Quantifying Feature Overlaps in Deep Neural Networks and Their Applications in Unsupervised Learning and Generative Adversarial Networks

Edward Collier

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.
QUANTIFYING FEATURE OVERLAPS IN DEEP NEURAL NETWORKS AND THEIR APPLICATIONS IN UNSUPERVISED LEARNING AND GENERATIVE ADVERSARIAL NETWORKS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

Division of Computer Science and Engineering

by
Edward D. Collier
B.S., Baylor University, 2014
M.S., Purdue University, 2016
August 2021
# Table of Contents

List of Tables ........................................ iv

List of Figures ........................................ v

Abstract ........................................ vii

Chapter 1. Introduction ................................. 1

Chapter 2. Neural Network Applicability ................ 3
  2.1. Background .................................. 3
  2.2. Related Work ............................... 5
  2.3. Applicability ............................... 6
  2.4. Applicability Predictor .................... 8
  2.5. Datasets ................................. 8
  2.6. Applicability Testing ..................... 9
  2.7. Applicability Predictor Evaluation ......... 12

Chapter 3. CactusNets ................................. 15
  3.1. Background ................................ 15
  3.2. Related Work ............................... 16
  3.3. Methodology ............................... 17
  3.4. Convolutional CactusNet .................... 20

Chapter 4. Octave Resonance .......................... 23
  4.1. Background ................................ 23
  4.2. Related Work ............................... 23
  4.3. Network Octave Variation ................... 24
  4.4. Octave Resonance Evaluation ............... 25
  4.5. Octave CactusNets ......................... 30

Chapter 5. Generative Applicability ................... 35
  5.1. Generative Adversarial Networks ............ 37
  5.2. Input Sets ............................... 38
  5.3. Discriminator Set Applicability ............. 38
  5.4. Discriminator Class Applicability .......... 39
  5.5. Generator Applicability .................... 40
  5.6. Transferring to the GAN ................... 40
  5.7. Generative Applicability Evaluation ....... 41
  5.8. Generative CactusNets ..................... 51

Chapter 6. Progressively Growing GANs ................ 55
  6.1. Introduction ................................ 55
  6.2. Related Work ............................... 57
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3. Data Preparation</td>
<td>59</td>
</tr>
<tr>
<td>6.4. Proposed Method</td>
<td>60</td>
</tr>
<tr>
<td>6.5. Experimental Evaluation</td>
<td>63</td>
</tr>
<tr>
<td>6.6. Applications in Earth Science</td>
<td>67</td>
</tr>
<tr>
<td>Chapter 7. Conclusions</td>
<td>69</td>
</tr>
<tr>
<td>Bibliography</td>
<td>71</td>
</tr>
<tr>
<td>Vita</td>
<td>78</td>
</tr>
</tbody>
</table>
**List of Tables**

2.1. Separability at conv5. ........................................... 10
2.2. Layer by layer class applicability. ........................................... 11
2.3. Image applicability network mean squared error. ................................. 13
2.4. Actual class applicability versus the predicted image applicability. ........ 14
3.1. CactusNet Classes .................................................... 20
3.2. Test Accuracy of CactusNet Branches ........................................... 22
4.1. Class by class octave resonance for the MNIST dataset. ......................... 27
4.2. Class by class octave resonance for the CIFAR10 dataset. ....................... 28
4.3. Test Accuracy of Constant Octave CactusNet Branches ............................... 32
4.4. Test Accuracy of Varying Octave CactusNet Branches ................................ 33
5.1. Layer by Layer set applicability scores for objective known classes from the MNIST dataset. ................................................................. 42
5.2. Layer by Layer set applicability scores for objective unknown classes from the MNIST dataset. ................................................................. 43
5.3. Layer by Layer set applicability scores for objective known classes from the CIFAR10 dataset. ................................................................. 44
5.4. Layer by Layer set applicability scores for objective unknown classes from the CIFAR10 dataset. ................................................................. 44
5.5. Layer by Layer set applicability scores for nonobjective classes from the CIFAR10 dataset. ................................................................. 45
5.6. FID scores for natural portion of CIFAR10. ........................................ 50
5.7. FID scores for Man-made portion of CIFAR10. ...................................... 50
5.8. Classification accuracy of GAN created images by method. ...................... 53
6.1. Summary of testing and training performance for U-Net, GAN and Progressive GAN. ................................................................. 65
List of Figures

2.1. Layer by layer applicability for the three subsets tested in experiment. 10

2.2. Layer by layer applicability for sample test classes. Top Left: Applicability for the Natural Known set. Top Right: Applicability for the Man-Made set. Bottom: Applicability for the Natural Unknown set. 11

2.3. Test results for the applicability predictor. 12

2.4. Work flow for applicability predictions in our CactusNet test. 13

3.1. The CactusNet architecture created in this paper. 16

3.2. CactusNet growth structure. 19

3.3. CactusNet results containing five branches. 21

4.1. Left: Prediction classes for MNIST for the down scale octaves with the lowest applicability. Right: Combined octave resonance for the Bangla Numeral dataset across all classes. 27

4.2. Left 10 class octave resonance from ILSVRC2012. Right Model octave resonance for the ILSVRC2012 dataset. 29

4.3. Left octave resonance for unknown classes that only the output layer was trained on. Right Combined octave resonance across all unknown classes. 29

4.4. Layer by layer octave resonance for the MNIST dataset. 31

4.5. Octave CactusNet trained on the MNIST dataset. 32

4.6. Octave CactusNet trained on the CIFAR10 dataset. The Cifar10 dataset has varying Octave across the training images. 33

5.1. Experimental set up for measuring applicability in the discriminator and generator. 35

5.2. Feature map activations for input class 5 mapped into 2D space for the discriminator, generator and classifier. 41

5.3. Class 5 activations mapped for classifier, discriminator and generator. 46

5.4. Test results for class applicability on the MNIST dataset. 47
5.5. Test results for class applicability on the CIFAR10 dataset.


5.7. Generator outputs for a GAN with various layers trained on the 0-4 classes and transferred to a GAN being trained on 5-9.

6.1. Example labeled images from the Las Vegas SpaceNet dataset.

6.2. Schematic flow diagram of the data preparation and proposed model that contains U-Net architecture as the generator.

6.3. Results of applying the three tested methods to sample images along with the input and the ground truth mask.

6.4. Progressive GAN results by resolution for xView images of different resolutions up to 256 x 256.

6.5. False negative results from applying progressively trained model.

6.6. Generator loss and accuracy over training Epochs for U-Net (a), GAN (b) and Progressive GAN (c).

6.7. Land use/land cover and temperature distribution patterns between 1990 and 2000. Reproduced from Chen et al., 2006 [16]
Abstract

Deep neural network learn a wide range of features from the input data. These features take many different forms from, structural to textural, and can be very scale invariant. The complexity of these features also differs from layer to layer. Much like the human brain, this behavior in deep neural networks can also be used to cluster and separate classes. Applicability in deep neural networks is the quantitative measurement of the networks ability to differentiate between clusters in feature space. Applicability can measure the differentiation between clusters of sets of classes, single classes, or even within the same class. In this work we present our metric and methodology for applicability, and compute the applicability for different sets, classes, inputs and octaves within a class. We also compute how applicability of features learned through adversarial training and show, to the first of our knowledge, how the features learned in a generator and discriminator overlap. Additionally, we use applicability to create a unsupervised tree like neural network that uses applicability to facilitate branching and maximized reuse of learned features. Lastly, we use Progressive training of Generative Adversarial Networks (GANs) to show how specializing and transferring features can lead to more accurate segmentation results.
Chapter 1. Introduction

The primary focus of this work is to present a first of its kind way to analyze overlapping clusters in multidimensional feature spaces learned by neural network. This method is called applicability, which is defined as the average differentiability between one cluster from all others in the feature space. This work covers 3 main overlapping topics associated with applicability. The first topic formally covers applicability. We give the quantitative definition of applicability. We split applicability into 3 groups based on scale, set, layer and class applicability. Next, we cover CactusNets. CactusNets are a new type of unsupervised neural network that uses applicability to calculate inflexion points where the learned features in a neural network no longer apply to the input. A new branch is created in the neural network at these inflexion points and trained on the new class.

We follow up our discussion on applicability and CactusNets by measuring the applicability of image octaves, which we call octave resonance. We use applicability to measure scale invariance in neural networks. Similar to the convolutional CactusNet we use applicability to train an octave CactusNet that will branch when the octave resonance falls below a specified branching threshold. Our remaining topic on applicability is Generative Adversarial Network (GAN) applicability. We measure the applicability learned during adversarial training; how is the generator applicability to the discriminator and visa versa.

The last topic covered is progressive growing and its application to segmentation. Progressive growing is a training method for neural networks that trains the network in segments, adding in a new layer after every training iteration. Training layers progressively specializes each layer to the task. This increases each layers applicability to the
training task, but reduced the applicability to other outside tasks.

We split the topics of this work into five chapters, Applicability, CactusNets, Octave Resonance, GAN Applicability and Progressive GAN. Each section is started with background information on the topic, followed by a summary of related work in the field that leads into the methodology and finally the results.
Chapter 2. Neural Network Applicability

2.1. Background

Neural networks have been improving by leaps and bounds for the last decade, most notably due to the emergence of CNN, unsupervised pretraining, and better regularization methods. On many difficult image recognition tasks, they are competitive with humans [35]. Nevertheless, there is lot of room for improvement. A human can easily learn to recognize a new type of animal from just one image of the animal, or even from a crude sketch. Even if the animal is oriented differently, it can most likely be learned in one shot. The same applies to non-image data.

Humans achieve this powerful zero shot or one shot understanding via learning by analogy. In other words, they start by trying to transfer detailed knowledge from another problem, then adjusting it to "fit" the new problem where necessary. We believe learning by analogy is the most advanced form of transfer learning, and is key to achieving truly impressive results on the machine learning algorithms of the future, both on images and raw data. A human can look at a problem and consider different solutions to past problems, and intuit whether they can be applied to part or all of the new problem. Note that the applicable solutions may not belong to similar problems. For example, insight from giving a political speech might provide insight into winning a chess match; or the fact that an image is of a chess match might make it easier to track where humans are in the image. This knowledge transfer in humans is enabled by the ability to discern what previous knowledge might apply to a new problem, even with little or no labeled data.

Deep convolutional neural networks trained over large datasets learn features that
are both generic to the whole dataset, and specific to individual classes in the dataset.

Learned features tend towards generic in the lower layers and specific in the higher layers of a network. Methods like fine-tuning are made possible because of the ability for one filter to apply to multiple target classes. Much like the human brain, this behavior can also be used to cluster and separate classes. However, to the best of our knowledge there is no metric for how applicable learned features are to specific classes.

We seek to measure how applicable a given network is to a given piece of data. For a convolutional network, it stands to reason that the pattern and intensity of the high level map responses should be noticeably different depending on whether many high level objects are recognized, and that how much of the image it successfully interprets is closely related to the applicability. The same principle applies to Deep Belief Networks (DBN); it should be possible by observing the neuron responses to determine whether the network (or part of the network) is recognizing familiar patterns. By measuring applicability we can tell what problems (networks) an unknown piece of data is applicable to, even without a label. Inversely, we can immediately tell when our current network is not adequately understanding a new piece of data, and can expand, retrain, or transfer knowledge into the network immediately in real time. This ability to measure network applicability in real time will be one of the key components in learning by analogy, particularly in non-convolution networks where knowledge is abstract mathematical relationships and any knowledge may potentially apply at any layer.
2.2. Related Work

For the past several years, advances in deep neural networks have shown to be a powerful tool for a variety of machine learning problems in multiple domains, including computer vision [61, 17, 27, 43], speech [31, 38], and text [2, 47]. For many of these domains, and especially for vision, each layer of the deep neural network learns features relevant to the target [5, 8]. For many objectives, a deep neural network requires a large-scale dataset to converge and obtain good accuracy [3]. For most tasks, however, large-scale datasets do not exist or are unobtainable. To circumvent this issue, existing deep neural networks can be fine-tuned for specific objectives. Fine-tuning repurposes the learned features of a pretrained deep neural networks which then can learn the unknown features needed for the new objective. Deep convolutional neural networks (CNN) trained on ImageNet [49] are commonly fine-tuned for different computer vision tasks. Fine-tuning significantly reduces the amount of training examples required to converge to a target objective [39].

Feature learning in deep neural networks exhibits a fascinating behavior in which the learned features tend to progress from generic, such as Gabor filters, to specific as the input moves down the network [72]. Such behavior is useful in understanding how a set of features in a deep neural network can be applied to multiple objectives. This is commonly referred to as transferability or transfer learning [62].

