandall
Michael Jones

Department of Computer Science
Brigham Young University

Copyright © 2020 Daniel Hurlburt
All Rights Reserved
ABSTRACT

The “What”-“Where” Network: A Tool for One-Shot Image Recognition and Localization

Daniel Hurlburt
Department of Computer Science, BYU
Master of Science

One common shortcoming of modern computer vision is the inability of most models to generalize to new classes—one/few shot image recognition. We propose a new problem formulation for this task and present a network architecture and training methodology to solve this task. Further, we provide insights into how careful focus on how not just the data, but the way data presented to the model can have significant impact on performance. Using these methods, we achieve high accuracy in few-shot image recognition tasks.

Keywords: one-shot image recognition, semantic segmentation, gradient
ACKNOWLEDGMENTS

Special thanks to Dr. David Wingate for invaluable assistance and support as well as many others here unmentioned for their assistance, both direct and indirect.
Table of Contents

List of Figures vi

List of Tables vii

1 Introduction .................................................. 1
2 Methodology .................................................. 2
  2.1 Network Architecture & Topology ......................... 2
  2.1.1 The “What” Network ................................ 3
  2.1.2 The “Where” Network ................................ 4
2.2 Dataset ...................................................... 4
  2.2.1 Train, Refinement, & Test Splits ..................... 4
  2.2.2 Properties & Considerations of Dynamic Dataset Generation 5
    2.2.2.1 Effective Size of the Dataset .................. 5
    2.2.2.2 Class Weighting ................................ 5
    2.2.2.3 Implied Regularization of Multi-Class Images ... 6
    2.2.2.4 Actual Regularization of Random Cropping .... 6
3 Gradient Enhancement & Training ............................. 7
  3.1 Positive/Negative Balancing ............................ 7
  3.2 Embedding Reliance Enforcement ........................ 7
  3.3 Reducing Possible “Noise” in the Dataset (e.g. Small Segmentation Elimination) ......................... 8
  3.4 Embedding Consistency Enforcement ..................... 8
**List of Figures**

1. Diagram of the “What/Where” Network .................................. 3
2. Class splits used for training. .............................................. 4
3. An example of the image object pairs which could be used in training. In this example the *bird* and *boat* images essentially represent “not cat”, and are any image with *no cats* and could therefore be images of *boats* or *birds*. The highlighted portions show the reversal of the positive object/image pairs 9
4. True Positive Accuracy, True Negative Accuracy, and IoU over Time .................................. 12
5. Examples of impressive performance on never before seen classes .................................. 13
6. Examples of ground-truth correction ........................................ 14
7. Examples of overt mistakes .................................................. 15
8. Examples of incomplete segmentations ...................................... 15
9. Examples of identifying zebras and elephants. Neither class was seen during training. Ex. 1 is a case where the algorithm failed to identify anything; Ex. 2 is a case where the algorithm identified an object it should not have; and Ex. 3 shows correct behavior .................................. 16
10. Examples of possible dataset memorization by the model .................. 17
11. Examples showing dataset errors .......................................... 18
12. Embedding space visualizations of classes by super class .................. 20
13. Embedding space visualizations by super class .......................... 21
List of Tables

1. The overall accuracy of the model ......................................................... 11
2. Results of a closer analysis of the model’s output. Images were classified via human judgement. ................................................................. 12
1 Introduction

Image recognition[4, 8, 12, 24] and semantic segmentation[1, 7, 18, 19, 21] are two of the main areas of focus in computer vision. However, they both suffer from the same pitfall—networks which perform these tasks are often restricted to performing on a certain set of pre-determined classes on which they have been trained. Generally, even the most effective models cannot perform well on classes on which they have not received extensive training. To further complicate matters, models which have been trained on some set of classes can rarely be trained further on some new class without jeopardizing that which has already been learned—a phenomenon known as catastrophic forgetting. And yet we live in a very dynamic world. We often encounter new objects; and so having to limit models to only working on pre-determined classes requires both the foreknowledge of what is relevant and the requirement that what is relevant will never change—that one will never desire to operate on images outside of that set of classes.

We propose a network which performs both image recognition and object-of-interest semantic segmentation conjunctively, facilitated through the creation and interpretation of an image embedding. Our overall neural network topology is what we term a “What”-“Where” Network. An image is passed into the “What” network along with a segmentation of the object-of-interest. This network is tasked with producing an embedding of the image and the object-of-interest. The “Where” network is then responsible for taking some new image and the image embedding from the “What” network and interpreting it to produce a semantic segmentation of all instances of the object-of-interest in the new image—if there are any.

In this way strict notions of class and class number limitations stemming from fixed-size output vectors/matrices are removed, and the network instead relies on the extremely flexible embedding space to denote and describe image classes. Therefore, because the “What” network seeks to represent the image and object-of-interest in the embedding space based on general image features, the network is ostensibly a general-purpose image recognition
and segmentation model—one that is able to perform recognition and segmentation tasks on never before seen classes.

