Semi-supervised Regression with Generative Adversarial Networks Using Minimal Labeled Data

by

Greg Olmschenk

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2019

Semi-supervised Regression with Generative Adversarial Networks Using Minimal Labeled Data

by

Greg Olmschenk

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

	Professor Zhigang Zhu
Date	Chair of Examining Committee
	Professor Ping Ji
Date	Executive Officer

Hao Tang

Jie Gong

Ioannis Stamos

Jie Wei

Supervisory Committee

The City University of New York

Abstract

Semi-supervised Regression with Generative Adversarial Networks Using Minimal Labeled Data

by

Greg Olmschenk

Adviser: Professor Zhigang Zhu

This work studies the generalization of semi-supervised generative adversarial networks (GANs) to regression tasks. A novel feature layer contrasting optimization function, in conjunction with a feature matching optimization, allows the adversarial network to learn from unannotated data and thereby reduce the number of labels required to train a predictive network. An analysis of simulated training conditions is performed to explore the capabilities and limitations of the method. In concert with the semi-supervised regression GANs, an improved label topology and upsampling technique for multi-target regression tasks are shown to reduce data requirements. Improvements are demonstrated on a wide variety of vision tasks, including dense crowd counting, age estimation, and automotive steering angle prediction. With training data limitations arguably being the most restrictive component of deep learning, methods which reduce data requirements hold immense value. The methods proposed here are general-purpose and can be incorporated into existing network architectures with little or no modifications to the existing structure.

Acknowledgments

For the continual mentorship, guidance, and support throughout this work, I would like to thank my advisor Professor Zhigang Zhu (City College of New York). Most importantly, Professor Zhu provided me with critical feedback, which showed where my research was weakest and kept me on track. Similarly, I would like to thank Professor Hao Tang (Borough of Manhattan Community College) for the extensive guidance given and for providing his support whenever it was needed.

Next, I would like to thank the remaining dissertation committee members, most of whom are offering their feedback and time without receiving anything in return. Professor Jie Gong (Rutgers University) has a focus on infrastructure modeling (among other work), and our collaboration in facility analytics directly led to the crowd analysis portion of this work. Professor Jie Wei (City College of New York) has frequently worked alongside our lab, and a project in his course on Computer Vision and Image Processing led my focus toward semi-supervised training for regression tasks. Professor Ioannis Stamos (Hunter College) is an expert in 3D computer vision, and his course in 3D Photography directed me toward scene analysis with a 3D perspective which played a significant role in this work.

I would also like to thank my research colleagues, especially those from the City College Visual Computing Lab, who have given me feedback and have collaborated in my work.

Finally, I would like to thank my family and friends, especially my parents for their continual support and my partner for enduring my schedule.

This work has been supported by various grants, agencies, and companies. This includes three National Science Foundation Awards: EFRI #1137172 from the Emerging Frontiers in Research and Innovation Program, PFI #1827505 from the Partnerships for Innovations Program, and SCC-Planning #1737533 from the Smart and Connected Community Program. They all have played a significant role in both my development in machine learning and in facility analysis for serving people in special needs specifically. Appointments to the U.S. Department of Homeland Security (DHS) Science & Technology Directorate Office of University Programs, administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and DHS lead to my first regression-based deep-learning projects and played a major role in supporting the crowd analysis portion of this work for security applications. ORISE is managed by ORAU under DOE contract number DE-AC05-06OR23100 and DE-SC0014664. The research was also supported by Bentley Systems, Incorporated, through a CUNY-Bentley Collaborative Research Agreement (CRA), whose focus on facility modeling and digital twins allowed this project to move forward with real-world considerations. Additional travel support was provided by the Defense Intelligence Agency via the Rutgers University Consortium for Critical Technology Studies and by a Doctoral Research Award provided by the CUNY Graduate Center, providing the attendance of conferences to present our work. Computing resources were partially supported by a Microsoft Azure for Research Award.

Contents

Notation

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
A	A tensor
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable
$P(\mathrm{a})$	A probability distribution over a discrete random variable
$p(\mathrm{a})$	A probability distribution over a continuous random variable, or over a variable whose type has not been specified

$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{x \sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(x)$
$\mathcal{U}(a, b)$	Uniform probability distribution over the range from a to b
$\mathcal{N}$	Normal probability distribution with mean 0 and variance 1
$a + \mathbf{A}$	For any scalar operator combining a scalar and multi-dimensional value, the operation is applied element-wise to the multi-dimensional value. This treatment of operations is also referred to as broadcasting.
$D(\boldsymbol{x})$	Discriminator network function forward pass with input $\boldsymbol{x}$
$G(\boldsymbol{z})$	Generator network function forward pass with input $\boldsymbol{z}$

Chapter 1

Introduction

1.1 Deep Learning and Regression

Deep learning [1], particularly deep neural networks (DNNs), has become the dominant focus in many areas of computer science in recent years. This is especially true in computer vision, where the advent of convolutional neural networks (CNNs) [2] has led to algorithms which can outperform humans in many vision tasks [3]. Today, deep learning plays a primary role in nearly every area which uses machine learning methods.

The most common use of DNNs is for classification tasks. These tasks consist of labeling some form of input data as belonging to one or more discrete categories or classes. For example, a classification task may comprise of applying a label to an image, where the label is the type of animal depicted in the image (e.g., 'dog', 'cat'). The vast majority of deep learning research is focused on these classification tasks. In contrast, a regression task consists of labeling some form of input data with a continuous value. For example, a regression task may consist of determining the distance from a camera to an object, given an image from the camera. The distance may be any value in the continuous range $[0, \infty)$, as opposed to a

discrete label in the case of classification. Significantly less deep learning research has been devoted to regression problems as compared to classification problems. While the transition from classification to regression may initially seem to be a trivial extension, the current formulations of many deep learning methods do not lend themselves to use in regression tasks. While it is possible to frame any regression task in terms of a classification task, in practice, this presents several significant problems. For example, to recast a regression task as a classification task, an arbitrary number of classes must be chosen, as a continuous range can be segmented into discrete bins in an arbitrary number of ways. This discretizing of the range imposes a limitation on the precision which can be achieved. However, more importantly, the relative locality of the regression values is discarded when converting to a binning classification problem. Due to the formulation of DNN training cost functions, such a naive classification approach would result in each incorrect class being considered equally as wrong. For example, we can consider a prediction of a real number in the range $[0, 10)$ which has been split into 10 discrete classes. Given a true label of 10, a prediction of 8 would produce the same loss as a prediction of 2. This results in a problematic training gradient, as well as encouraging the network to overfit since only the single bin shows any improved predictive capabilities. The smaller the bins become, the more problematic this limitation becomes. Depending on the accuracy required by the application, this binning classification approach may be acceptable, but these problems are more naturally framed as the original regression tasks.

The set of regression tasks encompasses a nearly limitless supply of problems that cannot be solved or would be poorly solved by framing them as classification problems. Some examples include crowd counting estimation [4], weather prediction models [5], stock index evaluation [6], object distance estimation [7], age estimation [8], data hole filling [9], curve coefficient estimation, ecological biomass prediction [10], traffic flow density prediction [11], orbital mechanics predictions [12], electrical grid load prediction [13], stellar spectral analysis [14],

network data load prediction [15], object orientation estimation [16], species population prediction [17], ocean current prediction [18], and countless others. Machine learning has been utilized for each of these tasks. The novel method presented in this work can be generalized to any such regression problem, allowing each of these applications to take advantage of the semi-supervised method discussed below.

1.2 Semi-supervised Generative Adversarial Networks

Within the field of deep learning, there are generative models, and more specifically, generative adversarial neural networks [19]. A generative model is one which learns how to produce samples from a data distribution. In the case of computer vision, usually takes the form of a neural network which learns how to generate images, often with specified characteristics. Generative models are particularly interesting because for such a model to generate new examples of data from a distribution, the model needs to "understand" that data distribution. Arguably, the most powerful type of generative model is currently the generative adversarial network (GAN) [19, 20]. GANs have been shown to be capable of producing fake data that appears to be real to human evaluators. For example, fake images of objects which a human evaluator can not distinguish from real-world images of those objects.

Beyond the generation of fake data, GANs, using a semi-supervised approach, have been shown to produce better results in discriminative tasks using relatively small amounts of labeled data [21], where equivalent DNNs/CNNs would require significantly more labeled data to accomplish the same level of accuracy. This semi-supervised training allows a model architecture which can learn from unlabeled data in addition to labeled data. As one of the greatest obstacles in deep learning is acquiring or producing the vast amounts of labeled data required to train such models, the ability to train these powerful models with less labeled data is of immense import.

While GANs have already shown significant potential in semi-supervised training, they have only been used for a limited number of cases. In particular, they have thus far almost exclusively been used for classification problems. This work generalizes semi-supervised GANs to regression problems. The nature of a GAN makes moving from classification to regression problems non-trivial. Specifically, the two parts of a GAN can be seen as playing a minimax game. The discriminating portion of the GAN has the objective of discriminating between fake and real data in some fashion. However, when the real data is labeled with continuous-valued numbers, deciding on what constitutes a "fake" labeling is not straight forward. To solve this issue, novel loss functions are required. At the same time, these loss functions need to provide a stable training dynamic between the discriminator and generator. These problems, for which solutions are provided in this work, make semi-supervised GANs in regression problems a challenging area of research.

1.3 Contributions of the Thesis

This work presents the following contributions:

1. A novel algorithm generalizing semi-supervised GANs to regression tasks, the Semi-supervised Regression GAN (SR-GAN).

2. An analysis of optimization rules which allows for stable, consistent training when using the SR-GAN, including experiments demonstrating the importance of these rules.

3. Systematic experiments demonstrating the SR-GAN on three typical real-world applications of regression – age estimation, steering angle prediction, and crowd analysis, all from real-world images. A comparative analysis of alternative methods is also performed.

4. An improved labeling topology for crowd counting, a multitarget regression task, and an analysis of full label resolution for crowd counting.

For a better understanding of the work, we now provide a more detailed summary of each of the above contributions.