Over years, researchers have worked to improve the transferability in neural networks. Deep Adaptive Networks (DAN)[57] increase the transferability of task-specific features by matching different feature distributions in a reproducing kernel Hilbert space.
Similar to our proposed method, DANs assume that the target dataset has little to no labeled data. DANs use multiple kernel maximum mean discrepancies (MK-MMD) [32] to minimize the error between two datasets to facilitate greater transferability. Our method instead quantifies how well a neural network knows or can recognize an input to facilitate unsupervised learning.

2.3. Applicability

We define class applicability for a trained layer in a deep neural network on an input as how well the known features can be used to differentiate the input class from all other input classes. Within an objective (classifying natural images) we identify three subsets, nonobjective unknown, objective unknown, and objective known. In the case of natural images objective known are images of classes the network has been trained on, while objective unknown are natural images of classes the network has never been trained on. Finally, nonobjective unknown are images of man made classes the network has never been trained on.

Together these three groups cover a wide range of applicabilities from low to high, allowing us to train a predictor. For a single class in this set we compare it in a series of one vs. one classifications to a separate group of classes that approximate the set of all possible inputs in a domain (be they images, sound, text, etc.)

Given a deep neural network $N$, and a number of unknown objective and nonobjective classes, we approximate the set of all possible classes, giving an unknown set, $un$, of $k = 20$ classes. To find the class applicability at layer $n_i$ in a neural network $N = \{n_0, n_1, n_2, ..., n_z\}$ for a given class, $x$, we measure the ability of $N$ to differentiate $x$ from all the
classes in \( un \). We pair \( x \) with every class \( un_j \) in \( un=\{un_0, un_1, un_2, \ldots, un_k\} \). For each \((x, un_j)\) pair we fine-tune \( N \) with all its layers up to \( n_i \) frozen, and record the test accuracy \( \xi_j \). This accuracy represents how well \( x \) can be differentiated from \( un_j \) using learned features from all the layers at and before \( n_i \) which we will identify by the function shown \( N((x, un_j), n_i) \) in equation 2.1 where \( N((x, un_j), n_i) \) indicates the neural network \( N \) fine-tuned with the layers \( n_1, \ldots, n_i \) frozen.

\[
\xi_j = N((x, un_j), n_i) \quad (2.1)
\]

To obtain the class applicability of \( n_i \) on \( x \) the function \( N((x, un_k), n_i) \) must be applied to all the classes \( un_k \) in \( un \). Our class applicability metric is then the average differentiability between \( x \) and all other \( un_j \) individually. This is shown in equation 2.2.

\[
App_x = \frac{\sum_{j=1}^{z} \xi_j}{z} \quad (2.2)
\]

While it seems plausible that there is an individual image applicability for each image, we have not found a way to explicitly define it or measure it directly. Therefore, we set up our class applicability such that the average class applicability across all classes should approach the average separability between single pairs of classes. By extension, individual image applicability across a class should average to the class applicability. We then train a second neural network to estimate image applicability from the map responses, using class applicabilities as our labels.
2.4. Applicability Predictor

The key factor for the CactusNet architecture shown in Fig. 3.1 is its ability to branch at every layer for new classes. This branching is what allows for maximum reuse of already learned features. To branch at each layer some threshold $\theta$ must be defined for each layer. We define three thresholds each corresponding to the three subsets identified for an objective (objective known, objective unknown, and nonobjective unknown). The threshold for a subset at a given layer is the average applicability across some representative sample of that subset.

In addition to the threshold the CactusNet needs to have the ability to calculate the applicability of a given input in real-time and without sample classes from the three subsets. To calculate this applicability, predictor networks are created for each layer. For a given input within the objective of a pretrained deep neural network, the applicability network gives the predicted applicability of that layer’s features for the input.

We train the applicability predictors on large samples from our three subsets of the objective. The input for the network are the feature activations while the targets are the applicability of that specific class. The network uses a categorical cross class entropy loss function, and a relu activation function to generate an approximate applicability for an input.

2.5. Datasets

ILSVRC2012 consists of 1.2 million images from 1000 classes. The object classes can be split between man-made and natural objects. We use the same split as described in [72] that gives 449 natural classes and 551 man-made classes. The 449 natural classes were
used to train a convolutional neural network to classify between them.

We defined applicability as how well a layer’s features can be used to differentiate the input class from all other input classes. It would be difficult to get a representative sample of all possible input classes, so we approximate this with 20 classes our network has not been trained on, 10 natural and 10 man-made. We only use classes the network has not been trained on because any finite network would be specifically trained on a small fraction of the infinite set of all possible image classes; so a sample with all unknown classes should be more representative.

To calculate class applicability, we separate classes into the three subsets: unknown objective, known objective, and unknown nonobjective. In our experiments, the objective set is the natural set while the man-made represents the nonobjective set. We chose 10 classes for each set as described in 2.3 to train with a good mix of high, medium, and low applicability targets. The 10 classes used for the unknown objective were natural classes that the network had not been trained on. So in all 50 classes, 30v20, were used for applicability testing: 30 classes we wanted to measure the applicability of, and 20 to approximate all possible classes to compare them against to actually find that applicability.

2.6. Applicability Testing

The applicability of a class at a specific layer is defined as the average differentiability between that class and all other classes in the unknown set. This involved training 1v1 convolutional neural networks with all the layers at, and before, the testing layer frozen. The final validation accuracy was used as the differentiability metric between the two classes. In all 600 1v1 convolutional neural networks were trained. Table 2.1 gives an
Figure 2.1. Layer by layer applicability for the three subsets tested in experiment.

Table 2.1. Separability at conv5.

<table>
<thead>
<tr>
<th>Class</th>
<th>Toilet Paper</th>
<th>Gong</th>
<th>Buckle</th>
<th>Bucket</th>
<th>Pen</th>
<th>Lavender</th>
<th>Giraffe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moth(NU)</td>
<td>0.98</td>
<td>0.932</td>
<td>0.952</td>
<td>0.964</td>
<td>0.976</td>
<td>0.972</td>
<td>0.98</td>
</tr>
<tr>
<td>Nutria(NU)</td>
<td>0.972</td>
<td>0.992</td>
<td>1</td>
<td>0.98</td>
<td>1</td>
<td>1</td>
<td>0.988</td>
</tr>
<tr>
<td>Screwdriver(MM)</td>
<td>0.928</td>
<td>0.9</td>
<td>0.864</td>
<td>0.92</td>
<td>0.96</td>
<td>0.988</td>
<td>0.976</td>
</tr>
<tr>
<td>Coffee-machine(MM)</td>
<td>0.808</td>
<td>0.78</td>
<td>0.808</td>
<td>0.78</td>
<td>0.7</td>
<td>0.988</td>
<td>1</td>
</tr>
<tr>
<td>Tabby Cat(NK)</td>
<td>0.928</td>
<td>0.984</td>
<td>0.968</td>
<td>0.956</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bee(NK)</td>
<td>0.988</td>
<td>0.992</td>
<td>0.944</td>
<td>0.976</td>
<td>0.984</td>
<td>0.956</td>
<td>0.992</td>
</tr>
<tr>
<td>Natural Unknown</td>
<td>0.9708</td>
<td>0.9652</td>
<td>0.9824</td>
<td>0.9796</td>
<td>0.9888</td>
<td>0.9916</td>
<td>0.9876</td>
</tr>
<tr>
<td>Man-made</td>
<td>0.899</td>
<td>0.882</td>
<td>0.9172</td>
<td>0.9052</td>
<td>0.926</td>
<td>0.9932</td>
<td>0.986</td>
</tr>
<tr>
<td>Natural known</td>
<td>0.9632</td>
<td>0.974</td>
<td>0.9748</td>
<td>0.9792</td>
<td>0.9816</td>
<td>0.9872</td>
<td>0.9924</td>
</tr>
</tbody>
</table>

example of the layer 5 separability between six classes, one from each subset, and 10 of the test classes, while Table 2.2 gives the applicability for 3 classes at each layer.

In Fig. 2.1, we plot the average applicability for each of the three subsets as they move through the network. In the graph, we can see that the lower layers are more applicable to all the subsets, but the groups begin to separate farther along the graph. This result reinforces the results from [72] where learned features start generic but become more specific the farther along the network. The features are less applicable to unknown nonob-
Table 2.2. Layer by layer class applicability.

<table>
<thead>
<tr>
<th>Class</th>
<th>Conv1</th>
<th>Conv2</th>
<th>Conv3</th>
<th>Conv4</th>
<th>Conv5</th>
<th>FC6</th>
<th>FC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dolphin</td>
<td>0.987</td>
<td>0.982</td>
<td>0.979</td>
<td>0.978</td>
<td>0.9726</td>
<td>0.964</td>
<td>0.953</td>
</tr>
<tr>
<td>Coffee Mug</td>
<td>0.945</td>
<td>0.944</td>
<td>0.941</td>
<td>0.930</td>
<td>0.914</td>
<td>0.899</td>
<td>0.884</td>
</tr>
<tr>
<td>Llama</td>
<td>0.985</td>
<td>0.982</td>
<td>0.980</td>
<td>0.978</td>
<td>0.967</td>
<td>0.957</td>
<td>0.942</td>
</tr>
</tbody>
</table>

Objective classes at the higher layers which would indicate the need for branching. The features are fairly applicable to unknown objective classes even at the higher layers which makes sense given that even high level features are bound to have some overlap for all classes, known or unknown, in the same objective. There is an unexpected gap between the applicability for man-made objects and the known objects at conv1. This suggests that perhaps some edge detectors are more applicable to natural images and vice versa.

Fig. 2.1 is broken down into smaller subsets in Fig. 2.2. We can further see that the spread of the applicability between classes increases between the natural and man-made sets. The average spread between the highest point and lowest point across all layers for the man-made set is 0.082, while the spread for the natural known and unknown are 0.296 and 0.21 respectively. Due to the large nature of even the natural half of ILSVRC12 there are many features that are likely applicable to even man-made objects, but there is also a gap of unknown features that would be applicable to the man-made set.

Figure 2.2. Layer by layer applicability for sample test classes. Top Left: Applicability for the Natural Known set. Top Right: Applicability for the Man-Made set. Bottom: Applicability for the Natural Unknown set.
2.7. Applicability Predictor Evaluation

To predict the image applicability, we used small CNN’s as our applicability networks. The input for each predictor is the corresponding layer’s output which is treated as an \((h, w, maps)\) image where \(maps\) is the number of feature maps within that layer. Each convolution block in the predictor consists of two convolutional layers and a 2x2 max pooling layer. Each convolutional layer in the first block contains 32 filters, while those in the second block contain 64. For fully connected layers we made modifications to the predictors and treated the layer output as an image of shape \((1, 1, outputs)\). We train our networks by minimizing the loss function in eq. 2.3 where \(x^i\) is the target value and \(\hat{x}^i\) is the predicted value.

![Test results for the applicability predictor.](image)
\[
\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (x^i - \hat{x}^i)^2
\]  

(2.3)

The average training MSE was recorded to be 0.1485 across all the applicability predictor networks. To test the applicability predictors each predictor was tested on a class from each objective subset that the predictor was not trained on. For the test data, the recorded MSE was 0.4889. We report the training and testing mean squared (MSE) error for each layer in Table 2.3.

<table>
<thead>
<tr>
<th>Test</th>
<th>Layer</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1893</td>
<td>Conv1</td>
<td>0.3108</td>
</tr>
<tr>
<td>0.1538</td>
<td>Conv2</td>
<td>0.6137</td>
</tr>
<tr>
<td>0.1238</td>
<td>Conv3</td>
<td>0.6632</td>
</tr>
<tr>
<td>0.1657</td>
<td>Conv4</td>
<td>0.4252</td>
</tr>
<tr>
<td>0.1433</td>
<td>Conv5</td>
<td>0.4857</td>
</tr>
<tr>
<td>0.1253</td>
<td>FC6</td>
<td>0.3883</td>
</tr>
<tr>
<td>0.1382</td>
<td>FC7</td>
<td>0.5344</td>
</tr>
</tbody>
</table>

Table 2.3. Image applicability network mean squared error.

![Work flow for applicability predictions in our CactusNet test.](image)

Figure 2.4. Work flow for applicability predictions in our CactusNet test.

Fig. 2.3 shows the layer by layer predicted applicability versus the actual applica-
Table 2.4. Actual class applicability versus the predicted image applicability.