Our contributions are three-fold. First, we contribute a new problem formulation for one-shot image recognition and localization. The idea of a “What”-“Where” network is, to the best of our knowledge, novel. Second, we show how a novel network topology can be built from common network architectures to help solve this problem. In theory, our method can be used with any state-of-the-art semantic segmentation network. Our primary technical contribution is the our third contribution: while the problem formulation and network architecture is fairly straightforward, because we train both networks jointly in an end-to-end manner, standard training algorithms require significant modification to effectively train the “What”-“Where” network. We contribute several non-trivial insights, which we categorize as dataset manipulation and gradient enhancement, which facilitate effective training.

While not perfect, our final results demonstrate that the “What”-“Where” network can successfully perform semantic segmentation on a variety of never-before-seen classes, given only a single example instance of the new class. Furthermore, our method can also segment multiple instances of the target object.

2 Methodology

We now describe specifics regarding the model architecture, the dataset, and training methodology.

2.1 Network Architecture & Topology

The topology of this network is composed of two parts, introduced earlier as the “What” and “Where” networks which perform complementary tasks to achieve the desired result. Training requires four images: two natural images and two segmentations. All four are taken
from the training set. We refer to these as the query image, and query segmentation, and the sample image and sample segmentation. The “What” network takes some query image and a corresponding segmentation of some object-of-interest in that image and produces an embedding. The “Where” network interprets that embedding and produces a segmentation of instances of that object-of-interest in some new sample image. These networks are more fully detailed below. A high level architecture is shown in Figure 1.

2.1.1 The “What” Network

The “What” network is a ResNet[8] with some modifications. The final classification layer is removed, and the output is flattened to produce a vector. To allow the network to take in an image concatenated with a segmentation, the initial layer was modified to accept a 4-channel image instead of a standard 3-channel image. One additional convolutional layer was put before the typical ResNet as well as two fully connected layers and three convolutional layers which were placed after to provide additional computation.

While the model was able to generalize to new classes with remarkable accuracy, the relative drop in accuracy from the training classes to the test classes indicated possible over-fitting in the embedder. To help combat this, we added noisy linear and noisy convolution layers as a means of regularization.
2.1.2 The “Where” Network

The “Where” network is a partially modified U-Net.\cite{21} It takes in a typical image and passes it through the network. However, at what has been termed the “bottom of the U”, i.e. after the final down-sampling layers and before the first up-sampling layers, the embedding from the “What” network is concatenated onto the output of the final down-sampling layer and is then passed through two convolutional layers to both process the image and reduce the number of channels to the expected amount. The rest of the network is unmodified.

2.2 Dataset

In our experiments we used the COCO 2017\cite{17} dataset, which contains over 200,000 images and 80 distinct classes. Further, annotations are provided which contain the segmentations for each instance of the various classes in each image. More specific considerations for our use and manipulation of the dataset are given in Figure 2.

2.2.1 Train, Refinement, & Test Splits

We subsetted the data by class into 3 distinct training sets: \textit{training\_set\_1}, \textit{training\_set\_2}, and \textit{training\_set\_3}, with the latter two containing 7 classes each and \textit{training\_set\_1} containing the remaining 66 classes. Training was done primarily on \textit{training\_set\_1}, with some training later
performed on *training set* 2 to help give new distinct examples and refine the embeddings, followed by more training on *training set* 1. We performed evaluation on *training set* 3, which contained classes never before seen by the model. Descriptions each training set and the classes they contain are given in Figure 2.

### 2.2.2 Properties & Considerations of Dynamic Dataset Generation

While many datasets are fixed, where some certain set of images are always presented once before another can be presented twice, or always shown in the same order, we used a dynamically generated dataset which would vary both in the order in which images were shown, but also which images were shown. Our dataset generates image pairs by first randomly selecting an object class, and then randomly selecting some image from that class. We then randomly select another image that contains that object and one that does not. This creates a single pair of images, representing a positive and negative example—whose motivation is described further in section 3.

#### 2.2.2.1 Effective Size of the Dataset

Because of the way the dataset is dynamically generated with random pairs of images—and the fact that images can represent more than one class—the effective size of the dataset was much larger than simply the number of actual images. And particularly in classes with high representation, the likelihood of being given the same pair twice is very low.

#### 2.2.2.2 Class Weighting

Unlike most standard image recognition datasets, COCO is not evenly weighted by class. Instead certain classes, such as *people*, far outweigh others, such as *hairdryer*. Thus there was a need to weigh the selection of classes when choosing images. If left unchecked, the network would see examples of *people* 33% of the time and examples of more obscure classes like hair dryers only 0.01% of the time. Contrariwise, if each class was sampled equally, the
model would likely over-fit on those classes which have poor representation and effectively
discard large amounts of the dataset by perhaps never using certain images from the classes
with high numbers of examples. To attempt to strike a balance, we weighted classes with the
simple formula $W = X + 1.5 \times \frac{\sum X}{|X|}$, where $X$ is an array containing the number of instances
of each class in the dataset. $W$ was then normalized into a probability distribution.