SR-GAN theory. The most important contribution is the generalization of semi-supervised GANs to regression in the form of the SR-GAN. As the primary limitation of deep neural networks is currently data limitations, easing this requirement on the countless existing regression problems is immensely important. In addition to the SR-GAN, as a comparison, this work also explores more naive approaches to regression GANs. The capabilities and limitations of these alternatives are demonstrated, and a theoretical understanding of its limitations is provided. The SR-GAN method is designed to avoid the limitations of the alternatives.

SR-GAN behaviors. While the theoretical solution for applying semi-supervised GANs to regression is provided in the first contribution, there are many factors that need to be addressed for this approach to work in practice. Chiefly, the details of loss functions used in the SR-GAN are analyzed. Several variants, each with their own theoretical justification, are examined and tested. This analysis provides both evidence of the SR-GAN's robust nature and a way to generalize the choice of hyperparameters used to train the SR-GAN.

SR-GAN applications. Next, this work provides several real-world applications where SR-GANs outperform traditional CNNs and other competing models. Specifically, the SR-GAN was applied to predict the age of individuals from single images, to predict the steering angle of a car based on a single frame of video, and to estimate crowd densities in surveillance footage. The age and steering angle estimation tasks provide real-world applications on which the SR-GAN can be used to reduce the data requirements, while still being simple enough to explain the training properties in detail. In the steering angle estimation, it is demonstrated that the SR-GAN approach performs better than previously proposed GAN methods. The

crowd analysis case provides a complex real-world application with immediate use cases and demonstrates the capabilities of the SR-GAN on open problems.

Beyond SR-GAN. In addition to the semi-supervised GAN for regression, two additional improvements are proposed to reduce the number of labeled examples required to train a system: improved label topology and use of full-resolution map labels. While these methods are general-purpose, this work only explores there use in the case of crowd analysis. We also discuss the integration of the SR-GAN into multi-optimization goal networks, where challenges are still present.

1.4 Thesis Organization

The remainder of this thesis will be divided into the following chapters.

First, background information will be discussed in Chapter 2. This will include an overview of the GAN and semi-supervised GAN concepts, early works in applying GANs to regression tasks, and the current state-of-the-art works for the specific applications that the SR-GAN will be demonstrated on.

Next, the motivation for this work will be detailed in Chapter 3. This will include a detailed analysis of the current state of GANs for regression, the limitations of these existing approaches, and the underlying theory explaining why another solution is required.

Chapter 4 will describe the methodology of the SR-GAN. Here a formal mathematical definition of this novel GAN method will be provided as well as an intuitive understanding of the approach.

In Chapter 5, a well-controlled synthesized dataset is defined on which can be tested the properties of the SR-GAN while controlling every aspect of the data. This approach allows us to directly observe the underlying data generating model and compare the output of the generator to it. Additionally, it allows us to artificially control the complexity of the dataset

allowing for a detailed analysis of the SR-GANs capabilities and limitations.

Chapter 6 provides a simple but real-world application for the SR-GAN, namely, age estimation of an individual given a single image. This application provides a direct understanding of the theoretical design of the model in a practical implementation. The simplicity allows the details of the model to easily be examined while still tackling a real task. In this chapter, improvements over a standard DNN method are the focus.

Similar to the problem of age estimation, Chapter 7 provides another simple real-world application. In this chapter, the network is given the task of predicting the steering angle of a car given an individual image of the upcoming road segment. This can be seen as a simplified version of the self-driving car problem. It provides yet another demonstration that the SR-GAN is a generalizable method. Here the focus is on the improvements of the SR-GAN over other potential GAN approaches.

Chapter 8 provides a complex, real-world application of the SR-GAN. Here the SR-GAN is used to predict crowd densities from surveillance images. Dense crowd counting is still a challenging problem and an open area of research. It is demonstrated that the SR-GAN model can reduce the data requirements of existing CNN state-of-the-art models, thereby improving accuracy overall. The reduction of data on this application is of immense import, as the manual effort required to annotate this kind of data is significant and error-prone.

Next, Chapter 9 provides additional methods beyond the SR-GAN in which crowd labels, and multitarget regression labels more generally, can be improved. This is achieved by an improved label topology and predictive label upsampling.

Finally, Chapter 10 concludes this work, with a discussion of the overall findings and possible implications.

Chapter 2

Background for Semi-Supervised Regression GANs

In 2012, a deep convolutional neural network called AlexNet [22] won the ImageNet Challenge [23], a competition measuring image classification accuracy. Since this time, deep learning techniques have been able to outperform traditional computer vision techniques in nearly any task where there is enough data to train the deep learning approaches. Furthermore, at the time of this writing, there is not an obvious limitation to capabilities of deep neural networks moving forward, other than the limitations of available training data and computing power. The work proposed here attempts to loosen the former limitation.

The development of generative adversarial networks (GANs) [20] specifically presents a means by which the amount of data required by a deep neural network can be lessened. A technique of using GANs in a semi-supervised fashion, where only a small portion of the dataset has labels, has been proposed and shown to produce high levels of accuracy relative to other methods using the same quantity of labeled data [21]. To better understand how this approach works, we will first give a brief explanation of GANs and then an explanation

of their use in a semi-supervised case.

The following sections give an explanation of GANs and semi-supervised GANs for classification. These explanations are useful for understanding the workings of the SR-GAN proposed here, but the previously existing methods are not directly applicable to regression, as will be shown.

2.1 Generative Adversarial Networks

A GAN consists of two neural networks which compete against one another. One of the networks generates fake data; hence, it is referred to as the generator. The other network attempts to distinguish between real data and the fake generated data; consequently, this network is called the discriminator. Both networks are trained together, each continually working to outperform the other and adapting in accordance to the other. In this way, both networks are essentially playing a minimax game [24].

Though GANs are now fairly common, to provide the groundwork for our SR-GAN, it is worth defining the details of a GAN from the viewpoint of probability distributions. First, an intuitive understanding of GANs is provided.

A conventional explanation of the competition between the generator and the discriminator is that of a counterfeiter and a detective. The counterfeiter, the generator, tries to manufacture counterfeit money. The detective, the discriminator, tries to figure out if a given piece of currency is real or fake. In this story, both start with no knowledge but are ready to learn. The generator begins by making a particularly bad example of currency, but as the discriminator is equally bad at determining true currency, it may label the fake example as real money. Here we step in and tell the discriminator it was wrong. The discriminator will then try to find something to help it distinguish between the real and fake data after being told it was wrong. The one remaining twist to the story is that the generator is provided full access

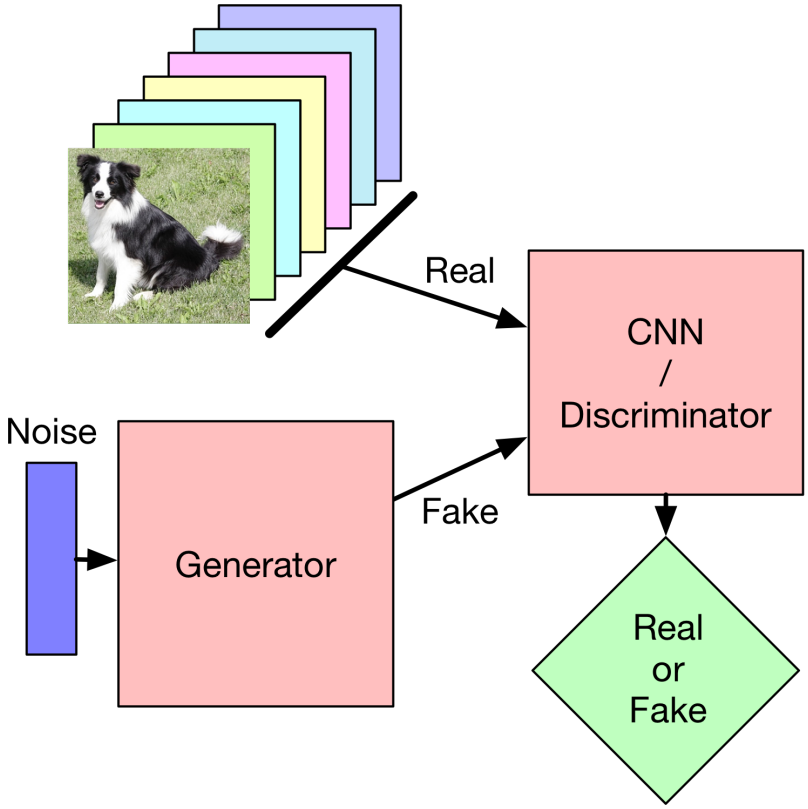


Figure 2.1: The structure of a basic GAN. Real and fake images are fed to a discriminator network, which tries to determine whether the images are real or fake. A generator network produces the fake images.

to how the discriminator determines which money is fake and which is real. In turn, it changes its counterfeiting approach to find flaws in the discriminator's new methods. For U.S. currency, the discriminator may first decide that the money needs to have the picture of a person. The generator will learn to make pictures of people. Then the discriminator may realize that it needs to be a specific person on the currency and so the generator will start to learn how to make that specific person's image. This process continues until both the generator and discriminator become very good at their jobs.

Now for a more formal description. To give a concrete understanding, the remainder of the explanations in this section is given in terms of computer vision problems, specifically where the datasets consist of images. This means an example of real data (and thus the

input of the discriminator) is an image and also the output of the generator is an image. The structure of such a GAN is shown in Figure 2.1.

The generator network takes random noise as input (usually sampled from a normal distribution) and outputs fake image data. The discriminator takes as input images and outputs a binary classification of either fake or real data. Images can be represented by a vector, with each element representing the value of a pixel in the image[1]. In any image, each element of this vector has a value representing the intensity of that pixel. For this explanation, a minimum element value (pixel value) of 0 is used, and a maximum of 1 is used. Of course, this vector can be represented as a point in N dimensional space, where N is the number of elements in the vector. The possible positions of an image's point are restricted to the N dimensional hypercube with a side length of 1. Here, it is important to note that real-world images are not equally spread throughout this cube. That is, most points in the cube correspond to images that would look like random noise to a human. Images from the real world usually have properties like local consistency in both texture and color, logical relative positioning of shapes, etc. Real-world images lie on a thin manifold within the cube [25]. Subsets of real-world images, such as the set of all images containing a dog, lie on yet a smaller manifold. This manifold represents a probability distribution of the real-world images. We can view the real world as a data generating probability distribution, with each position on the manifold having a certain probability based on how likely that image is to exist in the real world.