<table>
<thead>
<tr>
<th>Objective</th>
<th>Actual</th>
<th>Predicted</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nat Unknown</td>
<td>97.82</td>
<td>97.95</td>
<td>0.13</td>
</tr>
<tr>
<td>Nat Unknown</td>
<td>97.82</td>
<td>97.95</td>
<td>0.13</td>
</tr>
<tr>
<td>Nat Unknown</td>
<td>97.82</td>
<td>98.26</td>
<td>0.44</td>
</tr>
<tr>
<td>Nat Unknown</td>
<td>97.82</td>
<td>97.90</td>
<td>0.08</td>
</tr>
<tr>
<td>Nat Known</td>
<td>98.70</td>
<td>98.84</td>
<td>0.14</td>
</tr>
<tr>
<td>Nat Known</td>
<td>98.70</td>
<td>97.91</td>
<td>0.79</td>
</tr>
<tr>
<td>Nat Known</td>
<td>98.70</td>
<td>98.82</td>
<td>0.12</td>
</tr>
<tr>
<td>Nat Known</td>
<td>98.70</td>
<td>98.44</td>
<td>0.26</td>
</tr>
<tr>
<td>Man-Made</td>
<td>95.92</td>
<td>96.70</td>
<td>0.78</td>
</tr>
<tr>
<td>Man-Made</td>
<td>95.92</td>
<td>97.31</td>
<td>1.36</td>
</tr>
<tr>
<td>Man-Made</td>
<td>95.92</td>
<td>94.80</td>
<td>1.12</td>
</tr>
<tr>
<td>Man-Made</td>
<td>95.92</td>
<td>95.34</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Applicability metric we computed for that class. We can see in the graph that our applicability predictors are able to produce an image applicability that is very close to its true value. This suggests that our applicability networks are able to distinguish if the CNN has or has not been trained on an image even if they are both highly applicable. In Table 6.1, we give sample results from the conv4 applicability predictor. Interestingly the predictions on the man-made dataset tend to have the most variance, but only marginally so.
Chapter 3. CactusNets

3.1. Background

In our framework we seek to use our definition of applicability to facilitate unsupervised learning in deep neural networks. The architecture of the CactusNet is shown in Fig. 3.1. The branching structure for the CactusNet is shown in Fig. 3.2 and its algorithm is described in Algorithm 1. The CactusNet uses the predicted image applicability of an input to determine whether the given input is either objective known, objective unknown or nonobjective unknown, and branches accordingly. The base of the CactusNet is a pretrained deep neural network. This network can be trained on any objective, and need not even be well trained for that objective. The minimum requirement is that the network has learned some features that are applicable to its target objective. The CactusNet’s branching architecture is designed to efficiently handle large numbers of classes. The lower layers that contain generic features and are applicable to most every class are shared amongst all classes, minimizing the resources allocated to each new class. In the event an output layer does contain too many classes we can split by applicability and create two new branches. The applicability can be used to route inputs to the correct branch.

We use the AlexNet pretrained on the ILSVRC2012 dataset [49] to measure applicability and create the CactusNet. The principles used in AlexNet pretrained on ImageNet can be expanded and applied to other deep neural networks, datasets, and objective. For the purposes of creating a large enough dataset of known and unknown classes, we split ImageNet in half into man-made and natural sets like in [72]. The deep neural network we use is trained on the natural half, making the man-made half the unknown portion.
Several natural classes were also held out from the training set to represent a group of unknown natural classes for applicability measurement.

3.2. Related Work

Transfer learning has also been explored for unsupervised learning as well. In survey of how transferability can be applied to unsupervised learning [9], the author stated that while the results look promising, transfer learning applications would improve significantly if the underlying variation in high-level features could be disentangled and made more invariant. In this work, we use applicability to demonstrate where in a network the features of an input go from invariant to variant. This point of inflection is where the Cac-
tusNet creates a branch and circumvents invariance at the more varying and more specific layers.

The human mind identifies and clusters objects based on their features regardless of whether an object is known or not [34]. Adaptive resonance theory (ART) [13, 33] is a machine learning theory that attempts to determine whether an object belongs to a known object class by comparing the detected features of the object with the expected features of all known classes individually. If the smallest difference between the detected features of the object and some known class’s expected features is within a set threshold then the object is classified and is considered to belong to that class. This threshold is known as the vigilance parameter. If the difference exceeds the vigilance parameter, however, the object is considered to belong to a new class. This allows ART to perform unsupervised learning as it classifies not based on a target class, but differences in features. Over the years, several new variations of ART have been proposed including Fuzzy ART [14] which, uses fuzzy logic to improve ART’s stability.

### 3.3. Methodology

For an input $x$, we compute the feature representation for each layer $n_i$ down the network. Then we compute the predicted applicability for a layer using the feature representations and that layers applicability predictor network. If the applicability is below the threshold we have set to determine if the class is new or not, then we branch off the current network trunk at layer $n_i$.

After branching, the architecture of the network can remain the same as the original branch, or a different architecture can be used as long as it is compatible with the
Algorithm 1: CactusNet Algorithm

```
Input : Network Input \( x \);
1 List of applicability networks \( A \). Output: Class label \( l \).
2 Fine-tuned network \( N \). initialize \( l \leftarrow \emptyset \);
3 initialize thresholds \( t_1 \) and \( t_2 \);
4 for \( i=1 \) to \( k \) do
5     Get features at each candidate \( n_i \);
6     \( app \leftarrow n_i \) with max(applicability);
7     if \( app > t_1 \) then
8         classify normally;
9         return \( l \leftarrow \text{class} \)
10    end
11 else
12    if \( app \leq t_1 \) and \( app > t_2 \) then
13        \( l \leftarrow \text{objective unknown} \)
14    end
15 else
16    if \( app \leq t_2 \) then
17        \( l \leftarrow \text{nonobjective unknown} \)
18    end
19 end
```

shared layers in the original trunk of the CactusNet. Once a new branch is generated, then the CactusNet automatically assumes the input is of a new class and will commence learning. Given that we have not inherently modified the network architecture, the CactusNet is flexible in its operation, especially for learning. If the desire is for the CactusNet to learn from a few training examples, then any of the existing one shot learning methods [66, 68, 26] can be used to train a new branch. If a simpler method is desired, then all classes, whether known or unknown, can be input in tiny batches for traditional learning. We chose not to focus on a particular learning method since that is a well explored topic and out of the scope of this paper.

The path an input takes down the network is determined by its applicability at
each layer. When branching occurs, the applicability threshold acts as a guide diverting inputs down the correct path. There can be any number of branches at a given layer.

Figure 3.2. CactusNet growth structure.

Once a layer is trained, the applicability threshold of that layer is found from the applicability predictor’s training results. Once a layer $n_i$ is found to be applicable, the layer output is forwarded down all branches leading off the current branch, to all candidates for layer $n_{i+1}$. Each candidate processes the data and checks its applicability to that data. If the applicability is above at least one layer’s threshold, the layer with the highest applicability becomes layer $n_{i+1}$ and the input $x$ is routed there and to all branches leading off from there. If the applicability of $x$ for the layer at the end of every branch $b_i$ falls below that layer’s threshold, then a new branch for $x$ will be created. The growth structure of the CactusNet is shown in Fig. 3.2. In Fig. 3.2, we use the term lobe to denote a
3.4. Convolutional CactusNet

Using the the ImageNet 2012 [49] dataset we trained a convolutional CactusNet.

The base network is trained on 10 natural classes which make up the known set. In our experiment we use 20 unknown classes, 10 objective and 10 nonobjective, to create the CactusNet. The classes that are uses in the CactusNet are presented in Table 3.1.

<table>
<thead>
<tr>
<th>Known</th>
<th>Objective</th>
<th>Unknown</th>
<th>NonObjective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheetah</td>
<td>Giraffe</td>
<td>Car</td>
<td></td>
</tr>
<tr>
<td>Cat</td>
<td>Dolphin</td>
<td>Airplane</td>
<td></td>
</tr>
<tr>
<td>Dog</td>
<td>Bear</td>
<td>18-Wheeler</td>
<td></td>
</tr>
<tr>
<td>Butterfly</td>
<td>Flower</td>
<td>House</td>
<td></td>
</tr>
<tr>
<td>Whale</td>
<td>Cow</td>
<td>Basketball</td>
<td></td>
</tr>
<tr>
<td>Tree</td>
<td>Beetle</td>
<td>Tractor</td>
<td></td>
</tr>
<tr>
<td>Chicken</td>
<td>Cactus</td>
<td>Boat</td>
<td></td>
</tr>
<tr>
<td>Eagle</td>
<td>Sponge</td>
<td>Pickup</td>
<td></td>
</tr>
<tr>
<td>Elephant</td>
<td>Stingray</td>
<td>Skyscraper</td>
<td></td>
</tr>
<tr>
<td>Swordfish</td>
<td>Cardinal</td>
<td>Bicycle</td>
<td></td>
</tr>
</tbody>
</table>

We use a set up similar to that in Fig. 3.2 where the applicability is measured using trained applicability predictors for each input at each layer. The applicability predictors are trained after the training of the neural network to ensure that the predictors learn from the final feature maps from the neural network. The test data from the neural network training is uses to train the predictors, with the validation data used for testing and validation. The original network in the CactusNet has five convolutional layers followed by a couple of fully connected layers, the last being for classification. Every branch in the CactusNet will follow the same structure; every input will traverse seven layers total regardless of branching. For example, if a class branches at layer 3 then there will be
four layers in their new branch. The layers in a new branch are initialized with the weights from the equivalent layers from the branch the CactusNet just branched from.

Figure 3.3. CactusNet results containing five branches.

We present a visualization of the resulting CactusNet in Fig. 3.3. All the labels for the classes were added after to Fig. 3.3 after the CactusNet was trained. During training all unknown classes were unlabeled. There were five total branches from the main branch. It is no surprise to see that branches only occur after the second convolutional layer when the learned features begin to become more class specific. All the nonobjective (manmade) classes, with one exception, branched early after the second convolutional layer. The one nonobjective class, basketball, that did not branch made it all the way up the main branch. This is likely a result of the simplistic features that identify a basketball. The learned features in the known branch were highly applicable to basketball, and only required the class to be added to the final fully connected layer.
Given that the number of classes used in the CactusNet is limited, it is easy to see correlations between the classes and features in each branch. Classes added to the main branch were mammals four legged animals close to many of the known classes, basketball as discussed earlier and dolphin which is similar to the known class whale. All these classes have many, and likely most, of their features already known by the network. Branch A contains classes that all have wheels. While wheel is certainly not the only identifier for branch A it is likely a key feature in separating branch A from main branch C. However, wheels are likely not a key feature in identifying classes in branch A since it does not differentiate the classes.

<table>
<thead>
<tr>
<th>Branch</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.995</td>
</tr>
<tr>
<td>B</td>
<td>0.988</td>
</tr>
<tr>
<td>C</td>
<td>0.994</td>
</tr>
<tr>
<td>D</td>
<td>0.995</td>
</tr>
<tr>
<td>E</td>
<td>0.992</td>
</tr>
<tr>
<td>F</td>
<td>0.989</td>
</tr>
</tbody>
</table>

Table 3.2 presents that test accuracy for each branch of the CactusNet. Each branch of the CactusNet has high accuracy. What we can see is that the longer the branch is the test accuracy is hire. In these cases the branch has more features specifically trained for its own classes. Shorter branches reuse more features learned from classes not in that branch. While these reused features are applicable, it is interesting to see a drop off between the reused features and the longer more specialized branches.
Chapter 4. Octave Resonance

4.1. Background

Applicability, traditionally, has little concern for scale. To compute octave resonance, \( un \) must be split into subsets based on the octave where points in one cluster can be transformed into another via some common function \( x \rightarrow z(x) \); in this case \( z(x) \) is zooming in or out an octave. When checking for the octave resonance, it is not useful to check within the same set. Instead, the comparison is between one set and the rest. Keeping this in mind, octave resonance is then redefined as a trained neural network’s ability to differentiate an input from one octave from inputs across all other octaves. The octaves that resonate best with the learned features will have the highest amount of differentiation between classes. This changes Eq. 2.1 to Eq. 4.1.

\[
\xi_j = N((z_k(x), un_j), n_i) \tag{4.1}
\]

Where input \( x \) is transformed to the octave set \( k \) with transformation \( z \), and compared to all classes in \( un \). In this case, \( un \) still covers all the octave subsets.

4.2. Related Work

For deep neural networks, regardless of domain, learning is done by fitting a distribution of features at each layer of the network [5, 8, 20], where the larger the dataset, the better fit the feature space is [3, 52]. By transferring the learned weights from one domain to a similar one and performing moderate fine-tuning [39] on available data, a network can achieve convergence quickly and without a large dataset [54]. Transferring weights has found usage in diverse applications that involve image classification [73] and language pro-
One of the fascinating behaviors that transfer learning utilizes is the generic to specific nature in which features are learned. It is known that the lower layers of neural networks pick up generic features, that are common to almost all inputs for a specific input type, such as images. As the inputs make a forward pass up the network, the learned features become more and more specific to the domain and class of the input. Applicability is a quantitative measure of transferability, or domain adaptability, measuring how well the features from a learned task apply to any other task, including the learned task.