### 2.2.2.3 Implied Regularization of Multi-Class Images

It is also important to note that a single image may be a constituent of several classes. For
example, in the COCO dataset there are several images which contain a person, a dog, a
horse, and a cow. Thus the model is largely unable to create simple mappings between images
and outputs because most images contain examples of several classes. Thus the network can
be shown the same image twice and once be asked to show the dog, and the next instance
be asked to show the cow. In fact, in testing we took two of these images with instances
of dogs, people, horses, and cows and the network was able to identify each distinct object
successfully when called upon. It is noted, however, that these classes and images were all
present during training. Still, this does show that the model was able to interpret and rely
on the embedding given by the “What” network.

### 2.2.2.4 Actual Regularization of Random Cropping

In addition to the more subtle regularization provided by a dataset comprised of examples
containing multiple classes in a single image, more overt regularization was provided in the
form of random cropping, which crops and resizes randomly selected portions of the image.
This is a common technique and has been shown to be very effective.[3] For our purposes we
parameterized the random crop with a scale of 0.6 to 1.0, meaning that the cropped image
would contain anywhere between 60-100% of the original image and an adjusted aspect ratio
which is between $\frac{3}{4}$ and $\frac{4}{3}$ of of the aspect ratio of the original image.
3 Gradient Enhancement & Training

While our problem setting and topology are both fairly straightforward in some ways, we found that standard supervised learning with randomly sampled examples was insufficient to train the network. Initial experiments make it clear that care had to be taken in how the images were presented to the network. What follows is a series of non-trivial modifications of the way our training data was sampled and presented to the network.

3.1 Positive/Negative Balancing

If the network was simply shown several images and their segmentations, it would be reasonable to think that the “Where” network could simply learn to produce segmentations of the most prominent aspects of an image and ignore the embedding of the “What” network, which is supposed to parameterize the “Where” network. To help combat this, the gradient had to be “balanced” with negative examples where the object-of-interest given to the “What” network was not to found in the new image given to the “Where” network, thus strongly decentivizing the “Where” network to simply produce segmentations of the most prominent objects in an image and therefore encouraging it to learn from and utilize the embedding from the “What” network. Practically, our implementation leveraged carefully constructed batches that contained identical numbers of positive and negative training examples.

3.2 Embedding Reliance Enforcement

Another key empirical observation concerns “image pairing”. Again, to help improve gradients and force the network to rely on the “What” image embeddings, each image presented to the “Where” network would be represented twice—once paired with an object-of-interest found in that image and once paired with an object-of-interest *not* found in that image. This presents the “Where” network with the same input image two times, but asks the network to output a different result for each, forcing the network to utilize the embedding from the “What” network to correctly determine the distinction. This again reinforces the importance
of the embedding and helps to further neutralize the problem mentioned above of the model simply producing a segmentation of prominent objects in that image.

### 3.3 Reducing Possible “Noise” in the Dataset (e.g. Small Segmentation Elimination)

When reviewing the examples from training, we noticed many instances where the segmentations—in either the sample object or the sample image—contain many very small patches of object segmentation; so small that features of those objects were largely indistinguishable. Because it seemed that these training examples would provide little or no meaningful information to the network—yet still affect the weights—they could be a detriment to successful learning. Thus we removed all images where the object segmentations were smaller than 7.5% of the overall image. In this way, the training examples would have more information and improve training accuracy. Implementing this improved the accuracy markedly. Nearly immediately after reducing the dataset in this way, accuracy jumped from 30% to 55%. While it may seem that these results stemmed from an increased ability to memorize the training set, examples produced show that the learning achieved by the model extends beyond simple memorization. It also showed that the model was able to learn with these images in, but when presented with them—while it may have learned proper representations of those given classes—was unable to produce satisfactory results, a sort of “I can’t make out what that is” problem.

### 3.4 Embedding Consistency Enforcement

The final use of gradient enhancement came with the realization that because in positive examples the images given to both the “What” and “Where” networks contain the same object, their respective positions may be switched. Thus you can add the “What” and “Where” images in the other position as well, helping to reinforce that these two objects are the same and the embedding should reflect that. This boosted training accuracy another
Figure 3: An example of the image object pairs which could be used in training. In this example the bird and boat images essentially represent “not cat”, and are any image with no cats and could therefore be images of boats or birds. The highlighted portions show the reversal of the positive object/image pairs

15 percentage points for a training accuracy over 70%. As a result, we had a dynamically generated dataset which consisted of randomly selected image pairs according to the above criteria. An example is diagrammed in Figure 3

3.5 Comments on Gradient Enhancement

Using these gradient enhancement methods greatly increased the efficacy of the model and underscored the importance of not simply ensuring that information is being passed back through the gradient, but that the right information is being passed back. In our case it was not enough to simply collect the data, prepare it, and send it through the network; rather, careful thought had to be given as to what information was being passed back and how that could be improved. As the gradient is the only information received by the model, its importance can be hardly overstated and it deserves (and in this case required) careful consideration.
4 Results & Examples

Training was completed over 10 days and 1.6 million iterations (though less likely would have sufficed). A graph of True Positive Accuracy, True Negative Accuracy, and IoU over training iterations is given in Figure 4. Because this is a somewhat unique problem formulation, comparisons to baselines can be difficult. While there have been studies of one-shot learning, none of them worked in this way, so while we give examples of baselines from other studies, we understand that they are not the sort of direct comparison one sees in analyzing performance on, say, the ImageNet Competition.