The goal of the generator is then to produce images which match the probability distribution of the manifold as closely as possible. Input to the generator is a point sampled from the probability distribution of (multidimensional) random normal noise, and the output is a point in the hypercube—an image. The generator is then a function which transforms a

[1]One element per pixel is in the case of grayscale images. For RGB images, there are three elements in the vector for each pixel, one for each color channel of the pixel.

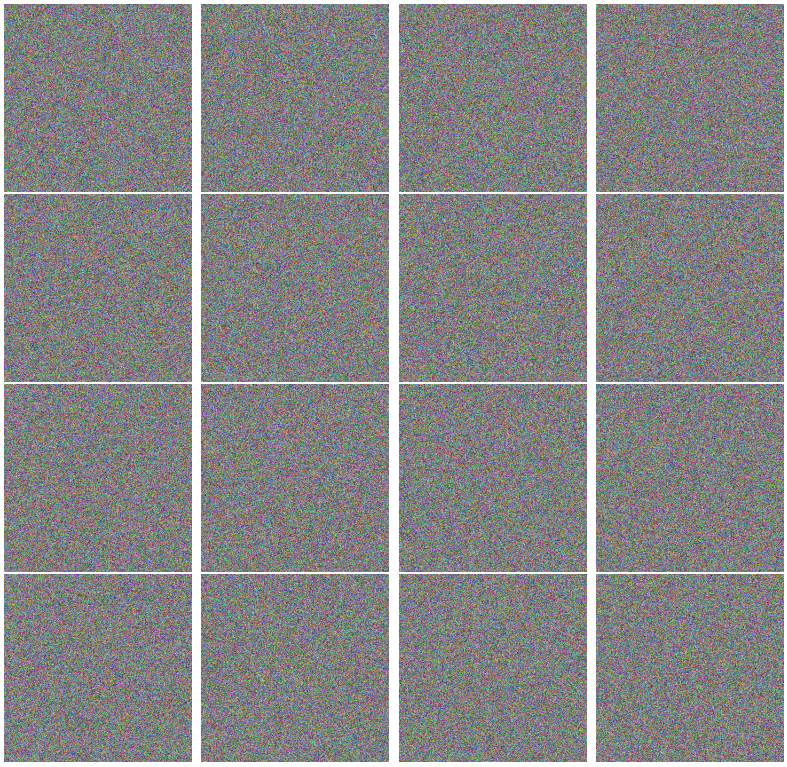


Figure 2.2: An intuitive display that real-world images are not uniformly distributed in image space. Each of the above images was randomly generated, and none show any semblance of structure that we expect to see in a real-world image. Real-world images almost always have some sort of structure or consistency to them (such as nearby pixels often having similar values).

normal distribution into an image data distribution. Formally,

$$p_{fake}(\boldsymbol{x}) = G(\mathcal{N}) \tag{2.1}$$

where G represents the generator function, $\boldsymbol{x}$ is a random variable representing an image, $\mathcal{N}$ is the normal distribution, and $p_{fake}(\boldsymbol{x})$ is the probability distribution of the images generated by the generator. The desired goal of the generator is to minimize the difference between the generated distribution and the true data distribution. One of the most common metrics to minimize this difference is the Kullback-Leibler (KL) divergence between the generator distribution and the true data distribution using maximum likelihood estimation. This is

done by finding the parameters of the generator, $\boldsymbol{\theta}$, which produce the smallest divergence,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}(p_{data}(\boldsymbol{x}) \parallel p_{fake}(\boldsymbol{x};\boldsymbol{\theta})). \tag{2.2}$$

To find this set of parameters, each of the discriminator and the generator works toward minimizing a loss function. With D being the discriminator's inference function, the discriminator's loss function is given by

$$L_D = -\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_{fake}}[\log(1 - D(\boldsymbol{x}))] \tag{2.3}$$

and the generator's loss function is given by

$$L_G = -\mathbb{E}_{\boldsymbol{x}\sim p_{fake}}[\log(D(\boldsymbol{x}))]. \tag{2.4}$$

For each network, the derivative of their respective losses are used to update the network parameters via the backpropagation algorithm. In the case of image data, this approach has led to generative models which can produce realistic looking images reliably [26].

However, this type of GAN does not improve the capabilities of the discriminator on any predictive task. Here, the discriminator is only used to train the generator. This can be understood further if we assume an ideal discriminator and generator network. In this case, the generator would eventually learn to produce data which exactly matches the data distribution of the real data, or, in the case of limited training data, produce images which exactly match those in the training dataset. In this case, the best prediction the discriminator can give is a 50% probability that the image is a real image.

2.2 Semi-Supervised GANs for Classification

In this section, a subset of GANs is explained which are used to improve the training of neural networks for discrimination and prediction tasks. Specifically, this type of GAN allows a predictive network to train with less labeled data using a semi-supervised approach. In this case, both a labeled and an unlabeled dataset are used. In addition to distinguishing between real and fake data, the discriminator also attempts to label real input data into one of the real world classifications. The primary goal of this type of GAN is to allow the discriminator's prediction task to be trained with relatively small amounts of labeled data using unlabeled data to provide the network with additional information. As unlabeled data is usually much easier to obtain than labeled data, this provides a powerful means to reduce the requirements of training neural networks. This semi-supervised GAN structure can be seen in Figure 2.3.

Where in a simple GAN the discriminator would be passed true examples and fake examples, in the semi-supervised GAN the discriminator is given true labeled examples, true unlabeled examples, and fake examples. We can better understand why this is useful by considering the case of image classification. In this case, the discriminator is being trained to predict the correct class of a true image, which can be one of the K classes that exist in the dataset. The discriminator is given the additional goal of attempting to label any fake images with a $K + 1$th class, which only exists to label fake data (i.e., does not exist in the true label dataset). For the case of unlabeled, all we know is that it must belong to one of the first K classes, as the $K + 1$th class does not exist in the real data. The discriminator is then punished for labeling true unlabeled data as the $K + 1$th class. This is useful because the discriminator cannot simply overfit to the labeled data, as it still has to accommodate for the unlabeled data. At the same time, the fake data prevents the discriminator from allowing simple features to be the deciding factor, as the generator is able to produce such simple

features. In doing so, the real/fake classification provides a form of regularization. However, unlike ordinary forms of regularization, this regularization is based on the distribution of the data.

To understand what is happening in this semi-supervised learning more intuitively, we can start with a non-GAN classification network. Such a network learns a transformation function which partitions image space into separate classes. The network can only learn from the labeled data and warps its partitions to best fit each of the labeled classes into the partition they belong to. At this point, adding unlabeled data to the training process provides no additional benefit. There is currently no constraint on the extent of the classification partitions (except at the boundary of another class), and so all unlabeled data will lie within one of the class partitions. As the new data is unlabeled, this provides no additional information. It is only with the additional cost of the fake class that the unlabeled data can be used with any value. Once the fake data narrows the range of the partitions, we can see them as manifolds once again. Now there are few labeled data points and many unlabeled data points, all of which must lie on the manifolds of the real classes. The manifolds have different regions (or even separate manifolds) for each class, but even the unlabeled data has to lie somewhere on the manifold(s). As the discriminator trains, it learns how to segment the data points into categories. To do this, it creates a mapping from a predictive manifold to a class, with the training warping the manifold to contain each of the data points for that class. At the same time, the generator prevents the manifold from warping too severely to reach data points in arbitrary ways. Intuitively, this is because severely warping the manifold to reach true data points can result in the manifold stretching into the area which does not represent true images. The generator acts a pressure on the manifold to reduce this. By generating images near the manifold, the generator forces the discriminator's manifold not to wander into areas that don't contain real images. In this sense, the generator is a form of regularization for the discriminator, but one which is based on real-world data. At the same

time, the unlabeled data forces the manifold to reach as much of the real data distribution as possible. In doing so, the discriminator must learn enough robust and complex features to reach all the real data while not learning arbitrary overfitting features that stretch the manifold into the fake regions.

As originally formulated by Salimans et al. [21], the semi-supervised GAN's discriminator loss function is then defined by

$$L_D = L_{supervised} + L_{unsupervised} \tag{2.5}$$

$$L_{supervised} = -\mathbb{E}_{\boldsymbol{x},y\sim p_{labeled}}\log[D(y \mid \boldsymbol{x}, y < K+1)] \tag{2.6}$$

$$\begin{aligned} L_{unsupervised} = & \\ & - \mathbb{E}_{\boldsymbol{x}\sim p_{unlabeled}}\log[1 - D(y = K+1 \mid \boldsymbol{x})] \\ & - \mathbb{E}_{\boldsymbol{x}\sim p_{fake}}\log[D(y = K+1 \mid \boldsymbol{x})]. \end{aligned} \tag{2.7}$$

As for the generator, the first option for a loss function is the straight forward one which aims to have the discriminator label the fake images as from real classes. Specifically,

$$L_G = -\mathbb{E}_{\boldsymbol{x}\sim p_{fake}}\log[D(y < K+1 \mid \boldsymbol{x})]. \tag{2.8}$$

However, Salimans et al. [21] achieved improved results by training the output activations of an intermediate layer of the discriminator have similar statistics in both the fake and real image cases. That is, the generator should attempt to make its images produce similar features in an intermediate layer (instead of final classification output) as is produced when true images are input. This can be intuitively understood as making the statistics of the

image be the same in both the fake and real cases, specifically, the feature statistics that are used in deciding a classification. The simplest and most useful statistic to try to match is the expected value for each feature. Formally put, if we denote $f(\boldsymbol{x})$ as the features output by an intermediate layer in the discriminator, then the loss function for the generator becomes

$$L_G = \left\| \mathbb{E}_{\boldsymbol{x} \sim p_{real}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x} \sim p_{fake}} f(\boldsymbol{x}) \right\|_2^2. \tag{2.9}$$

This benefits the process by preventing the generator from producing images which exactly match those from the real training dataset. If the generator learns to produce exactly images from the training data, the generator provides no new information not gained from the training data itself. Instead, the feature matching goal encourages the generator to produce images from the same image distribution, but not specifically the same images.