This work was originally published in [19] and has been approved for reuse.

4.3. Network Octave Variation

We hypothesize that in a trained model, containing only a single octave present within the training dataset, the applicability will drop-off more severely for images at varying octaves than if it were trained on a dataset with multiple octaves. Because neural networks have a finite number of trainable parameters, there is a cap on the amount that can be learned. For a neural network to have effectively learned an octave within its training set, it would have to fit to the distribution of the features for the octave that exist within the training dataset. Even over a large training set, a neural network will likely ignore the least common octaves, as those features would not be identified as important.

4.3.1. Layer Octave Resonance

While we have been measuring the octave resonance for the entire model, applicability can also be measured for individual layers. Layer applicability is identical to full
model applicability, except every layer from the first layer through the layer being measured are frozen. Model applicability is then the layer applicability for the final layer of the model.

4.4. Octave Resonance Evaluation

We present our experimental results in this section. In our experiments, we focus on datasets from two primary categories. The first are datasets which contain objects at only one scale, and the second are datasets that contain multiple scales. We show how the features resonate with various octaves across the two dataset types. Lastly, we measure the octave applicability layer by layer to show how the generic layers (i.e., layers that learn generic features) resonate better than specific layers (i.e., layers that learn specific features).

4.4.1. Datasets

We use four primary datasets in our evaluation; MNIST [51], Bangla Numeral [11, 10, 53], CIFAR10 [48], and ILSVRC2012 [49]. We separate the four datasets into two groups of two each. The first group consists of the MNIST and Bangla Numeral datasets which both have a consistent octave. The second group, CIFAR10 and ILSVRC2012, are datasets that cover multiple octaves.

There are an infinite number of octaves with which an octave can be compared against, we instead test over a finite, but representative range of octaves. Zero padding is used for all the images that have outward zooming. We have found during testing that the models have little meaningful variance for scale factors outside of range $z = (0.01, 6.0)$. 
4.4.2. Octave Resonance Evaluation

Octave resonance at any layer is a neural network’s average differentiability between one octave and all other octaves. To compute octave resonance, we finetuned a 1v1 neural network for all possible combinations of classes across all octaves, excluding identical pairs at different octaves, for each dataset. The layer(s) before and at which the applicability is being computed are frozen to preserve the learned features and octaves. The validation accuracy between the two classes, after convergence, is the differentiability metric between the two classes. The average differentiability metric for each octave is the applicability of that octave and the average differentiability for each class in each octave.

4.4.3. Constant Octave Model

For testing our constant octave models, we have trained models on two datasets that maintain a fairly consistent scale throughout, the MNIST dataset and the Bangla Numeral one [11, 10, 53]. For the MNIST and Bangla Numeral datasets, there were 170 1v1 neural networks trained. Figure 4.1 shows the class octave resonance across all the tested octave for the Bangla Numeral dataset. In Table 4.1, we present a numerical breakdown of octave resonance for each class, for select octaves, in the MNIST dataset.

Looking at both Fig. 4.1 and Table 4.1, we can see a clear applicability peak centered around the original octave. For all classes, the highest applicability occurs at the original octave. This is corroborated by the fact that the models were trained on a fairly constant octave, in addition to the fact that the set of features under consideration are relatively simple. The mode of the applicability for the classes varies in length slightly, some lasting much longer, but in general the applicability for scales 0.75, 1, and 1.25 are above
Table 4.1. Class by class octave resonance for the MNIST dataset.

<table>
<thead>
<tr>
<th>Class</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1.0</th>
<th>1.25</th>
<th>1.5</th>
<th>1.75</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.93</td>
<td>1.00</td>
<td>0.95</td>
<td>0.66</td>
<td>0.44</td>
<td>0.27</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.08</td>
<td>0.00</td>
<td>0.96</td>
<td>0.99</td>
<td>1.00</td>
<td>0.98</td>
<td>0.97</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.35</td>
<td>0.99</td>
<td>0.99</td>
<td>0.96</td>
<td>0.84</td>
<td>0.63</td>
<td>0.53</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.42</td>
<td>0.00</td>
<td>0.35</td>
<td>0.98</td>
<td>0.99</td>
<td>0.96</td>
<td>0.78</td>
<td>0.31</td>
<td>0.10</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.40</td>
<td>0.94</td>
<td>0.99</td>
<td>0.96</td>
<td>0.89</td>
<td>0.64</td>
<td>0.54</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.10</td>
<td>0.01</td>
<td>0.00</td>
<td>0.13</td>
<td>0.84</td>
<td>0.98</td>
<td>0.96</td>
<td>0.86</td>
<td>0.66</td>
<td>0.46</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.98</td>
<td>0.99</td>
<td>0.94</td>
<td>0.82</td>
<td>0.76</td>
<td>0.62</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.07</td>
<td>0.94</td>
<td>0.99</td>
<td>0.96</td>
<td>0.85</td>
<td>0.74</td>
<td>0.68</td>
</tr>
<tr>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.95</td>
<td>0.99</td>
<td>0.92</td>
<td>0.67</td>
<td>0.31</td>
<td>0.20</td>
</tr>
<tr>
<td>9</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
<td>0.85</td>
<td>0.94</td>
<td>0.98</td>
<td>0.94</td>
<td>0.72</td>
<td>0.40</td>
<td>0.16</td>
</tr>
</tbody>
</table>

0.9 before dropping off, sometimes drastically.

There are classes, like class 1 in MNIST, that have a less drastic applicability drop off when upscaling. This is likely a result of certain classes possessing scale invariant features. For example class 1 in MNIST is very simple and could possibly be defined by a vertical edge detection feature. This feature can exist regardless of scale and will lead to higher differentiability for larger scales than more complex scale variant features. This behavior also affects the accuracy of the low end features. When down scaling, the applicability has a much steeper drop-off before settling at close to zero.

Figure 4.1. *Left:* Prediction classes for MNIST for the down scale octaves with the lowest applicability. *Right:* Combined octave resonance for the Bangla Numeral dataset across all classes.
4.4.4. Varying Octave Model

Similar to case of the constant octave models, we trained 170 1v1 models for both the CIFAR10 and ILSVRC2012 datasets. In the case of ILSVRC2012, we hand selected 10 classes from the dataset to test on. For the more complex models, we chose to use the ResNet model [36] for both the ILSVRC2012 and CIFAR10 datasets. On the Right side of Fig. 4.2, we present the octave resonance for the test classes in ILSVRC2012, on the left side, we present 10 class octave resonance from ILSVRC2012, while in Table 4.2, we provide numeric values for the octave resonance for the CIFAR10 classes.

Table 4.2. Class by class octave resonance for the CIFAR10 dataset.

<table>
<thead>
<tr>
<th>Class</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1.0</th>
<th>1.25</th>
<th>1.5</th>
<th>1.75</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>bird</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.35</td>
<td>0.99</td>
<td>0.99</td>
<td>0.96</td>
<td>0.95</td>
<td>0.95</td>
<td>0.90</td>
</tr>
<tr>
<td>cat</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.10</td>
<td>0.35</td>
<td>0.98</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>deer</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.14</td>
<td>0.95</td>
<td>0.98</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>dog</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.30</td>
<td>0.85</td>
<td>0.92</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>0.94</td>
<td>0.88</td>
</tr>
<tr>
<td>frog</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.15</td>
<td>0.50</td>
<td>0.98</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.96</td>
<td>0.94</td>
</tr>
<tr>
<td>horse</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.45</td>
<td>0.97</td>
<td>0.98</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>ship</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.12</td>
<td>0.19</td>
<td>0.88</td>
<td>0.94</td>
<td>0.99</td>
<td>0.99</td>
<td>0.85</td>
</tr>
<tr>
<td>truck</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
<td>0.00</td>
<td>0.85</td>
<td>0.94</td>
<td>0.98</td>
<td>0.94</td>
<td>0.92</td>
<td>0.80</td>
<td>0.66</td>
</tr>
</tbody>
</table>

It is notable that compared to the single octave datasets, the applicability peak lasts substantially longer and the drop-off for upscaling is much less drastic and more gradual. The opposite is true in the down scale direction. While the mode does persist longer for the down scaling, the drop-off is extreme, almost as if there is a point where no feature can fit that octave. Another cause for the rapid applicability drop-off is the 0 padding. Unlike the constant octave models, the training datasets images are RGB and not binary. The zero padding adds an unnatural artifact to the image that could be the cause of the applicability drop.
One of the more interesting behaviors, that we can see on the \textit{Left} side of Fig. 4.1, is that the highest applicability is not centered around the original octave. This behavior can be best explained by looking at the applicability of an individual image. Because the scale of the inputs varies between all the images, the model will learn the features at the most common or important octaves. This will partially exclude some octaves from the feature set reducing the applicability for individual images on the periphery of the feature space.

\textbf{4.4.5. Octave Resonance of Unknown Classes}

Figure 4.3. \textit{Left} octave resonance for unknown classes that only the output layer was trained on. \textit{Right} Combined octave resonance across all unknown classes.
It is also valuable to observe how octaves react to classes that the features are not necessarily learned for. In Fig. 4.3, we present the applicability for classes that the model was not originally trained on. We split the MNIST and Bangla numeral datasets in half, and trained a neural network on the first half, which we will refer to the known set. We can see that comparing the unknown graphs (Fig. 4.3) to those in Fig. 4.1, the octave resonance is still centered around one, the base applicability, as expected. The peak in the unknown set is much smaller with the applicability drop-off much more severe than for the known classes. Because the lower layers were frozen, the unknown classes are fitting themselves to features that might not necessarily apply perfectly to them.

4.4.6. Layer Octave Resonance

It has been shown that the applicability of earlier layers in a neural network is high for almost all classes in a domain, known or unknown [18]. Generic features are represented at the earlier layers of a neural network and have more applicability across the domain. In Fig. 4.4, we measure the octave resonance layer by layer. We can see that in the earlier layers, the applicability for each octave is close together, supporting the hypothesis that the generic features can resonate across octaves. The octave resonance diminishes as the layers become more specific, where, by the last layer the octaves on the extremes have substantially lower applicability than the original octave.

4.5. Octave CactusNets

We create the Octave CactusNet is much the same fashion as in section 3.4. The key difference for the Octave CactusNet is that the main branch is trained on all classes. The unknown classes are instead the known classes that are up scaled and down scaled.
Figure 4.4. Layer by layer octave resonance for the MNIST dataset.

We use the same scaling that is used in Fig. 4.1 and Fig. 4.2, where 1.0 is the known octave. We train a constant octave, on MNIST, and variable octave CactusNet, on CIFAR10, to observe the behavior each have on the resulting CactusNet.

Octave CactusNets also use applicability predictors to measure the applicability of individual inputs at each layer. The data is split between train and test using a 70/30 split. The test data is split in half, 15% of total, and used for applicability predictor training. The images in the applicability predictor dataset are scaled to each octave the Cactus will be tested on. The labels are the resonance for that class at that octave.

We present the resulting Octave CatusNets in Fig. 4.5 for the constant octave model, and Fig. 4.6 for the varying octave model. At first glance one of the differences between the two that jumps out is the number of branches between the constant and varying octave models. The constant octave CactusNet contains six branches where the varying octave branch does not. This is a result of the varying octave model have learned features at many octaves already in the main branch. Resulting branches occur for octaves
that do not belong to the training data or are sparsely present in the training data. This behaviour matches what we have seen in Fig. 4.1 and Fig. 4.2.

Table 4.3. Test Accuracy of Constant Octave CactusNet Branches

<table>
<thead>
<tr>
<th>Branch</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.331</td>
</tr>
<tr>
<td>B</td>
<td>0.487</td>
</tr>
<tr>
<td>C</td>
<td>0.995</td>
</tr>
<tr>
<td>D</td>
<td>0.998</td>
</tr>
<tr>
<td>E</td>
<td>0.997</td>
</tr>
<tr>
<td>F</td>
<td>0.997</td>
</tr>
<tr>
<td>G</td>
<td>0.996</td>
</tr>
</tbody>
</table>

Table 4.3 provides the test accuracy of the constant octave CactusNet branch wise. The first thing that jumps out is the low accuracy scores for branches A and B. All the classes down scaled at 0.2 and below are contained within branches A and B. Branch B also contains classes at the highest octaves as well. As we have see, images down scaled to this point may only contain a few non zero pixels which is not enough to have distinguishing features. High octaves can result in a similar effect but instead of the image containing all zeros it might contain all ones. Branch A contains images at the very lowest and
Figure 4.6. Octave CactusNet trained on the CIFAR10 dataset. The Cifar10 dataset has varying Octave across the training images.

least differentiable octaves, [0.01, 0.15], and is doing little more than randomly guessing. Branch B contains some classes and octaves, like class 3 at octave 0.2, that can be differentiated but outside of those the rest of the classes and octaves cannot be differentiated. Branches with the intermediate to high octaves have accuracy greater than 0.99.