4.1 Baselines

By leveraging text data and image data, researchers at Google in their DeVise[6] paper were able to perform image recognition on 1,589 new labels which were within 2 hops of similarity with the training set with an accuracy of 6%. Increasing the novelty of the new labels to 3 hops and 7,860, they achieved and accuracy of 1.7%. In getting a model to select another instance of the same novel character from a set of 20 possible characters, Lake et al.[13] produced a Bayesian model which exceeded human performance, with an error rate of just 3.3%. These show the abilities of other models across different one-shot learning tasks.

4.2 Accuracy & Performance

The accuracy achieved on never before seen classes was quite good. For the purposes of discussion and we break them up into two parts: official accuracy and human-analyzed accuracy.

4.2.1 Cannonical Accuracy & Performance

Here we describe the pixel-wise accuracy of the model computed during testing on new images, using standard metrics. After training the model achieved accuracies as shown in Table 1. We report true positive, true negative, false positive, false negative, and IoU. The overall
accuracy of the method is quite good; examples of segmentations are shown in Figures 5 and 9.

<table>
<thead>
<tr>
<th></th>
<th>true positive</th>
<th>true negative</th>
<th>false positive</th>
<th>false negative</th>
<th>IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>88.2%</td>
<td>35.1%</td>
<td>95.2%</td>
<td>4.6%</td>
<td>66.7%</td>
</tr>
</tbody>
</table>

Table 1: The overall accuracy of the model

4.2.2 Human-Analyzed Accuracy & Performance

However, by taking a closer look at the images produced, we were able to get a more nuanced view of the model’s performance. It became clear that there were several possible outcomes.

For positive examples where it *should* produce a segmentation of some object, the model could:

- Identify the Wrong Class: Produce a segmentation, but of the incorrect object
- No Output: Produce no segmentation
- Partial Segmentation: Produce a segmentation, but one that is incomplete or very fuzzy
- Perfect Segmentation: Produce a segmentation that is very close to the ground truth

For negative examples where it *should not* produce a segmentation of some object, the model could:

- Identify the Wrong Class: Produce a segmentation
- Produce a True Negative: Produce no segmentation

Table 2 shows the results of this closer analysis. By adding instances of correct output, we show the model has an accuracy of 51.53% (62.87% if you include those for which it was close) and an error of 37.1%. This more fine-grained analysis of the output of the model shows remarkable performance on generalization to unseen classes.
Figure 4: True Positive Accuracy, True Negative Accuracy, and IoU over Time

<table>
<thead>
<tr>
<th>Identified</th>
<th>Positive Examples</th>
<th>Negative Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrong Class</td>
<td>No Output</td>
<td>Partial Segmentation</td>
</tr>
<tr>
<td>5.15%</td>
<td>14.43%</td>
<td>11.34%</td>
</tr>
</tbody>
</table>

Table 2: Results of a closer analysis of the model’s output. Images were classified via human judgement.

### 4.3 Positive Results

Looking at the output of the model confirmed what was shown above and demonstrated impressive results. The model was able to perform successful and impressive generalization to new object classes remarkably well. At times, the segmentations produced are nearly identical to the objects in the image, and at times, as seen with the kite, sometimes they are even better than the actual ground truth in that they only included the kite, and not
the hole within the kite. In many ways, the model exceeded our expectations. Examples are shown in Figure 5.

4.3.1 **Ground-truth Correction**

Some of the strongest evidence against memorization by the model is in what we term *dataset ground-truth correction*. In these instances the model produces a segmentation that is more complete than the ground-truth segmentation from the dataset during training. Examples were seen where the model was asked to locate instances of people, and the model successfully produced a segmentation of more people than that given by an incomplete ground-truth, as shown in Figure 6.
4.4 Common Mistakes & Shortcomings

While our overall results are positive, the model did make clear mistakes. We briefly list some failure modes here.

4.4.1 Overt Mistakes

As can be expected, the model made some mistakes. While some were understandable, others were clear mistakes where completely dissimilar objects were confused, unclear objects were referenced in the output segmentation, or obvious positive examples were missed. See Figure 7.

4.4.2 Partial Segmentations

At times the model gave what could be considered a correct response, though an incomplete one. In these cases the segmentation produced either only covered part of the actual object, was a bit fuzzy, or had poor boundaries. While this still successfully demonstrated the ability to recognize an object from the embedding, it lacked the desired completeness, as shown in Figure 8.
<table>
<thead>
<tr>
<th>Example</th>
<th>Query Image</th>
<th>Query Segmentation</th>
<th>Sample Image</th>
<th>Sample Segmentation</th>
<th>Network Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
<tr>
<td>Example 2</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
<tr>
<td>Example 3</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
<tr>
<td>Example 4</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
</tbody>
</table>

Figure 7: Examples of overt mistakes

<table>
<thead>
<tr>
<th>Example</th>
<th>Query Image</th>
<th>Query Segmentation</th>
<th>Sample Image</th>
<th>Sample Segmentation</th>
<th>Network Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
<tr>
<td>Example 2</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
<tr>
<td>Example 3</td>
<td><img src="image1" alt="Query Image" /></td>
<td><img src="image2" alt="Query Segmentation" /></td>
<td><img src="image3" alt="Sample Image" /></td>
<td><img src="image4" alt="Sample Segmentation" /></td>
<td><img src="image5" alt="Network Output" /></td>
</tr>
</tbody>
</table>

Figure 8: Examples of incomplete segmentations
4.4.3 Zebras & Elephants

While they are both four-legged animals which are native to Africa, zebras and elephants aren’t that similar. However, when training was extended from \textit{training set 1} and \textit{training set 2} (no elephants or zebras) to \textit{training set 3} (with \texttt{elephants} and \texttt{zebras}), we found that the model had great difficulty distinguishing between the two. Examples of these mistakes are found in Figure 9. However, as is noted in Figure ??, the model was at times able to successfully recognize each animal.