Since their development, semi-supervised GANs have been used to improve training in many areas of classification, including digit classification [27, 28, 21], object classification [27, 28, 21], facial attribute identification [28], and image segmentation (per pixel object classification) [29].

2.3 Alternative Semi-supervised Regression GAN Methods

2.3.1 Explicit label semi-supervised regression GANs

While we have defined why this is an important area to research, it has not yet been explained why this task would require any great effort. Why can we not simply take our solution for classification and apply it to a regression problem?

The first issue readily presents itself when examining the discriminator loss functions of a

classification semi-supervised GAN (Equations (2.5) to (2.7)). These functions call for the labels to belong to a specific class, whether it be one of the real classes or the generated fake class. Obviously, regression problems do not have classes. The discriminator desires the labeled examples to be given the correct class, the unlabeled examples to be given any existing real class, and the fake examples to be given a new fake class. For regression tasks, the labeled case is still straight forward: the discriminator is trained to minimize the error between its predictions and the true labels. The goal of the discriminator for the unlabeled examples and the fake examples is less clear. The first option would be to have the discriminator distinguish between real and fake data using the predicted regression value. Yet, which values to choose for what constitutes real data and what constitutes fake data is challenging.

In many applications, the possible real data values may be in the range $(-\infty, \infty)$, in which case, there is no option left for a fake label value. Even if we dismiss this limitation, other significant issues still arise. Most notably, using an explicit label range to constitute a fake label results in a prediction bias. For example, we can consider the case of crowd counting, where the goal is to predict the number of people in a given image. Here, the possible true values are in the range $[0, \infty)$. We might then choose to make a negative number constitute a fake label. In the case of an ideal generator, the generator would learn to produce exactly images from the real dataset. The ideal discriminator then has a dilemma. The same image may have come from the real data or the generator. In the former case, it should predict the correct count, while in the latter case, it should predict a negative number. In this situation, the discriminators best choice is to predict half the correct count of the real image. Since the overall goal of the semi-supervised GAN is to improve the predictive powers of the discriminator, this produces an undesired bias. This scenario is depicted in Figure 2.4. Of course, the generator will never perfectly produce duplicates of the real data, but a significant bias effect is still present in this model.

Although the bias is the primary issue with this model, there are yet more difficulties,

such as the need to manually select the explicit fake label range. In many cases, the exact range of the real labels is not known. Rezagholiradeh and Haidar [30] presents such an explicit label GAN, which is compared against the proposed SR-GAN method in the driving steering angle experiments.

2.3.2 Dual-goal GAN for regression

Another potential alternative to the proposed method is a dual goal GAN (DG-GAN) approach. A DG-GAN outputs two labels: a regression value prediction and a fake/real classification prediction. The idea is that the network must learn both how to distinguish between real and fake examples, and how to predict the correct value for a labeled example. However, this approach also has several potential flaws. First, the method does not enforce that these two prediction goals be related. Part of the network may learn the task of identifying real/fake images, while another portion of the network learns the task of predicting regression values. A representation of this split learning is shown in Figure 2.5. If the objective of distinguishing being real and fake examples is weighted strongly enough, the network may devote larger portions of the network to the real/fake classification task, thereby reducing its effectiveness in the regression prediction.

Additionally, this approach presents a bias similar to the one encountered using the explicit label model. Specifically, two linear transformations are used from the final feature vector to produce the regression value and the real/fake classification. Although the two predictions undergo different non-linear activation functions (a linear activation for the regression value and a sigmoid activation for the real/fake classification), the values entering these activation functions are linearly dependent. The bias here is mitigated by the non-linear activation, but each feature is linked to both output values with a static weight (after training). Once again, certain regression values are considered more real than others, leading to a bias in the predicted values.

2.3.3 Binning regression into a classification task

Finally, it is possible to simply reformulate any regression task into a classification task by binning the regression values into discrete bins. In practice, this presents several significant problems. For example, to recast a regression task as a classification task, an arbitrary number of classes must be chosen, as a continuous range can be segmented into discrete bins in an arbitrary number of ways. This discretizing of the range imposes a limitation on the precision which can be achieved; Using the center value of the bin as the predicted value, the average error for any value within the bin is $\frac{1}{4}$ the size of the bin. As the bins become larger, the greater the precision limitation of the bin.

More significantly, the relative locality of the regression values is discarded when converting to a binning classification problem. The usual softmax loss for multiclass tasks results in each incorrect class bin being considered equally as wrong. For example, we can consider a prediction of a real number in the range $(0, 100)$ which has been split into 4 discrete classes. Given a true label of 95, a prediction of the $(50, 75)$ bin would produce the same loss as a prediction of $(0, 25)$. Firstly, this results in a problematic training gradient, as any nearby bins are considered entirely wrong, and moving from a far bin to a close, but wrong, bin does not produce a lower loss. Without this training information, the network may stall in improvements. Furthermore, it encourages networks which overfit, since only the single bin improves the loss, and a solution which moves many values closer to their correct bin will be ignored for one that increases the weight of an already near correct prediction. These problems are depicted in Figure 2.6. The smaller the bins become, the more problematic this limitation becomes. Loss functions which take into account locality of bins are significantly more inefficient and impractical for use in neural network training.

2.4 Obliquely Related Methods

Zero/one/few-shot learning methods [31] and other meta/transfer learning methods [32] have been rapidly developing in the last few years. These methods seek a similar end goal as the semi-supervised GAN approach: provide a network which functions well on a task with little labeled data for the task. However, the method by which they approach the problem is significantly different. Few-shot methods focus on learning a separate task from the final goal, then work to use those learned transformations on for the new task. The semi-supervised GAN approaches only includes data from the desired task but takes advantage of learning from unlabeled data.

These two approaches are complementary rather than competing approaches. Zhang et al. [33] uses a semi-supervised GAN on top of a few-shot method to demonstrate further improvements. In particular, Zhang et al. [33] shows that this approach generalizes to most few-shot models. Beyond the complementary nature, few-shot approaches are even newer than semi-supervised GANs, are (currently) typically applied to simpler applications than semi-supervised GANs, though this may change in the near future.

Similar to the case of semi-supervised GANs, these few-shot approaches are almost exclusively applied to classification rather than regression tasks. Given all these factors, few-shot learning and meta-learning approaches are not relevant enough to the method proposed here to be included in this work.

Another distinct category of related work is that of regression in conditional GANs. Conditional GANs are a type of GAN designed to produce realistic examples which have specific desired properties in the example. Bazrafkan and Corcoran [34] provides an approach to generate images with specific characteristics in a conditional GAN. In particular, they use a regressor in parallel with the discriminator network to provide more variation in the generated examples. These works are attempting to produce realistic-looking generated

examples. The product is not a predictive network for real examples.

In contrast, our approach is designed to improve the predictive capabilities of the discriminator on real examples. Notably, *we do not expect our generator to produce realistic-looking examples*. On the contrary, we expect the examples generated will not look realistic. As noted by Salimans et al. [21], the use of feature matching (which is also used in our work) improves the discriminator's predictive accuracy while reducing the realism of the generated examples. We expect our feature contrasting approach further erodes the realism. Furthermore, works such as Dai et al. [35] show how a generator which produces examples that are too realistic may be less advantageous for improving a discriminator's predictive abilities.

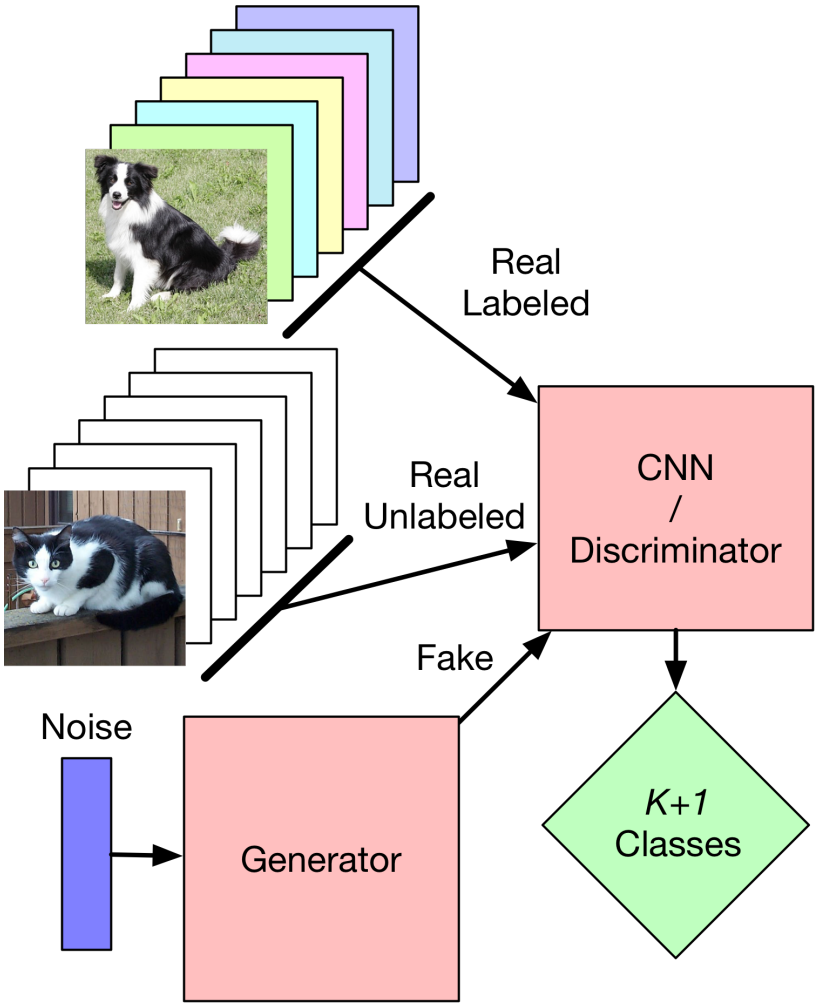


Figure 2.3: The structure of a semi-supervised GAN. Both labeled and unlabeled real images, as well as fake images, are fed to a discriminator network, which tries to determine which class each image belongs to (K real classes and one fake class). The discriminator wishes to label images from the generator as belonging to a special "fake" class.