Table 4.4. Test Accuracy of Varying Octave CactusNet Branches

<table>
<thead>
<tr>
<th>Branch</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.551</td>
</tr>
<tr>
<td>B</td>
<td>0.993</td>
</tr>
<tr>
<td>C</td>
<td>0.991</td>
</tr>
<tr>
<td>D</td>
<td>0.994</td>
</tr>
</tbody>
</table>

In Table 4.4 we present the test accuracy for the varying octave CactusNet. This is the least branching of the two Octave CactusNets. Most of the octaves still matched to classes in the main branch. Interestingly, the main branch did not have the highest accuracy, and instead branch D did followed by branch C. This could be because the num-
ber of classes and octaves learned with in the main branch is significantly greater than branches D and C which are specialized on only a few classes and octaves. These specialized branches have a simpler task with fewer features and produce a higher test accuracy. Similar to the constant octave CactusNet there is a branch where almost all the classes at the lowest and highest octaves fall in too. Images at these octaves contain almost no features, resulting in a branch that does little more than random guessing.
Chapter 5. Generative Applicability

Since their inception, generative adversarial networks (GANs) have enjoyed tremendous success in a large number of domains including image synthesis, domain adaptation, and data generation [29, 40, 69, 21]. GANs incorporate two networks, a discriminator and a generator [29], that engage in a minmax game with one another. A generator $X$ tries to produce samples from a target distribution $x$, with the discriminator $X$ aiming to discriminate if its input is real, i.e., from $x$, or fake, generated by $X$. This minmax game played between the two competing networks minimizes an adversarial loss term, where the end goal is attaining the Nash equilibrium [29, 37].

Figure 5.1. Experimental set up for measuring applicability in the discriminator and generator.

GANs need copious volumes of training data for learning the mapping onto $x$ accurately. Acquiring a sizable amount of data, enough for convergence of the training procedure, is a bottleneck in generative adversarial learning today. A promising approach for
mitigating this bottleneck is transfer learning or domain adaptation. By using a source domain learned features (or at least some of them), a target domain has a built up prior knowledge base [72]. So long as there is some overlap between the two feature domains, this prior is a good initializer for neural networks [24].

It should be noted though that the phenomenon known as negative transfer [62] can have the opposite effect on transfer learning. Taking inspiration from self-taught learning in humans, learning which features to transfer helps uniformly stabilize the learning process after transfer [55], combating the negative transfer problem. One of the keys to understanding transferring features is making sense of the general to specific nature by which neural networks learn features [72]. The learned feature space in the early layer(s) is either the same or can have large overlaps and can be transferred to almost any network, however, the ability to transfer breaks down as the features get more specific and the distance between the feature spaces grows. Consequently, quantifying how well learned features can apply to both individual classes and inputs reveals that there are varying degrees of overlap between classes in the same domain as well as across domains [18].

Recent work on transfer learning has helped improve the understanding of how neural networks learn features and the type of features they learn, by leaps and bounds [72, 57, 58]. Despite this, there has been little to no research on how features are learned during an adversarial process and during synthesis or what features can be transferred in an adversarial training regimen. GANs provide a unique opportunity to look into the features of classes with competing parts but with a unified goal.

This work was originally published in [23] and has been approved for reuse. In the following section, we outline our proposed methodology for measuring and observing the
applicability of GANs. We detail the tests and the theoretical background for set and layer applicability in discriminators. We follow that by transferring weights in generators to measure applicability of generators. Lastly we outline how applicability to a whole GAN can be quantified and measured.

5.1. Generative Adversarial Networks

There are many flavors of generative adversarial networks, but the goal in general remains the same. A Generator tries to ”cheat” a discriminator whose job is to distinguish real samples drawn from a probability distribution from fake ones generated by . For the most basic GAN, we express this objective function as [29]:

$$\min \max \mathcal{L}(\cdot) = \mathbb{E}_x[\log(x)] + \mathbb{E}_\tau[\log(1 - ((\tau)))],$$  \hspace{1cm} (5.1)

with $x$ being the ground truth drawn iid from the probability distribution to be learned and $\tau$ being some latent noise encoding.

It has been shown in recent work that under ideal hyperparameter tuning and with random initializers most GAN flavors can produce identical behavior [59]. It is for this reason that we choose to focus our work on some of most common styles of GAN [29, 64]. We assume that our findings will translate to various other GAN architectures.

Evaluating and scoring the performance of GANs has been one of the biggest problems facing researchers in this area. Fréchet Inception Distance (FID) has been shown to be a good metric for evaluating GAN performance [25, 37, 59, 71]. FID is computed by representing the layer embeddings of true and fake data as Gaussians and computing the distance between them. This is mathematically represented as [25, 37]:

$$\text{FID}(x, \hat{x}) = ||\mu_x - \mu||^2 + Tr\left(\sigma_x + \sigma - 2(\sigma_x\sigma)^{1/2}\right)$$  \hspace{1cm} (5.2)
where \( x \) and \( \ast \) are the ground truth and generator distributions respectively, \((\sigma, \mu)\) are the means and covariance of the respective Gaussians, and \( Tr \) represents the trace. In our work, we compare the FID scores of a GANs with varying amounts of transfer at different layers to quantify the effect class and set specific features have on adversarial training.

5.2. Input Sets

To test applicability, we use the three types of input sets defined for applicability testing: \textit{objective known}, \textit{objective unknown}, and \textit{nonobjective}. Each type maintains their original definition for generative applicability. Each set applies to the entire GAN, both generator and discriminator, not just part of the system.

5.3. Discriminator Set Applicability

Set applicability is defined as a trained network’s ability to apply to problems from a different set or domain [72]. This is measured by the difference of the source domain performance of the model from the target domain performance, \( \epsilon = |\psi(d_t) - \psi(d_s)| \) [72], where \( d_s \) and \( d_t \) respectively denote the source and the target domains, and \( \psi() \) denotes the objective function of the neural network.

Another way to think of applicability is a feature’s ability to differentiate between \( d_t \) and \( d_s \). For set applicability for discriminators, we follow a similar set up to that proposed in [72]. The source domain and the classes created by the generative model will be defined as \( A \) and the target domain will be defined as \( B \). We will transfer, layer by layer, the weights from the discriminator in \( A \) onto \( B \) testing the accuracy with each transferred layer. We compare the ability of the discriminator of \( B \) to classify, with a baseline neural
network trained on the transferred classes. We perform this experiment for each type of input set.

5.4. Discriminator Class Applicability

Apart from observing how well the features from a learned discriminator apply to different domains, we also look at how the layers within a discriminator learn features and how applicable those features can be to individual classes. It is important to observe what types of features the discriminator learns class by class during the adversarial process and how they differ from a traditional classification model.

We define class applicability similar to [18]. In this set up, we seek to measure how well a neural network, layer by layer, can differentiate an input class \( c \) from each individual class in the union of the objective unknown and the nonobjective sets (which represents the set of all unknown classes). Let \( N = \{n_0, \ldots, n_t\} \) be a neural network with layers \( n_i, i = 1, \ldots, t \). Let \( un \) be the set of unknown classes. For each input class/unknown class pair, which we will denote by \((x, un_j)\), where \( un_j \) denotes the \( j \)th unknown class, we freeze all the layers up to and including the desired layer we are measuring, which we label as \( n_i \), for the neural network \( N \). \( N \) is fine tuned as a binary classifier on \((x, un_j)\). We denote this function as \( \xi_i = N((x, un_j), n_i) \). The class applicability is then the averaged sum of \( \xi \) over all the layers which represents the average differentiability.

\[
App_x(un_j) = \frac{\sum_{v=1}^{t} \xi_v}{t} \quad (5.3)
\]

Similar to set applicability, this experiment is performed on the discriminator and compared to a traditionally trained classifier measured under identical circumstances.
5.5. Generator Applicability

Apart from measuring the applicability of features in discriminators, we also observe the features learned in generators and their applicability to other tasks, such as classification. To do this, we transfer the learned weights from a trained generator to a classifier. For decoder generators, we invert the generator [60] to convert the decoder to an architecture that makes sense for classification. This allows tests that were performed on the discriminators for applicability to be used for measuring applicability in a generator. We test both set and class applicability of the trained generator.

Figure 6.2 gives a general schema for testing applicability for both discriminators and generators. As shown in Figure 6.2 the GAN will be trained on half the classes of a given dataset, either MNIST or CIFAR10 in our case. After training, the layers will be transferred to an untrained classification network and frozen during training to ensure the network uses only the previously learned weights at those layers. Multiple classification networks are built using the GAN layers each measuring the applicability at that layer. For MNIST our GANs follow the architecture proposed in [29] and for CIFAR10 we follow the architecture from [64].

5.6. Transferring to the GAN

One of the more practical metrics for evaluating applicability of GAN features is the effect transfer has on the whole adversarial process. Evaluating metrics like discriminator loss and convergence time, when transferring learned features to a GAN, can provide a macro view of the effect transferring has on the GAN. More importantly, we compute the FID of the final generator after convergence to evaluate performance from a more mi-
cro perspective. Similar to previous set ups, we transfer the learned features to the generator and discriminator of a GAN and use that as prior knowledge for training. The transferred layers are frozen to ensure that the model is entirely reliant on those features. The objective of the GAN will be to generate samples from the training distribution using, either entirely or partially, the features transferred to it. As with the other experiments performed, we use the three objective sets, known objective, unknown objective, and nonobjective, to understand the applicability of the representations acquired by the GAN.

5.7. Generative Applicability Evaluation

Figure 5.2. Feature map activations for input class 5 mapped into 2D space for the discriminator, generator and classifier.

In this section, we provide experiments and results for generative applicability. We performed experiments on two separate datasets; CIFAR10 and MNIST. We use a natural/man-made class split in CIFAR10 to generate the objective and nonobjective sets, while for MNIST, we only have known objective and unknown objective sets. Additionally, we focus our work on the standard GAN architecture for MNIST and the DCGAN architecture for CIFAR10. Recent work has shown that under consistent circumstances the performance of many GAN architectures is similar [59], so we assume that the results presented will apply to other architectures as well.
5.7.1. Set Applicability

Using MNIST, we test the set applicability for a GAN whose generator and discriminator each contain four fully connected layers with size $[128, 256, 512, 1024]$, each with relu activation and batch normalization. For MNIST, we test both objective known and objective unknown applicability for each layer of the discriminator and generator. Each are compared with a standard classification network as the control. For objective unknown, the data is split between classes 0-4 and 5-9. We run multiple tests: one using 0-4 as the base with 5-9 being transferred and the other with 5-9 as the base and 0-4 being transferred.

To test the applicability for pretrained weights on a set at a layer $n$, all the layers before and including $n$ are frozen during training. The unfrozen layers contain randomly initialized weights and are trained using backpropagation. The output layer indicates that all the layers except the final classification layer were frozen. The results for set applicability on the objective known set of MNIST are presented in Table 5.1, while the objective unknown tests on both splits are presented in Table 5.2. The values shown in both tables are the validation accuracy for each model after convergence.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Discriminator</th>
<th>Generator</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>0.980</td>
<td>0.953</td>
<td>0.989</td>
</tr>
<tr>
<td>Layer 2</td>
<td>0.971</td>
<td>0.948</td>
<td>0.981</td>
</tr>
<tr>
<td>Layer 3</td>
<td>0.964</td>
<td>0.941</td>
<td>0.974</td>
</tr>
<tr>
<td>Output</td>
<td>0.955</td>
<td>0.926</td>
<td>0.969</td>
</tr>
</tbody>
</table>

In Table 5.1, we present the set applicability scores for the objective known set. As we would expect, the control performs best, but the scores between the discriminator
Table 5.2. Layer by Layer set applicability scores for objective unknown classes from the MNIST dataset.