4.4.4 Possible Memorization

Additionally, even some of the successes were concerning. Certain correct segmentations gave the impression that on the small classes with few examples the model simply memorized the training examples. While the model may still have been able to learn generalizable information from these, their existence is always a concerning. This concern, however, was at least partially overcome as the model did show clear ability to generalize to new classes and
Figure 10: Examples of possible dataset memorization by the model images. However, it begs the question as to how much the model could be improved were this avoided. Shown in Figure 10.

4.4.5 Possible Explanations for These Shortcomings

As with most black box models, it is very difficult to pinpoint what causes difficulties. In this case, it perhaps could be that the embedding space is not robust enough, or that the What network is failing to capture relevant information of the image in that embedding space. Or, it could be that the “What” network struggles to interpret that embedding, or that the weights in that model are conditioned such that reliance on the embedding is difficult. Or perhaps the problem itself is too difficult, and may not be able to be learned with high accuracy. It was too difficult to answer any of these questions, but there were some possible explanations for imperfect performance that we did notice. We leave answering these questions for future work.

4.4.5.1 Dataset Limitations

While the dataset did contain a reasonably large number of classes, some of those classes were severely underrepresented. For example, the most common class, people, had 40,000 examples, while the least common, hairdryer, had only 150. Moreover, the bottom 50% of classes contained a meager 10% of training examples. This perhaps led the model to lack the robustness necessary for generalization.
There are also some instances of very poor segmentations in the dataset. While not very prevalent, these issues could affect learning and performance. Examples of these errors are shown in Figure 11.

Example 1

Example 2

Example 3

Figure 11: Examples showing dataset errors

5 Embedding Space Visualizations

By taking the embedding output of the “What” network, we can use PCA and T-SNE to visualize the high-dimensional embedding vector in 2 dimensions. In these visualizations, distinct class clusters imply that objects of different classes have unique representations in the embedding space produced and that the model was able to successfully learn to represent properties of images and objects in the embedding space.
5.1 Classes by Superclass

Figure ?? shows t-SNE results. When visualizing the different classes within each given super class, there is fairly good separation of classes. While there are instances of poor separation, it is interesting to note that they are usually for one of two reasons: the two classes share similar features or inadequate representation in the dataset. In the first image of the figure, there appears to be good separation, with significant overlap between the handbag, suitcase, and backpack classes. Interestingly, these objects all look fairly similar, so the overlapping representation is understandable. Similarly, while adequately separated, it is interesting to note that in the middle graph the classes cell phone and remote are very near one another and also have similar characteristics. Lastly, the final graph shows confusion between the classes scissors, hair dryer, and tooth brush. All three of these classes have extremely poor representation in the dataset, with fewer than 25 examples each.

5.2 Super Class Separation

An interesting and somewhat unexpected result is that there was fairly good separation of the super class categories. We see that food, animals, vehicles, and many other super classes containing the individual classes all showed fairly good clustering. While it would be unreasonable to expect the sort of clear separation we would hope to see in individual classes, it is interesting and reassuring to see that the model was able to learn image features well enough such that it was able to learn characteristics of images and classes about which it was given no information, as super class labels were never used anywhere in training.

6 Conclusion and Future Work

In this paper we have explored a new problem formulation for one-shot image recognition and localization, and developed a topology composed of standard network architectures which allowed us to solve this task. By combining this with several non-trivial methods of gradient
Figure 12: Embedding space visualizations of classes by super class
enhancement and dataset manipulation, we were able to learn a model that performs well on never-before-seen classes.

There is much work to be done, however. Additional experiments and ablation tests could be done to quantify the exact effect of many of our contributions. While we used the U-Net architecture as our segmentation network, we recognize that this not the state-of-the-art network for semantic segmentation; initial experiments with other architectures (such as DeepLab v3+) failed to converge in time for publication. Assessing these, and other issues, are critical steps for future work.

However, the general idea that neural networks can be jointly trained end-to-end as general purpose object descriptors and object recognizers is powerful, and merits further study; the ultimate limits and possibilities of this paradigm are both open questions.

7 Related Work

7.1 Foundations of Computer Vision

Computer vision has been a topic of serious study since at least the 1970’s. Since then there have been many advances which have allowed computers to be able to solve complex problems
pertaining to computer vision. Single models have been trained which can successfully identify images of thousands of distinct classes spanning a wide variety of genres (such as cars, planes, dogs, etc.[5, 8, 12, 24]). Others have been able to differentiate between classes within the same general area, in which some classes may be remarkably similar (such as 200 different types of birds[11]). As impressive as these results are, computer vision is nowhere near as robust and powerful as human vision, and the inflexibility dealing with novel image classes is one of those areas of weakness. But great progress has been made to bring us where we are today.