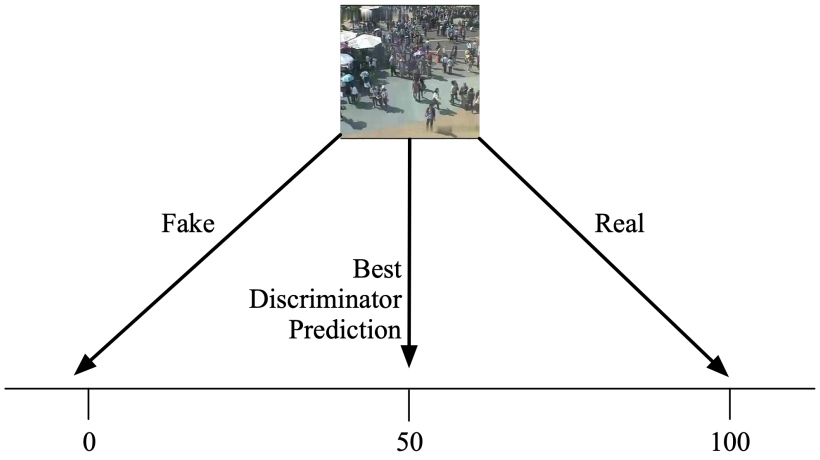


Figure 2.4: A prediction bias is introduced when using an explicit fake label range for regression values. In crowd counting, true values are in the range $[0, \infty)$, so negative values may represent the fake label. However, if a generator can produce images which exactly match the real image distribution, the best prediction the discriminator can pick is half the value. Here, an image containing 100 people could be either from the real data or the hypothetical perfectly trained generator. In this case, the discriminator's best prediction is 50. This results in an extreme prediction bias, and does not improve the predictive capabilities of the discriminator for crowd counting.

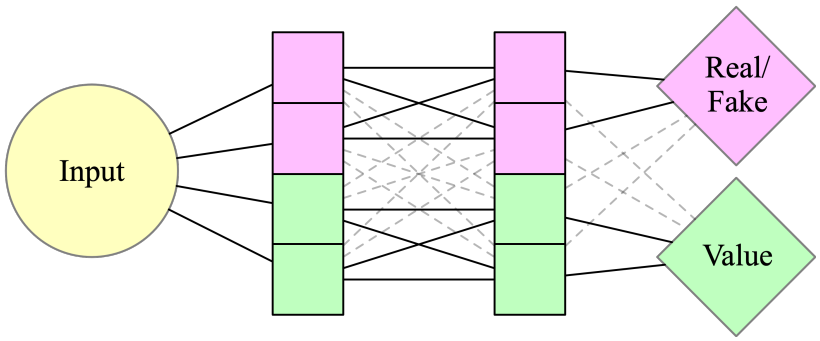


Figure 2.5: A DG-GAN network splitting the network into solving the two objectives independently, rather than using a shared representation. The dashed lines represent connections which exist but have very low weights. The degree of this division of learning can vary.

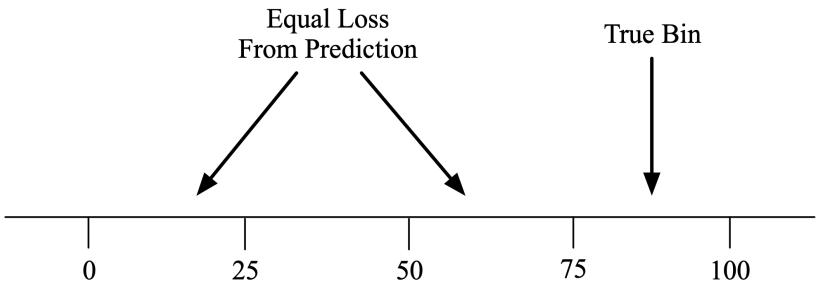


Figure 2.6: Reframing a regression problem as a classification task by binning regression values has two major limitations. First, the bin size limits the precision of the prediction. This problem becomes more significant as the larger the bins become. Second, applicable loss functions do not consider the locality of the bins. This results in several significant training limitations. This problem becomes more severe as the smaller the bins become.

Chapter 3

Motivation of SR-GANs

3.1 Benefits of SR-GANs

The main advantage of the SR-GAN is the reduction of data for training a network. However, this manifests itself in three important ways: 1) Networks can be trained in cases where not much data is available, or large amounts of effort are required to annotate data. 2) Training can be easily generalized to new situations, as little or no new annotated data is needed from that specific situation to train the network to it. 3) Models for existing datasets can be trained to higher accuracies with the SR-GAN than would be possible with the traditional network alone.

3.1.1 Training with minimal data

In the case of the MNIST handwritten digits dataset [36], Salimans et al. [21] found that 99% accuracy could be achieved with only 100 training images using semi-supervised GANs, wherewith a standard CNN it would take around 10,000 to achieve the same level of accuracy. This reduction of data requirements is one of the primary reasons we wish to expand on this

work.

Most current deep learning implementations are limited only by a lack of data or lack of time or computing resources to train the networks. That is, for most well-defined problems, if we were provided a virtually limitless supply of data and time, the existing neural network formulations could solve the problem. However, in most cases data is limited, often severely so. The SR-GAN works to expand these capabilities to regression tasks. Expanding the range of problems a semi-supervised GAN can apply to allows a way to solve the issue of limited data in more applications, and this is the primary benefit of the SR-GAN.

3.1.2 Generalizability of SR-GANs to new situations

Another reason semi-supervised GANs are compelling is one that arises implicitly from needing less data. Specifically, it presents networks which are able to generate new examples and compares them against its own predictions. That is, being able to abstract new instances from the idea of an example rather than merely trying to learn a simple representation to fit every example in the training dataset. This process of generating new versions of the thing being learned, and comparing them against real examples to look for inconsistencies forces the GAN to learn more robust and complete representations. Although perhaps speculative, this may be leading neural networks closer toward how human brains are able to learn ideas with fewer examples.

3.1.3 Increasing accuracy with existing dataset quantities

Ideally, for training deep neural networks, the network would have access to an unlimited supply of annotated data. Of course, producing such data is impractical or impossible. Instead, if a network cannot be designed which can reach the level of accuracy the developer requires, then more data is typically annotated. Once the required accuracy is reached, often,

there is not much incentive for the developer to annotate more data. However, if more data would allow the network (or a more powerful network) to perform the task even better than we expect the SR-GAN would help improve the results with the current level of data.

On the other hand, we *expect* a limitation to the benefit of the SR-GAN. In the best case, as the quantity of annotated data approaches infinity, we should expect that the SR-GAN would provide no benefit, as labeled data can always provide more information than unlabeled data used in a clever way. Furthermore, we can expect that, if imperfectly designed, the SR-GAN would actually be a determent to accuracy when the amount of annotated data approaches infinity. This is because the SR-GAN encourages the network to minimize a loss which is not directly being trained for. If there is unlimited annotated data, training only for the primary task should yield the best possible results, and the SR-GAN introducing a new goal can at best do nothing, but may also encourage the network to move away from the ideal network. If there is no chance the quantity of data available would lead to this, then this isn't an issue. However, if we want to make the SR-GAN generalized to work in every case, even cases where extremely large quantities of the labeled data is available, this issue needs to be taken into account. Luckily, the real-world experiments presented in this work did not encounter this issue. This suggests that the detriment of using the SR-GAN in problems with large amounts of labeled data is negligible.

3.1.4 Improved understanding of GANs

While the above generally explains why it is interesting to investigate GANs and semi-supervised GANs in more detail, there are also specific reasons for semi-supervised GANs for regression problems.

Firstly, there are countless problems which require the use of a regression method to be solved. The crowd analysis case described in this work is one such example. Beyond the methods described in Section 2.3, the solutions for classification problems currently handled

by semi-supervised GANs only work for classification. In Chapter 4, it is shown how the SR-GAN avoids the problems and limitations of the methods from Section 2.3.

More importantly, semi-supervised GANs for regression are interesting because regression is more general than classification. Specifically, classification is a subset of regression. In particular, any classification problem can be framed as a regression problem. The typical formulation of a neural network is to output a continuous number as a "score" for each class, and simply choose the highest score to be the predicted class. The SR-GAN methods can be applied directly to this output, or the following loss function (typically from a softmax) without any issues. This is not true of converting from regression to classification, as described in Section 2.3.3. Classification can be considered a subset of regression problems, and any of the solutions used for general regression can be applied to classification. As such, this work helps expand the overall application and understanding of semi-supervised GANs.

3.2 Problems and Proposed Solutions

While we have defined why this is an important area to research, a complete reasoning of why this is a challenging task needs to be expounded.

3.2.1 Reformulating GANs for regression

The first major issues have already described in the limitations of alternative semi-supervised regression GAN models in Section 2.3. Namely, how to formulate the mathematics of the semi-supervised regression GAN so as not to introduce biases or other training issues. Converting the problem to a binned classification problem leads to both precision and limited training issues. An explicit fake label method leads to significant prediction bias. The DG-GAN method leads to split goal network training and additional biases. So the first issue to solve is in developing a mathematical formulation of the SR-GAN which does not introduce these

problems or others. Such a formulation is provided in Chapter 4.

3.2.2 Training stabilization

A major challenge in training any type of GAN (not exclusively an SR-GAN), is building the network cost that reliably converges [37]. Particularly in the case of the proposed SR-GAN method (see Chapter 4), the possible choices of loss functions present several ways where the training between the discriminator and the generator could oscillate or otherwise fail to converge.

Several methods have been suggested as to how to stabilize GAN training for the case of classification. Throughout this work, we use a gradient penalty method [37] to dampen and converge generator/discriminator oscillation. However, this dampening is only applicable if the model is only failing to converge due to training oscillation. If the semi-supervised loss functions are inherently unstable, no amount of dampening would resolve the problem. While there are countless choices of unstable loss function combinations, choosing a stable set is relatively simple once the details are understood. Furthermore, the experiments in Chapter 5 demonstrate that the choice of loss functions is relatively unimportant to the resulting accuracy, so long as a stable set of loss functions is chosen.