<table>
<thead>
<tr>
<th></th>
<th>Discriminator 0-4</th>
<th>Generator 0-4</th>
<th>Control 0-4</th>
<th>Discriminator 5-9</th>
<th>Generator 5-9</th>
<th>Control 5-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>0.971</td>
<td>0.941</td>
<td>0.982</td>
<td>0.977</td>
<td>0.943</td>
<td>0.979</td>
</tr>
<tr>
<td>Layer 2</td>
<td>0.966</td>
<td>0.932</td>
<td>0.975</td>
<td>0.973</td>
<td>0.935</td>
<td>0.978</td>
</tr>
<tr>
<td>Layer 3</td>
<td>0.965</td>
<td>0.929</td>
<td>0.969</td>
<td>0.969</td>
<td>0.930</td>
<td>0.971</td>
</tr>
<tr>
<td>Output</td>
<td>0.963</td>
<td>0.921</td>
<td>0.964</td>
<td>0.975</td>
<td>0.926</td>
<td>0.962</td>
</tr>
</tbody>
</table>

and generator are of particular interest. The discriminator shows considerably greater applicability to the known set, classes the GAN was trained on, than the generator with an average applicability of 0.968, compared with the generator whose average applicability is 0.942. This would indicate that the features learned for differentiating outputs, as either real or fake, can be better applied to classification that those learned in for synthesis. The features learned for synthesis can be well applied to classification as the generator still does perform well on classification.

In Table 5.2, it is seen that the trend of the results shown in Table 5.1 continues to hold true for the unknown objective set; with the control performing best, followed by the discriminator, and then the generator. As we’d expect, classification using features from other classes performs worse, but the accuracy scores still indicate that there is a large overlap. Interestingly, overall the features learned from the classes in the 5-9 range applied better to the 0-4 classes than vice versa.

To observe feature applicability for models with more complex features we repeat our experiments for models trained on the CIFAR10 dataset. Additionally, because there are both man made and natural classes, we also measure set applicability for the third set: nonobjective. The set up is almost identical to the previous MNIST test. However, while
the objective/nonobjective split is between man made and natural, the objective known
and unknown split is less obvious. In this case, we chose to train the control and GAN on
three random classes from the objective set and use the remaining for testing. The net-
work architecture in this case is four convolutional layers with size $[32, 64, 128, 256]$ ending
with a fully connected classification layer. Results from the objective known and unknown
tests are presented in Table 5.3 and Table 5.4, while results from the nonobjective tests
are shown in Table 5.5.

Table 5.3. Layer by Layer set applicability scores for objective known classes from the
CIFAR10 dataset.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Discriminator</th>
<th>Generator</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>0.782</td>
<td>0.778</td>
<td>0.790</td>
</tr>
<tr>
<td>Layer 2</td>
<td>0.776</td>
<td>0.774</td>
<td>0.788</td>
</tr>
<tr>
<td>Layer 3</td>
<td>0.768</td>
<td>0.765</td>
<td>0.785</td>
</tr>
<tr>
<td>Layer 4</td>
<td>0.760</td>
<td>0.758</td>
<td>0.780</td>
</tr>
<tr>
<td>Layer 5</td>
<td>0.754</td>
<td>0.750</td>
<td>0.776</td>
</tr>
<tr>
<td>Output</td>
<td>0.750</td>
<td>0.745</td>
<td>0.771</td>
</tr>
</tbody>
</table>

Table 5.4. Layer by Layer set applicability scores for objective unknown classes from the
CIFAR10 dataset.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Discriminator Man-made</th>
<th>Generator Man-made</th>
<th>Control Man-made</th>
<th>Discriminator Natural</th>
<th>Generator Natural</th>
<th>Control Natural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>0.777</td>
<td>0.775</td>
<td>0.788</td>
<td>0.774</td>
<td>0.772</td>
<td>0.785</td>
</tr>
<tr>
<td>Layer 2</td>
<td>0.774</td>
<td>0.769</td>
<td>0.784</td>
<td>0.770</td>
<td>0.768</td>
<td>0.782</td>
</tr>
<tr>
<td>Layer 3</td>
<td>0.766</td>
<td>0.763</td>
<td>0.780</td>
<td>0.763</td>
<td>0.760</td>
<td>0.777</td>
</tr>
<tr>
<td>Layer 4</td>
<td>0.757</td>
<td>0.755</td>
<td>0.774</td>
<td>0.757</td>
<td>0.753</td>
<td>0.772</td>
</tr>
<tr>
<td>Layer 5</td>
<td>0.750</td>
<td>0.748</td>
<td>0.770</td>
<td>0.751</td>
<td>0.744</td>
<td>0.768</td>
</tr>
<tr>
<td>Output</td>
<td>0.748</td>
<td>0.740</td>
<td>0.764</td>
<td>0.746</td>
<td>0.738</td>
<td>0.761</td>
</tr>
</tbody>
</table>

We can see that the trends follow a similar pattern to those for MNIST except that
the applicability drop-off from layer to layer in both Table 5.3 and Table 5.5 is greater and
more drastic. This is most likely due to the models learning more specialized features due
to the increased complexity of the data. What using the CIFAR10 data allows for is to observe the applicability of the features from two sets of classes from different objectives; man-made and natural in this case.

Table 5.5. Layer by Layer set applicability scores for nonobjective classes from the CIFAR10 dataset.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Discriminator Man-made</th>
<th>Generator Man-made</th>
<th>Control Man-made</th>
<th>Discriminator Natural</th>
<th>Generator Natural</th>
<th>Control Natural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>0.756</td>
<td>0.75</td>
<td>0.767</td>
<td>0.753</td>
<td>0.749</td>
<td>0.765</td>
</tr>
<tr>
<td>Layer 2</td>
<td>0.745</td>
<td>0.743</td>
<td>0.763</td>
<td>0.748</td>
<td>0.743</td>
<td>0.761</td>
</tr>
<tr>
<td>Layer 3</td>
<td>0.736</td>
<td>0.733</td>
<td>0.759</td>
<td>0.740</td>
<td>0.732</td>
<td>0.756</td>
</tr>
<tr>
<td>Layer 4</td>
<td>0.728</td>
<td>0.728</td>
<td>0.752</td>
<td>0.731</td>
<td>0.727</td>
<td>0.750</td>
</tr>
<tr>
<td>Layer 5</td>
<td>0.72</td>
<td>0.719</td>
<td>0.744</td>
<td>0.725</td>
<td>0.719</td>
<td>0.742</td>
</tr>
<tr>
<td>Output</td>
<td>0.718</td>
<td>0.714</td>
<td>0.741</td>
<td>0.721</td>
<td>0.714</td>
<td>0.737</td>
</tr>
</tbody>
</table>

We can see in Table 5.5 that there is even considerably greater drop off using features from one objective to classify another. This holds true in the control and the adversarial models, but it is drastically more so in the generator. Even when layers only contain features from a different objective, the generic nature of the features in the discriminator give it flexibility. This flexibility isn’t enough to bridge the gap with the traditional classifier, however. Like the control, the generator would likely have very specialized features for the domain for synthesis, but unlike the control it is the worst performer. The likely cause is that even if the generator is specialized to the specific domain and classes, the features learned for synthesis are not as applicable as those generic ones for discrimination.

Fig. 5.2 plots the activations at the final non-output layer for each model when given an unknown class, in this case class 5 from MNIST. The activations for the control classifier look much more compact than those for the discriminator, while the generator activations appear to cluster towards the edges of the input image. If we map the activa-
Figure 5.3. Class 5 activations mapped for classifier, discriminator and generator.

tions together like in Fig. 5.3, we can see the the scales are massively different with the classifier covering a large range and the generator being more tightly packed. The cause of this difference in scale is possibly down to the exact features learned and the types of features. The features in the classifier are specifically designed to classify, whereas those in the other two models are designed for other tasks.

5.7.2. Class Applicability

The setup for testing class applicability is similar to that of set applicability but instead we wish to measure how well features apply to individual classes. In this case, each class is compared to all the rest from each set. Each test uses a model with base features trained on the objective known set. We train the base layers using a GAN and compare the results to a traditional classifier as the control. We measure the total class applicability at each layer using the equation presented in Equation 5.3.

In Figure 5.4 the class applicability for the MNIST dataset is mapped. The plots
are the average separability between all the classes at that specific layer. The results show a similar behavior to what [18] showed. We can see that as the layers, even in the simple model used for MNIST, get more specific, the divergence between the known and unknown sets grows further down the model. Figure 5.4 also shows that in this scenario each model is essentially a downward shift of the previous. The distance between the known and unknown plots are relatively the same for all three models.

Figure 5.5 gives a similar look but for CIFAR10 with the nonobjective plots as well. While the discriminator does show greater applicability compared to the generator, the difference is very small. In fact, there is occasional overlap between the two. This tells us that for a more complex problems whose features are more fine grain, there is little difference between how applicable a generator and discriminator’s features are to specific classes. It should be pointed out that both the generator and discriminator could easily have learned entirely different sets of features that just have the same applicability, but those sets of features have little need of being class specific. In Figure 5.6 we present the
class by class break down of applicability in the generator and the discriminator. We can see the clear separation between the natural and man-made classes, but the known and unknown natural applicability tends blend more in from a class view.

5.7.3. GAN Applicability

Our last experiment is to test the impact features have to the entire adversarial process. To this end, we trained a GAN on only the natural images of the CIFAR10 dataset. We then transferred both the generator and discriminator weights to a new GAN. The model is then trained on the nonobjective set, tests are run using both man-made and natural. We test the effects the learned features at each layer have by freezing that layer \( n \) and all the previous layers to ensure that the model is only applying the previously learned features. We measure the performance of the GAN by computing the FID score at the end.

Table 5.6 and 5.7 shows that the performance of the GAN on CIFAR10 dramatically decreases as the specificity of prior features increases. This is shown by the average
Figure 5.6. Layer by layer class applicability for discriminator and generators on CIFAR10. Left: Class applicability for discriminator. Right: Class applicability for generator.

FID of 99.0 and 95.3 when all the layers in the discriminator and generator, excluding the final output layer, are frozen and can only use the prior knowledge. These results go to reinforce the conclusions that even during adversarial training, when there is no consideration for input class, the generator and discriminator learn some class specific features, while exhibiting the general to specific flow that is seen in classification networks. It should be noted that if each model had its hyperparameters tuned though more rigorous hyperparameter tuning, it could improve the FID scores.
Table 5.6. FID scores for natural portion of CIFAR10.

<table>
<thead>
<tr>
<th></th>
<th>Objective Known</th>
<th>Objective Unknown</th>
<th>Nonobjective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Natural</td>
<td>Natural</td>
<td>Natural</td>
</tr>
<tr>
<td>Disc 1</td>
<td>76.3</td>
<td>79.5</td>
<td>90.3</td>
</tr>
<tr>
<td>Disc 2</td>
<td>79.9</td>
<td>81.5</td>
<td>96.0</td>
</tr>
<tr>
<td>Disc 3</td>
<td>85.1</td>
<td>87.7</td>
<td>101.9</td>
</tr>
<tr>
<td>Disc 4</td>
<td>88.7</td>
<td>92.4</td>
<td>102.6</td>
</tr>
<tr>
<td>Disc 5</td>
<td>95.8</td>
<td>99.3</td>
<td>104.5</td>
</tr>
<tr>
<td>Gen 1</td>
<td>74.3</td>
<td>78.0</td>
<td>85.1</td>
</tr>
<tr>
<td>Gen 2</td>
<td>76.1</td>
<td>79.9</td>
<td>89.1</td>
</tr>
<tr>
<td>Gen 3</td>
<td>80.4</td>
<td>84.8</td>
<td>94.8</td>
</tr>
<tr>
<td>Gen 4</td>
<td>84.4</td>
<td>90.0</td>
<td>100.4</td>
</tr>
<tr>
<td>Gen 5</td>
<td>90.8</td>
<td>93.1</td>
<td>103.6</td>
</tr>
</tbody>
</table>

Table 5.7. FID scores for Man-made portion of CIFAR10.

<table>
<thead>
<tr>
<th></th>
<th>Objective Known</th>
<th>Objective Unknown</th>
<th>Nonobjective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Man-made</td>
<td>Man-made</td>
<td>Man-made</td>
</tr>
<tr>
<td>Disc 1</td>
<td>74.3</td>
<td>79.8</td>
<td>85.3</td>
</tr>
<tr>
<td>Disc 2</td>
<td>77.7</td>
<td>80.8</td>
<td>90.1</td>
</tr>
<tr>
<td>Disc 3</td>
<td>83.4</td>
<td>86.7</td>
<td>96.7</td>
</tr>
<tr>
<td>Disc 4</td>
<td>86.5</td>
<td>91.1</td>
<td>100.2</td>
</tr>
<tr>
<td>Disc 5</td>
<td>94.2</td>
<td>98.8</td>
<td>101.8</td>
</tr>
<tr>
<td>Gen 1</td>
<td>73.3</td>
<td>77.2</td>
<td>82.5</td>
</tr>
<tr>
<td>Gen 2</td>
<td>75.4</td>
<td>79.4</td>
<td>87.4</td>
</tr>
<tr>
<td>Gen 3</td>
<td>80.7</td>
<td>83.2</td>
<td>92.8</td>
</tr>
<tr>
<td>Gen 4</td>
<td>84.6</td>
<td>88.8</td>
<td>98.2</td>
</tr>
<tr>
<td>Gen 5</td>
<td>89.9</td>
<td>92.3</td>
<td>101.9</td>
</tr>
</tbody>
</table>

Furthermore, we can see that unlike the previous tests, which measured how the features learned during the adversarial training applied to classifying specific classes, in this test the generator outperforms the discriminator. To visualize this, observe in Figure 5.7 the GAN outputs for classes 5-9 when using base features from 0-4. We can see that the generator will still synthesize images using its original set of features that were transferred to it. The generator only receives noise as input and the gradients represent whether or not the discriminator guessed if the image was real or fake correctly. In this
scenario, because the generator already outputs images that are very close in feature space to those tested in the discriminator, the loss is small from the start. The generator then doesn’t learn to generate new classes, but instead seems to converge to producing the classes that would fool the discriminator most often. Even when using nonobjective features from CIFAR10, transferring the generator weights performs better than the discriminator; though the improvement is not as great.