In the early days of computer vision, models focused heavily on direct analysis and manipulation of pixels in each image and often applied precise mathematical formulas and heuristics to achieve the desired result. Methods ranged from simple and intuitive—yet effective—formulas to sophisticated and complex mathematical methods—all based and focused on operations directly on image pixels. For example, Otsu’s method[20] segments an image into separate parts based on a threshold value which is selected to create maximum diversity between sections. This proved to be an effective and simple way to perform binary segmentation.

In the mid-1980’s Geoffrey Hinton popularized Rumelhart’s discovery of backpropogation[22] as a means to train ANNs, causing them to re-surge in popularity. Using this method LeCun used an ANN trained with backprop to recognize hand-written images[15]. By flattening images, or essentially cutting the image into strips of each row of pixels and linking all these strips together, one after another to form a 1-D vector containing the pixels of the image in a single row, images could be analyzed by traditional ANNs. However, flattening images in this way removes much of the spatial relativity between pixels. Instead, Yann LeCun was able to pass whole images as input into his network[14], which was comprised of 3 hidden layers, which were essentially fully connected layers working on unique, overlapping 3x3 subsets of the image. While previously ANNs often used hand-coded feature vectors, these layers could
be stacked, allowing the networks themselves to discover and operate upon features in the image on their own.

Progress continued along these lines until LeCun catapulted the field forward with the development of the Convolutional Neural Network[16] (CNN). CNNs work by passing over the image directly with so-called filters. In this context, a filter is typically some odd-sized matrix, which allows there to be one pixel in the center surrounded by all the pixels adjacent in some radius. For example, a 3x3 filter would have a single pixel in the center and also cover the 8 pixels directly adjacent in a 1-pixel radius. Passing several unique filters over the image allowed for different feature maps of the image to be created as the image is viewed through several different lenses so to speak.

This unique methodology allows the CNN to not only preserve the image structure and use important information on pixel relativity when analyzing and performing operations on the image, but also to perform several passes of analysis on features of increasing complexity and abstraction—forming feature hierarchies which can facilitate complex decision functions which can recognize objects in the image. In the years since its development, the CNN has come to completely dominate the field of computer vision and the CNN became the basic building block used to fuel many further developments, be they in image recognition, semantic segmentation, or image generation.

### 7.2 Image Recognition

In the years that followed, however, other machine learning methods showed increased promise and overshadowed CNNs. But with the publication of the popularly termed AlexNet[12] in 2012, CNNs once again came to dominate the field of Computer Vision. They found that ReLU activation functions were superior to the then-popular tanh functions and that overlapping pooling operations increased accuracy and decreased the likelihood of over-fitting the data. Additionally, they were able to show that the significant training time required for CNNs could be mitigated through training on multiple GPUs. Their most important
discovery, however, was that the depth of a network was essential to its function, as it allowed the network to build more complex feature hierarchies. Applying these principles, their Deep Convolutional Neural Network obliterated rival methods in the 2012 ImageNet competition; and Deep Convolutional Neural Networks became ubiquitous in Computer Vision tasks as advancements in CNNs allowed for greater and greater performance in Computer Vision tasks.

One such advancement was the development of the so-called VGG[24] network. While the component parts of this network were first applied by others, they put all these components together to create an network which was unique in its choice of layers and depth and essentially created the modern CNN architecture. Rather than using the large (11x11) filters of AlexNet, they shrunk the filter size to 3x3. This sharply reduced the complexity of the network, decreasing the number of parameters and allowing the network to increase in depth. Moreover, they found that convolutions of this size could be stacked and then in practice have the same size receptive field as the larger filters, but with the benefit of added depth. Additionally, the use of 1x1 convolutions to perform a pixel-wise linear transformation on the image allowed for additional computation and for the network to increase in depth, but without the higher parameter cost of convolutional layers. Both of these techniques—by increasing the network’s depth—increase the number of non-linear activations used between layers and therefore increases the capacity of the network. This allowed them to create a CNN of an unprecedented 19 layers and achieve remarkable performance.

But as the quest for networks of greater and greater depth continued, problems arose. As networks increased in depth and the gradient had to be backpropogated through more and more layers, the gradient began to diminish in a phenomenon known as the vanishing gradient problem. This meant that as you went deeper in the network towards the initial layers, the gradient was so small by the time that it reached them that they could not learn—causing decreases in training accuracy. In creating the ResNet[8], researchers at Microsoft were able
to overcome the vanishing gradient problem through the application of an existing principle known as skip connections. By taking the output of certain layers and forking the output into two paths—one that fed forward into the following layer as usual and another that would skip over the next two layers and then be concatenated to the output of the layer two layers forward—the gradient could be preserved. Further, by using identity skip connections which did not alter the data, network complexity was not increased. Using residual blocks containing these skip connections, networks were created that spanned 152 layers, with state-of-the-art accuracy.