3.2.3 Developing generally applicable hyperparameters

While the accuracy of neural networks is arguably the most crucial reason they are so widely used today, another highly important reason is their generalizability. Within computer vision, well-known networks such as AlexNet [22] and VGGNet [38] have been used for countless specific applications with little or no change to the existing network structure. Much of the success of neural networks stems from their ability to require little change to function in a new setting. Earlier networks often had difficulty converging or would take too long to train

to be useful. Advances, such as the replacement of stochastic gradient descent optimizers with more complex optimizers like the Adam optimizer [39], have allowed neural networks to provide a working solution with far less application-specific hyperparameters.

In a similar fashion, the SR-GAN requiring many handcrafted hyperparameters for a single application is not desirable. We wish to obtain a model which will function over a wide range of scenarios with little or no change to the main SR-GAN algorithm or hyperparameters. In particular, how to weigh the relative values of the various supervised and unsupervised optimizations initially seems to be a daunting task. However, in Chapter 5, it is demonstrated that the choice of weighting the various components is quite forgiving, so long as a few generalizable guidelines are followed.

Chapter 4

SR-GAN Methodology

4.1 SR-GAN Formulation Using Feature Contrasting

The semi-supervised regression GAN (SR-GAN) approaches regression estimation by comparing the types of available data (labeled, unlabeled, and fake) as probability distributions rather than individual examples. In this method, the discriminator does not attempt to predict a label for the unlabeled data or fake data. Instead, the statistics of the features within the network for each type of data are compared. Here is the fundamental idea: We have the discriminator seek to make the unlabeled examples have a similar feature distribution as the labeled examples. The discriminator also works to have fake examples have a feature distribution as different from the labeled examples distribution as possible. This forces the discriminator to see both the labeled and unlabeled examples as coming from the same distribution, and fake data as coming from a different distribution. In this way, it learns to distinguish between the different distributions of the examples rather than distinguish individual examples as real or fake. The generator, on the other hand, will be trained to produce examples which match the unlabeled example distribution, and because of this, the

generator and discriminator have opposing goals. How a label is assigned to an example drawn from that distribution is still decided based on the labeled examples (as it is in ordinary DNN/CNN training), but the fact that the unlabeled examples must lie on the true example distribution manifold forces the discriminator to more closely conform to the true underlying data generating distribution. At the same time, avoiding the fake distribution prevents the discriminator from stretching the prediction manifold away from the true data distribution.

4.1.1 SR-GAN optimization

The SR-GAN structure is shown in Figure 4.1 with the age estimation application depicted as an example. The method for matching the features of two types of data was initially proposed in Salimans et al. [21]. Salimans et al. [21] used the method to train the generator to avoid generator mode collapse. We also use the feature matching to train the generator to match the fake and unlabeled data. However, we also extend the feature matching to the case of training the discriminator to match the two real data distributions (labeled and unlabeled). In the case of training the discriminator to distinguish between the unlabeled distribution and fake distribution, we propose a novel approach, *feature contrasting*, which is antithetical to feature matching. In this case, the discriminator attempts to make the features of the real and fake data as dissimilar as possible, while the generator is attempting to make these features as similar as possible.

Specifically, the loss functions as defined for the semi-supervised GAN for classification (Equations (2.5) to (2.7)) are entirely reformulated in the case of regression. First, we separate the loss of the discriminator into several terms for clarity. This is given by

$$\begin{aligned} L_D &= L_{supervised} + L_{unsupervised} \\ &= L_{labeled} + L_{matching} + L_{contrasting} \end{aligned} . \tag{4.1}$$

What we refer to as the "labeled loss", is given by

$$L_{labeled} = \mathbb{E}_{\boldsymbol{x},y\sim p_{data}(\boldsymbol{x},y)}[(D(\boldsymbol{x}) - y)^2]. \tag{4.2}$$

This loss is similar to an ordinary fully supervised loss (for regression training). Next, the "matching loss" causes the discriminator to attempt to make the feature statistics of the real labeled data and the real unlabeled data be as similar as possible. This matching loss is given by

$$L_{matching} = \|\mathbb{E}_{\boldsymbol{x}\sim p_{labeled}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim p_{unlabeled}} f(\boldsymbol{x})\|_2^2. \tag{4.3}$$

In contrast, the "contrasting loss" causes to the discriminator to attempt to make the feature statistics of the real data as dissimilar to the fake data as possible. With $f(\boldsymbol{x})$ being a feature vector within the network given input $\boldsymbol{x}$, this feature contrasting is accomplished with the loss function given by

$$L_{contrasting} = -\left\|\log\left(|\mathbb{E}_{\boldsymbol{x}\sim p_{fake}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim p_{unlabeled}} f(\boldsymbol{x})| + 1\right)\right\|_1. \tag{4.4}$$

Finally, the generator attempts to make the feature statistics of the real data match those of the fake data. This goal is accomplished by the generator loss given by

$$L_G = \left\|\mathbb{E}_{\boldsymbol{x}\sim p_{fake}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim p_{unlabeled}} f(\boldsymbol{x})\right\|_2^2. \tag{4.5}$$

Here, $L_{matching}$ and L_G are identical except in which data distributions are being compared. Additionally, the feature contrasting in Equation (4.4) is in direct opposition to the feature matching in Equation (4.5). Notably, there is no possibility for the generator and discriminator to both benefit by a change in one of the features; A decreased loss for one necessarily results an increased loss for the other (for the change in an individual feature difference). A comparison

of a change in the loss from a single feature can be seen in Figure 4.2. The specific details of the functions used to measure the distance between the two distributions can vary, and the choice of these loss functions are discussed in more detail in Chapter 5.

Notably, the matching and contrasting functions for the two distributions need to provide conflicting goals for any given feature difference to prevent the discriminator and generator from converging. Additionally, it is important to note that the derivative of the loss function is the value which the neural network is actually trained on. The contrasting function cannot simply be the negative of the matching function. Where the matching function has little slope near zero, the contrasting function should have the largest slope near zero. Intuitively, the largest desire to create an increased contrast occurs when the distributions to be contrasted are exactly matching. For matching, the further the two distributions are apart, the greater the need to make them match. If the contrasting slope were to increase (rather than decrease) further from zero, there is the potential for a runaway training gradient.

To summarize, the SR-GAN uses feature matching for the discriminator loss functions where in previous methods a separate "fake" class is defined. Specifically this can be seen in the change from the unsupervised loss in Equation (2.7) (which uses a "fake" class in the discriminator) to Equations (4.3) and (4.4) (which uses feature layer statistics). This accomplishes several goals:

1. Regression problems have no classes, and the previous methods require a "fake" class definition, and the SR-GAN approach allows regression problems to be approached.

2. The feature statistics comparison does not introduce any bias in the discriminator label prediction, as the final label output is not used in the unsupervised loss.

3. The feature statistics comparison uses every feature of a layer in its comparison.

4. The approach requires no prior information about the data and requires no manual definition of goals beyond the original loss function for labeled examples.

Notably, the above points solve the issues with the alternative models in Section 2.3. Firstly, no arbitrary binning is required, with the training goal is the original regression goal.

Next, the final output label is not used as part of the unsupervised loss. In particular, a change in the labeled, unlabeled, and fake distributions does not require a change in the output regression values; statistically different feature vectors can produce the same regression value, resulting in optimization option which does not introduce a bias in the regression prediction.

Where the DG-GAN can split the model into achieving two separate goals, the SR-GAN cannot. Unlike in the DG-GAN, every value in the SR-GAN's feature vector is included in distinguishing between real and fake distributions; since there is no benefit to splitting the model, all values are used in the regression prediction. This is depicted in Figure 4.3.

Finally, both the explicit fake label GAN and the binning classification GAN require manually setting hyperparameters, requiring expert domain knowledge. The SR-GAN requires no such information and automatically distinguishes between distributions.

4.1.2 Gradient penalty

An additional challenge preventing the use of an SR-GAN is the difficulty of designing an objective which reliably and consistently converges. GANs can easily fail to converge under various circumstances Barnett [40]. To solve these general GAN instability issues, we use the gradient penalty approach proposed by Arjovsky, Chintala, and Bottou [41] and Gulrajani et al. [37].

The gradient penalty as defined by Gulrajani et al. [37] is not applicable to our situation, because their gradient penalty is based on the final output of the discriminator. As the final output of the discriminator is not used in producing the gradient to the generator, we use a modified form of the gradient penalty. This gradient penalty term is added to the rest of the

loss function resulting in

$$\begin{aligned} L &= L_{labeled} + L_{matching} + L_{contrasting} \\ &\quad + \lambda \, \mathbb{E}_{\boldsymbol{x} \sim p_{interpolate}} \left[\max \left(\left(\|\nabla_{\hat{\boldsymbol{x}}}(f(\boldsymbol{x}))\|_2^2 - 1 \right), 0 \right) \right]. \end{aligned} \tag{4.6}$$

where $p_{interpolate}$ examples are generated by $\alpha p_{unlabeled} + (1 - \alpha) p_{fake}$ for $\alpha \sim \mathcal{U}$, and λ being a manually chosen scalar. The last term provides a restriction on how quickly the discriminator can change relative to the generator's output. Our version of the gradient penalty term is modified in multiple ways from the original. First, as noted above, the final discriminator output cannot be used, nor should it, as the discriminator's interpretation of the generated data only matter in regard to the feature vector, $f(\boldsymbol{x})$. Second, the gradient penalty is normally applied to a term similar to the $L_{contrasting}$ term using the interpolated values, as this value is related to how the discriminator views the generator's data. However, our $L_{contrasting}$ is based on the average of a batch of fake examples whose difference is then taken from a batch of real examples. As both the $L_{contrasting}$ term and interpolates are calculated based on the real data, the resulting gradient penalty is negligible. Instead, we apply the gradient penalty directly to the mean feature vector of a batch of interpolated examples and do not apply the feature distance loss function compared to the mean real feature vector. As this penalizes the gradient even for mean feature vectors far from the mean real feature vector, it may slow training. However, near the real feature vector, it approximates the original gradient penalty formulation and works well in practice. Lastly, we use the one-sided version of the gradient penalty described by Gulrajani et al. [37]. As mentioned in their work, the one-sided penalty more closely matches the desired discriminator training properties, and we found this approach to produce higher accuracies than the two-sided penalty.