Figure 5.7. Generator outputs for a GAN with various layers trained on the 0-4 classes and transferred to a GAN being trained on 5-9.

5.8. Generative CactusNets

We use the CactusNet created in chapter 3 as both the generator and discriminator, separately, to measure create generative CactusNets. To create the generator from the trained CactusNet we reverse the process shown in Fig. 6.2. We transfer the trained weights from the trained convolutional CactusNet to the corresponding convolutional layers in the generators; each followed by an upsampling layer. The last two fully connected layers do not get transferred over and are instead substituted by a final convolutional layer.
that outputs the final image. An input layer is also added that takes in a latent vector. The generator is briefly trained with only the last layer unfrozen. The other layers in the CactusNet are frozen to preserve the original weights. When the CactusNet is the generator the discriminator is a standard convolutional network designed to discern real from fake.

When using the trained CactusNet as the discriminator the only modification that needs to be made is that the final layer needs to be replaced. For each branch of the CactusNet, the final fully connected layer is changed to a softmax one with a single neuron to give the probability of real or fake. When using the trained CactusNet as the discriminator, we use an 5 layer up-convolutional network that takes in a latent vector as the generator. The CactusNet discriminator layers are frozen while training the generator to preserve the original weights.

To measure the quality of images produced by both GANs quantitatively, we get the total classification accuracy when using a trained convolutional neural network to classify the generated images. This is a common practice in quantitative analysis of GANs. We use a newly trained convolution neural network that has been trained equally on all the classes the generative CactusNet learned to generate. We choose not to use the CactusNet from section chapter 3 because the features learned in that network are used it generate images in the generative CactusNet. This could lead to some artificially higher scores. A GAN that has a generator and a discriminator with five convolutional layers each in addition to the output layers is used for the control. This structure is identical to the main branch of the CactusNet and a common architecture that we believe provides a good control for the CactusNets.
<table>
<thead>
<tr>
<th>Class</th>
<th>Control</th>
<th>Generator</th>
<th>Discriminator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>0.921</td>
<td>0.918</td>
<td>0.920</td>
</tr>
<tr>
<td>18-Wheeler</td>
<td>0.935</td>
<td>0.929</td>
<td>0.937</td>
</tr>
<tr>
<td>Tractor</td>
<td>0.914</td>
<td>0.909</td>
<td>0.911</td>
</tr>
<tr>
<td>Pickup</td>
<td>0.929</td>
<td>0.925</td>
<td>0.927</td>
</tr>
<tr>
<td>Airplane</td>
<td>0.929</td>
<td>0.914</td>
<td>0.932</td>
</tr>
<tr>
<td>Bicycle</td>
<td>0.935</td>
<td>0.936</td>
<td>0.933</td>
</tr>
<tr>
<td>House</td>
<td>0.931</td>
<td>0.919</td>
<td>0.928</td>
</tr>
<tr>
<td>Skyscraper</td>
<td>0.939</td>
<td>0.933</td>
<td>0.937</td>
</tr>
<tr>
<td>Boat</td>
<td>0.963</td>
<td>0.959</td>
<td>0.959</td>
</tr>
<tr>
<td>Basketball</td>
<td>0.941</td>
<td>0.924</td>
<td>0.938</td>
</tr>
<tr>
<td>Giraffe</td>
<td>0.966</td>
<td>0.961</td>
<td>0.966</td>
</tr>
<tr>
<td>Bear</td>
<td>0.957</td>
<td>0.952</td>
<td>0.958</td>
</tr>
<tr>
<td>Cow</td>
<td>0.978</td>
<td>0.966</td>
<td>0.972</td>
</tr>
<tr>
<td>Dolphin</td>
<td>0.949</td>
<td>0.933</td>
<td>0.941</td>
</tr>
<tr>
<td>Cardinal</td>
<td>0.951</td>
<td>0.948</td>
<td>0.949</td>
</tr>
<tr>
<td>Stingray</td>
<td>0.952</td>
<td>0.953</td>
<td>0.955</td>
</tr>
<tr>
<td>Beetle</td>
<td>0.958</td>
<td>0.945</td>
<td>0.955</td>
</tr>
<tr>
<td>Sponge</td>
<td>0.947</td>
<td>0.930</td>
<td>0.945</td>
</tr>
<tr>
<td>Cactus</td>
<td>0.974</td>
<td>0.971</td>
<td>0.977</td>
</tr>
<tr>
<td>Flower</td>
<td>0.981</td>
<td>0.978</td>
<td>0.980</td>
</tr>
<tr>
<td>Non-Objective</td>
<td>0.934</td>
<td>0.927</td>
<td>0.932</td>
</tr>
<tr>
<td>Objective</td>
<td>0.961</td>
<td>0.954</td>
<td>0.960</td>
</tr>
<tr>
<td>Average</td>
<td>0.948</td>
<td>0.941</td>
<td>0.946</td>
</tr>
</tbody>
</table>

tusNet are both trained on an individual class, with the network being retrained for every new class in the experiment.

Table 5.8 gives the accuracy for every class that the CactusNet created in section 3.4 was trained on. Both generative CactusNet types (discriminator and generator) perform well with an average accuracy above 0.95. The discriminator performs comparably well to the control, but the generator performs notably worse. The control and the discriminator CactusNet have a generator whose learned features are dedicated to and individual class and generation, while the generator CactusNet’s learned features are not as specialized. Because the discriminators task is closer to classification the discriminator
CactusNet is able to perform closer to the control.

Interestingly classifying generated objective (natural) images are notably more accurate than non-objective (man-made) images. This is reflected in both the generator and discriminator CactusNets, but more drastically in the generator CactusNet. This suggests that the applicability drop off between objective and non-objective, even in the CactusNet, results in diminished image quality.
Chapter 6. Progressively Growing GANs

6.1. Introduction

Due to the massive, and increasing, amount of satellite data available, a significant effort has been devoted to developing machine learning methods for satellite image processing. Among the higher level products sought, rooftop detection has received particular attention due to the diverse insights available from rooftop products. Rooftop detection is used to track urban growth, estimate population, assess damage from natural disasters and classify land use, among other applications.

Training rooftop segmentation models presents challenges, like the similar appearance of rooftops to other objects such as cars. Rooftops also have dissimilar appearances from city to city. Building shape, building material, and surrounding land cover vary widely from scene to scene and present challenges for transfer of models between cities. As such, no generalizable model yet exists that can accurately detect roofs in the full population of satellite images.

Remote sensing provides one of the fastest, lowest cost methods to gather information about damaged areas in post-disaster damage assessment. Automated generation of
high-resolution rooftop products creates a running inventory of assets, which can be lever-aged to track damages.

However, generation of high-resolution images presents challenges for tradition-ally trained deep neural networks. The existence of false positives or multiple foot prints blending together can give in accurate assessments. This is particularly a problem in con-densed and noisy urban areas, where traditional neural network methods struggle. In this paper, we aim to solve the short comings of neural networks applied to rooftop segmen-tation by adapting the progressive training of a generative adversarial network (GAN) [45] to segmentation. We introduce progressive training to the decoder of and encoder-decoder generator, while allowing the full encoder to learn the best latent encoding for the map. We evaluate the efficacy of rooftop segmentation using multi-spectral satellite images and show how using progressive training can limit the percentage of false positives and product blending while still producing accurate segmentations. This is, to the best of our knowl-edge, the first results of progressive training for semantic segmentation. A preliminary version of this paper was published in DMESS 2018, a satellite workshop of ICDM 2018.

The GAN [30] consists of a generator and a discriminator, which are linked through an ad-versarial training algorithm. The generator learns to generate mappings from the input to the target and the discriminator learns to evaluate them. Feedback from the discrimina-tor enables the generator to produce highly realistic outputs. We employ U-Net architec-ture, a convolutional neural network consisting of an encoder-decoder, as the generator. We apply progressive growing of the generator and the discriminator. Progressive growing is a transfer learning process wherein increasingly deep networks are trained to learn in-creasingly complex features. Accuracy of rooftop classification is assessed and results are
Figure 6.2. Schematic flow diagram of the data preparation and proposed model that contains U-Net architecture as the generator.

compared with those of a traditionally trained generative model and with those of non-generative U-Net. Our progressively trained GAN approach beats both traditional GAN and non-generative U-Net in accuracy, by four percent and eight percent respectively on the Spacenet [1] dataset, and by two percent and six percent respectively on the xView [50] dataset.

6.2. Related Work

Significant accomplishments have been made in computer vision, resulting in increasingly effective state-of-the-art methods for image processing [44, 7]. Early efforts in automatic rooftop segmentation used methods like edge detection, corner detection, and segmentation into homogeneous regions via k-means clustering or support Vector Machines (SVM) to identify candidate rooftops in [42]. Discriminative features used to evaluate can-
didate rooftops include building shadows, geometry, and spectral characteristics [65, 41]. Several approaches have used LIDAR alone or in addition to multi-spectral images [70, 12]. Newer-generation machine learning techniques [6] have also been applied in satellite image classification [4, 52] and in rooftop segmentation specifically [3, 15]. Convolutional neural networks (CNNs) have greatly improved the state-of-the-art in semantic segmentation tasks wherein each pixel in an image is associated with a class label [56]. High-resolution rooftop detection presents a dense prediction problem in which proper pixel-wise labeling is paramount to a produce a product with well-defined rooftops. In [46], stacked U-Nets were used that enhanced the results of the previous U-Net. This study found that stacking of just two CNNs outperforms the state-of-the-art method. Introduced in 2015, U-Nets utilize skip connections and an encoder-decoder structure to learn a latent translation from input to output [67]. CNN performance is sometimes hampered by blurry results, which satisfy the loss function by reducing the Euclidean distance between predictions and the target [63]. Generative adversarial networks (GAN) address this pitfall by simultaneously training a discriminator network to differentiate between real and generated images [30]. The original classic GAN algorithm [30] is further improved upon by progressively grown GANs [45]. In working with high-resolution images, GANs run into issues with real and generated images being too easy to discriminate. Progressively grown GANs address this challenge by utilizing transfer learning in deep neural networks [45].

A preliminary version of this paper appeared in [22]. The present version extends that in [22] by first introducing a modified algorithm for progressive training that includes smooth fading. This method doubles the number of training cycles by introducing a residual connection over the new layer to the output layer. This step aims to increase the
accuracy by preserving the features learned in the previous layer during the addition of
the new layer. Additionally, we further evaluate the methodology on the xView dataset.
xView introduces noisy masks with large areas of background noise labeled with positive
pixels which helps further evaluate the progressively trained model’s ability to minimize
false positives and blending.

6.3. Data Preparation

Our experiments are run on the SpaceNet version 2 dataset [1], and the xView
dataset [50]. These datasets contains high resolution commercial satellite images along
with the masks of building and road footprints, as depicted in Figure 6.1. The following
experiments are run strictly on rooftop segmentation for both SpaceNet and xView. Both
datasets, that we have used, are limited to the greater Las Vegas area. We leave other
datasets and class segmentations for future work and evaluation. The xView dataset does
not contain segmentation masks, but ROI’s (region of interests) contained in a geojson
file. To circumvent this, we translate the ROI’s for each image into an image mask. The
positive segments of the mask have a 1 to 1 translation to the area of the image contained
with an ROI. The resulting masks are different from those in the SpaceNet data set be-
cause they do not snap to the building foot prints. In xView, all the ROIs are rectangles
aligned with the image axis. The results can leave artifacts and background in the seg-
mentation that are not actually part of a building footprint. Each mask is paired with the
original image and then split into train and test data to complete the dataset. Figure 6.2
gives the flow of our data preparation for converting a ground truth geojson to a binary
mask.
The xview dataset suffers from class imbalance. Building footprints makeup only a portion of the objects contained in xView. If not handled, then the majority of the train and test datasets would contain masks with no positive examples, leading to poor performance. To get around this, we created our own masks using the ROI’s for classes that we identified as being buildings with rooftops. The shortfall of this dataset is that the building footprints are not exact outlines of the buildings, but just segment out their area. This provides a different challenge compared to the SpaceNet dataset.