While convolutions are more parameter-efficient than the fully connected layers of MLPs, they still contain a significant number of parameters and impose significant time in training. Standard convolutions require that each filter has the same depth as the number of channels in the input, and each channel of the input is operated on by a set of weights in the filter which are unique to that channel. Thus a convolution that with a 3x3 filter which operates on an input of 128 channels and produces an output of 256 channels would have 256 filters of size 3x3x128. The results of each filter pass over the image is then summed over the number of channels to produce a feature map of a single channel dimension which is the result of convolving that image with that filter. The final output of the convolutional layer is achieved by repeating this process for each of the filters and concatenating the resultant feature maps.

In another move towards greater efficiency, depthwise separable convolutions separated the two passes of the convolution. Instead of having a several filters in which each filter has a unique set of weights for each channel, depthwise separable convolutions contain only one such filter, and produce a feature map of equal dimensions in terms of channels to the input by skipping the summation. To allow for a change in channel dimensionality and provide additional computation, a 1x1 convolution is then performed. This greatly reduces the number of parameters while maintaining the analytical power of the convolution operation.
Though discovered earlier by Laurent Sifre[23], this type of convolution was popularized by Chollet in his paper proposing the Xception network[4].

7.3 Semantic Segmentation

As the field progressed from its early stages and images could be analyzed more effectively, semantic segmentation became a new problem. Instead of performing segmentation based solely on basic properties of the image (lines, borders, shading, etc.) and producing segmentations of essentially “thing 1”, “thing 2”, and “thing 3”, new models could make determinations about the nature of the image and could produce segmentations of specific classes, such as “dog”, “cat”, or “bus”. Moreover, segmentations of objects of the same class could be given the same label, e.g. cat, whereas previously images were segmented only as “thing 1” and “thing 2” even if they were both segmentations of the same object, e.g. cats. Now, models are able to segment and label pixels of a variety of classes and do so with incredible accuracy.

While image recognition tasks were finding great success due to their use of CNNs, the question of their application in semantic segmentation remained unanswered. In their development of the RCNN[7], Girshick et al. were able to use the advancements made in the field of image recognition and apply them to semantic segmentation by means of bounding boxes—a common way to label regions of an image as belonging to a specific class. Rather than label each pixel of an image as belonging to some class, bounding boxes form a perimeter around a space in the image which contains an example of that class, but not going so far as to label each pixel in the image individually. To create the semantically labeled bounding boxes, the RCNN selects and then processes parts of the image considered candidates for containing certain objects. However, as CNNs can only take images of a fixed, exact size, these candidate areas need to be reshaped to a consistent shape. But as many of these areas are likely not going to be perfect squares, they have to be reshaped in a way that does not preserve their original aspect ratio. Interestingly, this did not effect the ability of the CNN to take the candidate image slices and produce a feature vector upon which class-specific Support Vector
Machines give a score as to whether the candidate image slice contains an object of a known class. Their model was a great success and gave an example of how models successful in image recognition could be extended to the field of semantic segmentation.

Though CNNs showed great capacity in working with images, the typical architecture posed problems in creating pixel-wise segmentations. As CNNs typically process images, the size of the image is reduced and the number of channels increases—as a result of down-sampling layers and convolutions. Thus towards the end of the network the dimensions of the image are quite small and there are many feature-rich channels—a far cry from anything that could be seen as the original image or its semantic segmentation and how to recover something like the original image was unclear. Zeiler et al., however, were able to reverse these operations by performing essentially a complementary ones called transpose convolution[25] and unpooling[26]. These operations allow you to take an image and increase the size of the image and decrease the number of channels. In this way, an image could now increase in size and reduce the number of channels to produce and output image of equivalent size to the input image. This allows for an image to be passed in to the network and an image of equal size showing the segmentation be produced from the high-level, information-rich features produced by the convolutional layers.

Similarly inspired by the achievements of Deep Convolutional Neural Networks in image recognition, Long et. al showed that a CNN could be trained end-to-end on images directly to produce pixel-wise semantic segmentations without the use of other networks or techniques—in a network aptly named the Fully Convolutional Network[18] (FCN). It had long been known that features closer to the image contained information more closely related to the visual aspects of the image, while those features from higher levels contained more abstract information about the nature of the image. In the case of the FCN, they used information from these lower layers and added them to the output of higher layers. To ensure that these layers were of the same size and enable their summation, deconvolutional layers were applied
to the higher layers. Thus they were able to show that you could successfully train a model which exclusively used a convolutional architecture to produce a pixel-wise segmentation of an image.

Building on their success, Noh et al. incorporated unpooling layers and transpose convolutions successively as a means of being able to produce a network architecture that was essentially the mirror image of the typical CNN architecture and performed essentially the same operations in reverse[19]. By using unpooling to increase the size of the image and a learned deconvolution to densify and give meaning to the rather sparse output of the unpooling layers, they were able to take the small dimension, high channel images found at the top of CNNs and produce a semantic segmentation of the original image. Further, while Long et al. used deconvolutional layers that were initialized via bilinear interpolation—a means of computing the value of a pixel based on the four nearest pixels—and had the weights of the final deconvolutional layer being fixed, Noh et al. showed that these deconvolutional layers could be trained along with typical convolutional layers, with no need for special initialization, bilinear interpolation, or fixed weights.