4.2 SR-GAN Experiments

To demonstrate the capabilities of the semi-supervised regression GANs, we use two experimental regimes, each of which consists of several individual trials and demonstrations.

The first experimental regime is of a synthesized dataset problem. This allows us to demonstrate the details of the theory behind a semi-supervised regression GAN in a well-controlled and understood environment. The dataset consists of values taken from randomly generate polynomials. The goal of the network is to predict the original polynomial coefficients based on the sampled data. This data generating distribution is explained in detail in Chapter 5. Using this simple problem, we can show how the semi-supervised regression GAN works in detail, what variations can influence its capabilities, and what its limitations are.

The second experimental regime consists of real-world applications with specific use cases. Three applications, driving steering angle prediction, age estimation and crowd analysis, have been chosen for this purpose. The simplest variant of the crowd analysis case is taking a single image and estimate how many people are in the image. Similarly, the steering angle and age estimation cases result in a single predicted value and are all determined from single images. However, there are more complicated areas we wish to deal with in regard to the crowd counting case, specific, per pixel density labels, and analysis of grouping within areas of the image. Of particular value, having the real-world applications gives meaning to the differences in accuracy between the SR-GAN method, the base DNN method, and the alternative GANs we are examining, and allows for comparisons in the future. Finally, the real-world case provides an area we can show direct improvements in compared to the state-of-the-art of that field.

4.2.1 Loss function and stability analysis on synthetic data

Of the challenges preventing the use of an SR-GAN, the greatest is undoubtedly the difficulty of designing an objective which reliably and consistently converges. GANs can easily come to a point where they fail to converge. To converge, the GAN must reach a Nash equilibrium. Unfortunately, optimization loops can occur in many places. For example, if we consider the simple cost functions for the discriminator and the generator as

$$L_D = xy \qquad \text{and} \qquad L_G = -xy \tag{4.7}$$

with the discriminator training x and the generator training y, we immediately see a cyclic pattern for any starting position besides $(0, 0)$, which is the desired Nash equilibrium point. It is important to note in this example, simply lowering the learning rate would not result in a converging system. In the high dimensional case of real-world applications, many optimization loops of this sort can occur.

We propose two primary methods to produce this stable training. The first is a selection of loss functions that results in stable training. Specifically, the discriminator's loss function can be built in such a way that, if we assume a perfectly trained generator, the discriminator should still approach the optimal solution. The second solution is using optimization penalties designed for specific problems. This includes weight clipping [21] and gradient penalties [37].

Using this, we can consider the case of the SR-GAN model. Here, the discriminator is trying to minimize the feature distance between the unlabeled and the labeled data, which is defined by the loss given by

$$L_{matching} = \|\mathbb{E}_{\boldsymbol{x}\sim p_{labeled}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim p_{unlabeled}} f(\boldsymbol{x})\|_2. \tag{4.8}$$

At the same the time discriminator is trying to maximize the feature distance between the

fake labels and the unlabeled data, which is defined by the loss given by

$$L_{contrasting} = -\left\|\mathbb{E}_{\boldsymbol{x}\sim p_{fake}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim p_{unlabeled}} f(\boldsymbol{x})\right\|_2. \tag{4.9}$$

Using this setup, if the generator is able to exactly mimic the unlabeled data features, then the discriminator will no longer be able to use the unlabeled data usefully, as the gradients from these two losses would cancel out. Worse still, if $L_{matching}$ produces a small gradient on a weight (perhaps most unlabeled examples show the weight should increase, while a few suggest it should decrease) while $L_{contrasting}$ produces a large gradient on a weight (perhaps all fake examples suggest that the weight should decrease), then the training pressure from the contrasting loss will override the matching loss. There are several ways to remedy this, but the most straight forward is scaling $L_{matching}$ by some factor.

A related issue is that of an infinite payoff to the discriminator for avoiding the generator's examples. As the training is iterative, if the payoff for changing a weight to decrease $L_{contrasting}$ during a single training step is much greater than that to decrease $L_{matching}$ (and perhaps $L_{labeled}$), then the discriminator may altogether avoid solving the main problem in favor of foiling the generator. A straight forward approach to solving this issue is to use higher-order losses on the $L_{matching}$ than on $L_{contrasting}$. This prevents the discriminator from straying too far from solving the primary problem, but results in a loss which becomes very small near an optimal solution for $L_{matching}$ (and this then, in turn, exacerbates the first problem mentioned).

Although solving the two problems above is paramount to a working SR-GAN, neither addresses the more general GAN instability issues explained with the simple Nash equilibrium described at the start of this section. That problem results from the possibility of improvement cycles between the generator and the discriminator. To solve these more general GAN instability issues, we use the approaches given by Salimans et al. [21] and Gulrajani et al.

[37]. However, as the SR-GAN feature matching and contrasting method results in many extra ways in which improvement cycles can manifest themselves, extra effort must be made to prevent these. In Section 5.2, we examine various potential loss functions, finding that the SR-GAN is intrinsically robust, and converges to similar accuracies regardless of the loss functions chosen (so long as they are inherently stable loss functions).

4.2.2 Hyperparameter generalization on synthetic data

The motivation for general hyperparameters has already been explained in Section 3.2.3. Here, we describe the approach for designing such generalized hyperparameters.

The first and most naive solution would be to develop a working SR-GAN for a simple application and trying to apply it to a new application with the same hyperparameters. Unfortunately, this typically leads to a choice of hyperparameters which work well for the individual problem but does not generalize well.

The second option is defining an SR-GAN that works over a wide range of situations. In large part, this ties back into the stability discussed in Section 4.2.1. An SR-GAN model which converges stably and reliably is more likely to work when applied to a new application. Of course, a model which results in a reliable stability does not guarantee high accuracy.

In particular, the primary hyperparameters required by the SR-GAN are those which dictate the relative weights of the various loss functions. Luckily, the results in Section 5.2 demonstrate that the SR-GAN is quite robust to changes in these hyperparameters. Though a few general guidelines about these weights must be followed, the exact specifications can widely vary, with the resulting model obtaining similar levels of accuracy.

4.2.3 Minimal data training on age estimation, steering angle prediction and crowd datasets

This next set of experiments show the primary purpose of the SR-GAN on three real-world applications: steering angle prediction, age estimation, and crowd analysis. Here we test how little data is needed to achieve different levels of prediction accuracy for age estimation, steering angle prediction, and crowd density prediction. Within this set of experiments, two types of comparisons are made.

First, for the age estimation dataset, experiments were performed using the same amount of data using a standard DNN/CNN as compared to the SR-GAN. The amount of labeled data used vary from few training examples up to near the entire available training dataset. Many intermediate values are considered using logarithmically increasing quantities of labeled data (e.g., 1, 3, 10, 30, etc. examples). This set of experiments gives a sense of how much of an accuracy improvement the SR-GAN over DNN/CNN provides for a given amount of labeled data. These experiments are found in Chapter 6.

For automotive driving steering angle prediction, the SR-GAN model is compared against both a CNN model and the DG-GAN model. In particular, Rezagholiradeh and Haidar [30] designed two versions of the DG-GAN. While their DG-GAN model is limited only to the case of steering angle prediction, it provides a useful comparison to one of the few existing methods for GANs on regression tasks. Various quantities of labeled data training data are tested with each method. This is done in Chapter 7.

The third set of experiments focuses on the complex problem of dense crowd counting, where we use state-of-the-art models in the field as the discriminator network for the SR-GAN. Here we examine the relative amounts of data required to achieve various levels of accuracy on this open problem. This work can be found in Chapter 8.

4.2.4 Crowd labeling: additional means for improving the performance

For the dense crowd counting application, additional developments were made in improving the labeling used by the field. The SR-GAN allows for learning from unlabeled data by using the limited labeled data statistics; however, its development also revealed flaws in the existing labeling schemes used for dense crowd counting. The development of a new label topology and upsampling technique have been demonstrated to improve crowd counting results. In particular, the existing accepted labeling topology is a Gaussian-based density map (see Section 9.2). This work shows that an inverse kNN scheme produces better results, and why this labeling topology provides the network with more effective training gradients. The upsampling technique presented here allows full use of this topology by providing the use of full-resolution labels. Both of these methods should be applicable to other multitarget regression tasks; however, this is not demonstrated in this work.

This labeling topology was designed, in part, to improve the capabilities of the SR-GAN in the crowd counting application. In particular, with the SR-GAN's ability to learn from unlabeled data being dependent on the statistics of the labeled data, the most significant benefits from the SR-GAN come from using the best available labels, in the case of dense crowd counting, the inverse kNN map. However, additional training issues arose in the case of the SR-GAN for multi-goal dense crowd counting. Some more details are discussed in Section 9.5, but this issue is still open and should be pursued in future research.