6.4. Proposed Method

Our training algorithm incorporates two primary components: adversarial training and progressive growing. Our method is unique to previous works in progressive growing due to the architecture of the generator and the discriminator [45]. In previous works the generator and discriminator mirror one another; in our model, the generator instead has an encoder-decoder structure. Our proposed model’s architecture and progressive growth are presented in Figure 6.2.

6.4.1. Network Architecture

Many out of the box segmentation models use the U-Net architecture because of its ability to learn a latent translation between the input and target sets. Additionally, mirrored layers in U-Net contain skip connections that allow structural information to be preserved when decoding from the learned latent encoding. This architecture has become a common generator structure in many domains of GANs. It is for these reasons along with its popularity that we have chosen to use the U-Net architecture in our framework as well.
6.4.2. GAN training

In the most basic form of a GAN, the generator learns a mapping of \( z \rightarrow y \), where \( z \) is some random latent vector that is translated onto the feature space defined by the task \( y \). If a GAN is being used to translate one image to another, then the task of the generator is to learn a mapping \( x \rightarrow y \) from input set \( x \) to target \( y \). This is done by mapping \( x \) to a latent encoding \( z \), \( x \rightarrow z \), which can be decoded to \( y \), \( z \rightarrow y \). In our case we seek to learn a mapping between a high resolution satellite image and the rooftop segment of that image. GANs learn these mappings between inputs and targets via a min/max game, \( \min_{\Gamma} \max_{\Delta} (L(\Gamma, \Delta)) \), played between the generator \( \Gamma \), with inputs \( x \) and \( z \) expressed as \( \Gamma(x, z) \), and the discriminator \( \Delta \), with inputs \( x \) and \( \Gamma(x, z) \) denoted by \( \Delta(x, \Gamma(x, z)) \), with loss \( L(\Gamma, \Delta) \). We express the standard GAN’s objective function as [30]:

\[
\min_{\Gamma} \max_{\Delta} L(\Gamma, \Delta) = \mathbb{E}_y[\log_{10}\Delta(y)] + \mathbb{E}_{x,z}[\log_{10}(1 - \Delta(x, \Gamma(x, z)))] \quad (6.1)
\]

In the case of segmentation, we desire the outputs of our generator to be as near as practicable to the ground truth mask. To do this we add the \( L_1 \) distance to the objective:

\[
L_1(\Gamma) = \mathbb{E}_{x,y,z} [|y - \Gamma(x, z)|] \quad (2)
\]

This imposes a second objective for generator’s output: to mirror the ground truth by forcing the generator to minimize the absolute distance from its output to the ground truth mask. Absolute error (\( L_1 \) distance) is used rather than mean squared error (\( L_2 \) distance) to discourage blurring.
6.4.3. Progressive Growing

In a progressive growing algorithm, layers are added to the generator and discriminator as training moves forward. As layers are added to the networks, generated images increase in spatial resolution. While all layers remain trainable throughout the training period, progressive growing allows Generator $\Gamma$ and Discriminator $\Delta$ to learn increasingly fine scaled features on increasingly high resolution images. Learning step by step presents a series of simpler tasks to the model. Progressive training is consequently more stable and more efficient than traditional training. Layers in progressive growing are not added to the network for training one after another. Instead, layers are added using a technique called smooth fading. In smooth fading, new higher resolution layers are added in two steps. First the new layer is added to the network, but treated as a residual block together with a skip connection. For the encoder in the generator, the upsampled encoding is passed through an RGB output layer and merged with the RGB output of the new high resolution layer to generate a faded output that is input to the discriminator. In the discriminator, the faded output from the generator feeds into both the new higher resolution layer and directly to the following lower resolution layer with a downsampling and skip connection. Progressive learning takes advantage of a deep neural networks’ ability to learn features from generic to specific, or low to high resolution. At each progressive step, the weights learned for all the layers in the last step are transferred to identical layers in the next step. This transfer leaves only one untrained layer at each step. By progressively adding layers, the network learns the features at each resolution independently, easing the learning task of each progressive network. We employ this technique to pro-
duce masks that mirror the input high resolution in sharpness. Traditionally, progressive
GANs are employed for generative tasks. We, however, seek to apply it to translation,
specifically segmentation. By using an encoder-decoder structure in the generator, we rely
on the encoder to map the high resolution input to a latent vector which is translated by
the decoder. Like in traditional progressively growing GANs, the decoder is progressively
trained. Because we desire the decoder to decode from a latent vector containing all the
information contained in the high spatial resolution of our input, the encoder is not pro-
gressively grown. The encoder instead maintains its full structure throughout the progres-
sive training. The discriminator grows in sequence with the decoder. This trains each suc-
cessive layer to discriminate specific resolutions.

6.5. Experimental Evaluation

In this section, we compare the results of our progressive GAN model to that from
a standard U-Net model and a traditionally-trained GAN model [30] that is not progres-
sively trained. We choose to use the U-Net model with residual connections as it is a tra-
ditional model that has been well researched and adapted to segmentation many times
[56, 46]. The U-Net is also built identically to the generator used in the progressive GAN,
allowing us to further isolate the effects of progressive training. Similarly, we use a stan-
dard GAN built identically to our progressive GAN to discern the difference between stan-
dard training and progressive training. We compare the results both visually and numeri-
cally by taking the per-pixel error of the masks.
Figure 6.3. Results of applying the three tested methods to sample images along with the input and the ground truth mask.

### 6.5.1. Implementation Details

For our experiment, the U-Net model has an encoder-decoder architecture. The encoder is built of 8 hidden layers with 64, 128, 256, 512, 512, 512, 512, and 512 hidden units per layer. The decoder is built of 8 hidden layers that mirror the encoder. This U-Net is used as the generator of the GAN. The GAN discriminator is built with the same architecture as the decoder, and grows in conjunction with it. Batch normalization with momentum=0.9 and dropout with probability=0.5 are employed during training to discourage overfitting.

### 6.5.2. Results

From Figure 6.3, we can see that model inferences of rooftop location for SpaceNet generally match the size and shape of ground truth. While border regions of rooftops leave
room for improvement, we show progressive growing improves the definition of individual buildings compared with its counterparts. In the two non-progressive methods, we can see that the building segments tend to blend together more than in progressive growing. Additionally, we can see that the standard GAN [30] and non-GAN approaches suffer from false positives where the progressive growing is able to minimize this. The xView dataset poses a more difficult problem for the progressive model as we can see in Figure 6.4. Because xView is not naturally made for segmentation, the footprints in the masks blend
together and background features commonly get caught in the footprint. This can cause the accuracy to fall off, with progressive GAN showing high rates of false negatives.

Figure 6.5. False negative results from applying progressively trained model.

The progressive model shows the ability to limit false positives compared to the other methods. The cause of this is due to the specificity of features in the later layers. This results in the progressively trained model preferring to not label pixels over committing false positives. Figure 6.5 presents progressive GAN output demonstrating this phenomenon. As a whole, the progressively trained GAN produces building footprints that snap to the original nicely while also minimizing the amount of false positive pixels compared to the standard methods. This is more conspicuous with the xView dataset. The mixing of background and building features confuses the progressive model, which tends heavily towards not guessing, as it has difficulty separating foreground from background, due to the training dataset limitations.

We present our accuracy scores as the per-pixel error between the ground truth mask and the masks produced by our models. The per-pixel provides a good view of how
well the produced masks fit to the high resolution buildings. From Table 6.1 we can see that the progressively trained GAN outperforms its counterparts in this metric. We also present both the training and testing accuracy of our models to verify that none have over-fit to the dataset.

In Figure 6.6, we present graphs for the accuracy of each method during training on SpaceNet. We can see that for the progressively trained GAN that each progressive step builds on top of the previous. The decreased loss and quicker convergence at each step show that there is good transfer of knowledge between the previous and successive steps. Another interesting result is the closeness of the higher resolution layers. This suggests that there exists an image resolution, in our case $256 \times 256$ pixels, for which all following image resolutions cannot be used to learn increasingly fine features.

6.6. Applications in Earth Science

Multiple scientific and social domains call for high-resolution rooftop products. Production of these products relies on both data collection and accurate methods for semantic segmentation. While some prior efforts have used physics and/or texture-based models, or additional data like elevation data, we focus on an approach relying only on
multi-spectral reflectance. Images that are only in the visual spectrum are even more ubiquitous, however a rooftop detection model that uses multi-spectral images is broadly applicable. Multi-spectral satellite images are publicly available from multiple satellites with global coverage and record length stretching back up to over a decade. These images can be used to develop a time series of rooftop data with multiple impactful uses, such as rapid change detection in satellite images in the aftermath of natural disasters.

The results presented demonstrate an improvement in semantic segmentation performance by GANs using progressive growing. This offers a step toward improvements in high-resolution segmentation. Applications such as 3D city modeling demand high-pixel accuracy rooftop products, and much space for improvement remains. The challenge of training a generalizable rooftop detection model also remains. In addition to variation in appearance of rooftops and non-rooftop area from city to city, buildings vary widely in appearance between highly developed areas and informal settlements like slums. Progressive growing could offer insights into which learnable features and layers are generally applicable to all cities, allowing quicker training for new locations.
Chapter 7. Conclusions

We cover four primary area in this dissertation, three cover applicability and the last covers progressive generative adversarial networks. First, we defined applicability, quantified it and demonstrated its connection to transferability of features. Applicability was expanded to image octaves in what we call octave resonance. We performed, to the best of our knowledge, the first feature analysis of GANs using applicability to compare the learned features between generators and discriminators. Lastly, we used applicability to create an efficient and self-growing deep neural network, called the CactusNet, that can perform unsupervised learning with efficient reuse of learned parameters. Second, applicability is expanded to octaves defining what we call octave resonance. By measuring the octave resonance of neural networks trained on constant and varying octave dataset, we showed that shifting the octaves of images can cause considerable drop off in applicability even when the image is still recognizable to the human eye. CactusNets were expanded and their effectiveness demonstrated on octaves. In our last area of applicability, we presented insights into the types of features that generators and discriminators learn and how applicable they are to a classification task and how CactusNets can be used as GANs.

In the last chapter, we presents a novel approach to semantic segmentation for high resolution satellite imagery that draws upon recently developed machine learning techniques. We use progressive training for semantic segmentation of rooftop products to create tighter fitting segmentation masks with less false positives that previous approaches. Our method was tested on both the SpaceNet and xView datasets. Each posed their own unique set of challenges for the model, such as the background noise being labeled as positive pixels in the xView ground truth. The experiments show that using progressive train-
ing is indeed able to control the percentage of false positives and product blending, even in the noisy xView data. The trade off to this approach is that there is the potential for more false negatives than the traditional U-Net approaches tested.
Bibliography

[1] Spacenet on amazon web services (aws) (2018),
https://spacenetchallenge.github.io/datasets/datasetHomePage.html


Edward Durham Collier

Employment History

2018 – Present  Solutions Engineer, Rhombus Power Inc., Moffett Field, CA.
2016 – 2021  Research Assistant Computer Science Department, School of Engineering and Compu-
ter Science, Louisiana State University.
2018 – 2018  Research Intern Bay Area Environmental Research Institute, Moffett Field, CA.
2014 – 2016  Research Assistant Computer Graphics Technology Department, School of Technol-
ogy, Purdue University.
2012 – 2012  Software Engineering Intern Hewlett Packard, Austin, TX.

Education

2016 – 2021  Ph.D., Computer Science, Louisiana State University
Dissertation title: Quantifying Feature Overlaps in Deep Neural Networks and their Applica-
tions in Unsupervised Learning and Generative Adversarial Networks.
2014 – 2016  M.S., Computer Graphics Technology, Purdue University
Thesis title: Avian Retinal Photoreceptor Detection and Classi-
fication using Convolutional Neural Networks.
2010 – 2014  B.S., Computer Science, Baylor University

Research Publications

Journal Articles


Conference Proceedings


Skills

Coding - Java, C/C++, Python, R, SQL, Matlab, LTeX, ...
Databases - MySQL, PostgreSQL, HSQL, SQLite.
Web Dev - HTML, css, JavaScript, Apache Web Server.
Misc. - Academic research, teaching, training, LTeX typesetting and publishing.

Miscellaneous Experience

Certification

2021 - Security+. Awarded by CompTIA.