The U-Net[21] then takes this basic architecture introduced by Noh et al. and re-introduces the idea of skip-connections which provide the spatially-rich information of lower layers to layers higher in the network. The U-Net takes its name from the picture created by taking a network of similar structure to the one described above and folding it in upon itself around where the down-sampling layers end and the up-sampling layers begin. It then creates a connection between the down-sampling and up-sampling layers at equivalent levels which produce and operate on images of the same dimensionality. By concatenating the low-level features of layers early in the network to the high level features of layers later in the network, the network is able to leverage the information given by each to produce meaningful semantic segmentations which are both faithful in their recreation of the image and competent in their ability to recognize various classes in the image.
The success of the U-Net and similar architectures with skip connections is the understanding that context is essential to understanding images. A single pixel, taken alone, yields almost no information about the image. But as the receptive field is expanded to include more and more pixels in some radius around it, the pixel begins to have more meaning. The problem is that with CNNs, using filters with large receptive fields can cause prohibitively high numbers of parameters and require either significant padding to the image or significant dimensionality reduction in the image in order to compensate. Atrous convolutions solve this by taking standard convolutional filters and parameterizing them with some rate \( r \) and placing \( r-1 \) spaces between the values of the filter and filling the rest with zeros. Thus a 3x3 filter with an atrous rate of 5 contains information about pixels that would be part of an 11x11 filter, but with over 13x fewer parameters.

Applying the Atrous convolution, Google developed Atrous Spatial Pyramid Pooling as part of their DeepLab[1] network. With this, an input is forked down several distinct paths, each with convolutions of distinct atrous rates. After the atrous convolution, each is passed through successive 1x1 convolutions. Finally, the outputs of the final 1x1 convolution from each atrous path is pooled to produce a feature map that contains information about the image from several different scales. Building on their work, in 2018 they applied depthwise separable convolutions to both the Atrous convolutions and the encoder network[2]. Additionally, they created a new simple decoder network which upsamples the encoder features, concatenates low-level features from the encoder, and upsamples again, before producing the final output. By combining these advancements, they were able to achieve state-of-the-art performance on semantic segmentation tasks.

### 7.4 Few-Shot Image Recognition

While most Image Recognition tasks work by outputting a vector of class probabilities, an alternate methodology suggests that perhaps the vector output by image recognition networks doesn’t necessarily need to be a descriptor of class probabilities. Instead that vector could be
used to describe the image generally. And while this new vector may not have the human interpretability of the class vector, this image descriptor could be used in other sways. As an instance of this, Siamese Neural Networks[9] essentially work by passing two images through a CNN to produce two vectors which are compared based on some distance metric, with the idea being that similar images will be closer together. Using this methodology, they were able to move outside the realm of describing images as being of some particular, finite set of classes, and instead describe whether or not two images likely belonged to the same class, whatever that class may be. Doing so allowed them to verify handwritten signatures by estimating the similarity between them.

Some years later Koch et al. showed that training these same types of networks allowed them to create models which could invent powerful discriminative features for describing images and that these could be used to generalize not only to new data, but entirely new distributions[10]. By training a network to determine the similarity of two characters (which would either be the same character drawn by a different subject or some other character, possibly from some other alphabet) they were able to take some image of a character and represent its features in some 4096 dimensional vector. In testing, using the 10 distinct alphabets not present during training, the network was able to match two unique instances of a single character, successfully identifying the match out of a sample of 20 possible characters which spanned characters within and without the same alphabet. By so doing they showed that an image could be powerfully described by a vector representing its features, and that these features could allow it to extend beyond its training set.

Not all approaches, however, take advantage of CNNs or similar networks. Using Bayesian probabilities, Lake et al. were able to achieve an even higher accuracy in that same task, surpassing even humans in their ability to match characters[13]. This Bayesian model learned to represent concepts as a composition of base parts. Thus different strokes and building blocks of the different characters were learned and used in the analysis of images of characters.
presented to it. Moreover, by so doing, their model was able to be used to both recognize and *generate* characters.

While finding and creating labeled datasets can be a challenge, there exists vast amounts of text data readily available. Frome et al. were able to leverage text data to aid in image recognition—even allowing them to generalize to classes of images which existed outside of their training set[6]. By first training a neural language model to create vector embeddings of words, they could then train a CNN to classify images by outputting the vector embedding of the name of the class. This was facilitated by removing the final layer of the CNN which outputs the vector of class probabilities and then apply a linear transformation from this penultimate layer’s 4096-dimensional representation of the image to the $h$-dimensional space used by the word embeddings. In this way, the image model learns the structure of the word embedding space—which already contains information placing similar objects near one another—and learns how features of the image relate to aspects of this space. Then, when the network is shown some image, it produces the name of some class by taking the vector output of the CNN and finding the word closest to that vector in the embedding space. This allowed for, in some cases (e.g. those in which the new class is *similar* to those used in training) the network be shown some new image from some new class—of which it has never seen a single example—and produce the correct label.
References