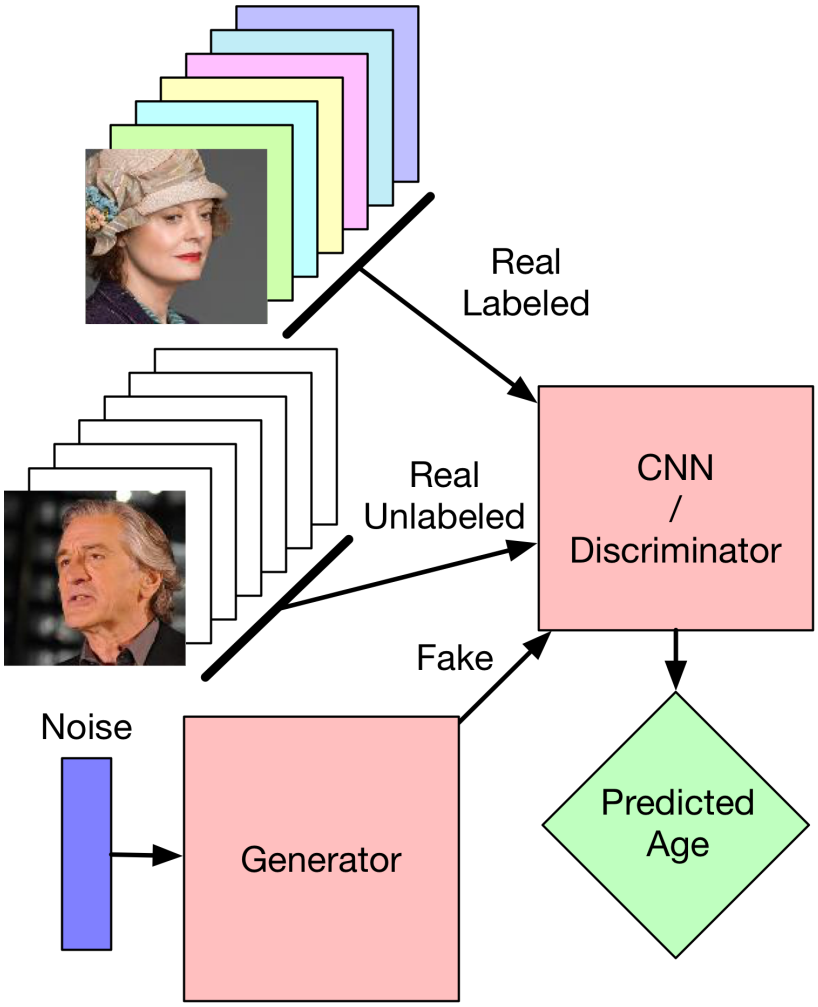


Figure 4.1: The structure of an SR-GAN. Its structure is similar to the semi-supervised GAN, with the major differences being in the objective functions and the output being a regression value. In this network, the discriminator distinguishes between fake and real images through feature statistics. No explicit real or fake label is assigned.

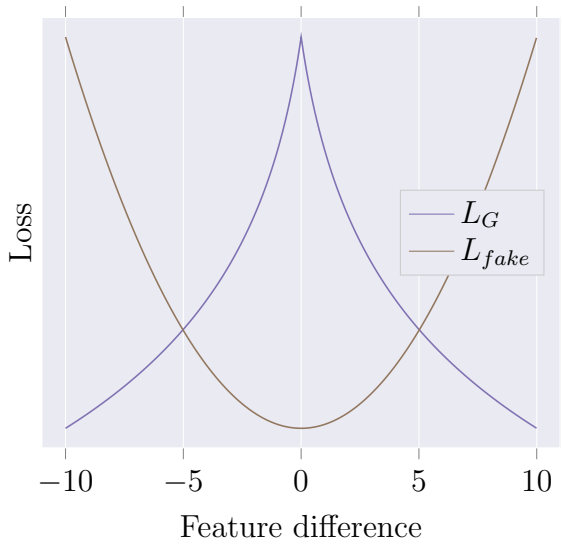


Figure 4.2: A comparison of the losses used for feature matching and feature contrasting, used in L_G and $L_{matching}$ respectively. The losses have been normalized for comparison. Shown in the change in loss due to a single feature (due to the norms used in the functions, multiple features changing together have a slightly different impact). Of particular note, a decreased loss for one necessarily results in an increased loss for the other.

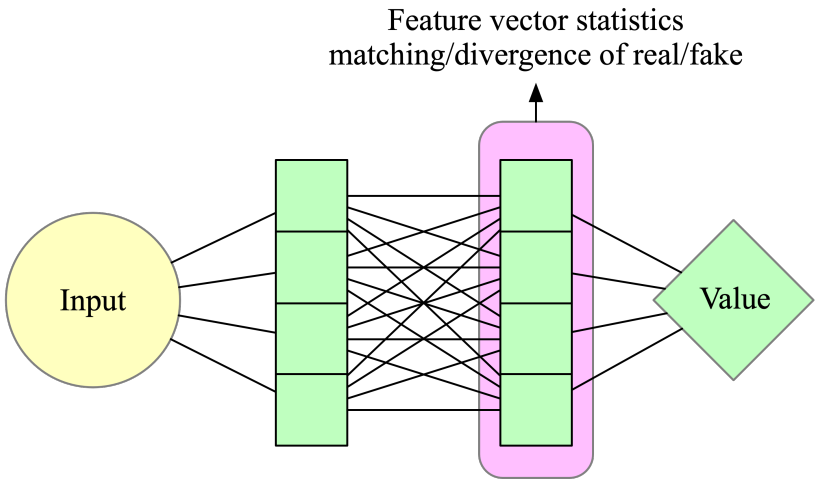


Figure 4.3: A SR-GAN network forces all features to be used in the distribution comparison, preventing the network splitting which can occur in the DG-GAN model (see Figure 2.5).

Chapter 5

Understanding the Behaviors of SR-GANs Through Polynomial Coefficient Prediction

The first experimental setup consists of a simple, well-controlled mathematical model, whose problem can be easily solved with simple neural networks when given enough examples. The example chosen is a polynomial coefficient estimation problem. This problem allows for an environment in which many properties of the semi-supervised regression GAN can be shown and its limits tested. In particular, the simple environment allows us to not only demonstrate the properties of the semi-supervised regression GAN but also give a clear theoretic understanding of why the network exhibits these behaviors. Four important aspects are discussed: 1) the dataset; 2) the experiment setup; 3) estimation with minimal data; 4) loss function analysis.

5.1 Polynomial Coefficient Estimation Overview

5.1.1 Polynomial coefficient estimation dataset

For the data of the synthesized application to appropriately represent the characteristics of a real regression application, we seek to create a data generating model that exhibits the following properties.

1. Able to produce any desired number of examples.
2. The distribution of the underlying data properties is selectable.
3. The relation between the raw data and the label is abstract, where the label is a regression value instead of one of a finite number of classes.
4. Able to contain latent properties that affect the relationship between the data and the labels.
5. Most of the data can be made to be irrelevant to the label.

Property 1 allows us to run any number of trials on new data, and run trials where data is unlimited. Property 2 reveals the inner workings of the data distribution. This is important, as we can monitor how closely the generator's examples match the real distribution and examine what kinds of distributions lead to limitations or advantages of the GAN model. Property 3 ensures the findings on the toy model are relevant to real deep learning applications for regression. That is, deep learning is typically used in cases where input data is complex, and an abstract, high-level meaning of that data is desired. When the relationship between the data and the label (the regression value) is too simple, more traditional prediction methods tend to be used. Property 4 is also important because of its relationship to real applications. Most applications involve cases where a property which is not the value to be predicted

directly affects the data related to value to be predicted. For example, in the case of age estimation, whether the image of the face is lit from the front or lit from the side drastically changes the data and what the CNN should be searching for, even though it has no direct impact on the age of the individual in the light. Finally, Property 5 requires that our model is able to filter which pieces of information are important and which are not. Again, in the case of age estimation, whether the background behind the person is outdoors or indoors should have little or no impact on the prediction of their age. In many, if not most, cases of deep learning applications, the majority of the input data has little to no relevance for the task at hand. The network must learn which information should be relied on and which data should be ignored.

This work uses the following design to produce a synthesized data generating distribution. First, we define a polynomial,

$$y = a_4x^4 + a_3x^3 + a_2x^2 + a_1x. \tag{5.1}$$

We set $a_1 = 1$. With $\mathcal{U}(r_0, r_1)$ representing a uniform distribution over the range from r_0 to r_1, a_3 is randomly chosen from $\mathcal{U}(-1, 1)$. a_2 and a_4 are randomly chosen from $b \cdot \mathcal{U}(-2, -1) + (1 - b) \cdot \mathcal{U}(1, 2)$ with b being randomly chosen from a standard binomial distribution. Then we sample y at 10 locations on the x-axis from linear space from -1 to 1. An example of such a polynomial and the observed points are shown in Figure 5.1. This one polynomial and the observed points constitutes a single example in our dataset. We choose the desired label of the task to be the a_3 value for each polynomial. That is, our network, when given the 10 observations, should be able to predict a_3.

We can compare the pieces of this data generating distribution to the standard image regression problem (think of age estimation from images) to better understand what parts of the toy model represent which parts in a real application model. The 10 observed values

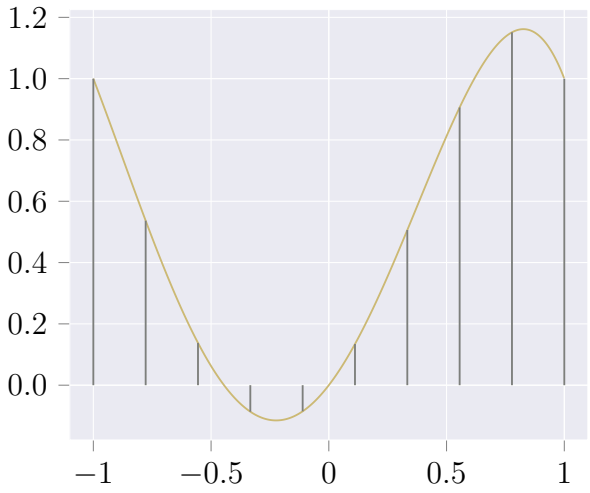


Figure 5.1: An example of a polynomial as described in Equation (5.1) with 10 points sampled. In this case, $a_2 = 2$, $a_3 = -1$, and $a_4 = -1$, but only a_3 is the coefficient to be estimated.

from the toy model are analogous to the pixel values in image regression. a_3 is equivalent to the object label (e.g. age value). Given the restrictions on how the coefficients are chosen, the set of all polynomials obtainable from Equation (5.1) is the underlying data generating distribution in the synthesized data case, which can correspond to the views of the real-world projections to an image plane in the regression case of age estimation.

This model fulfills all but the last property defined above. To satisfy Property 5, we make every example in the dataset consist of 5 different polynomials each chosen and sampled from as previously explained. However, for this single example (consisting of 5 polynomials) only the a_3 coefficient of the first example is the label. Thus, each example consists of 50 observations, only 10 of which are related to the label. Lastly, we apply noise to every observation.

5.1.2 Coefficient estimation experimental setup

In the coefficient estimation experiments, both the discriminator and generator each consisted of a 4 layer fully connected neural network. Each layer contained 10 hidden units. All

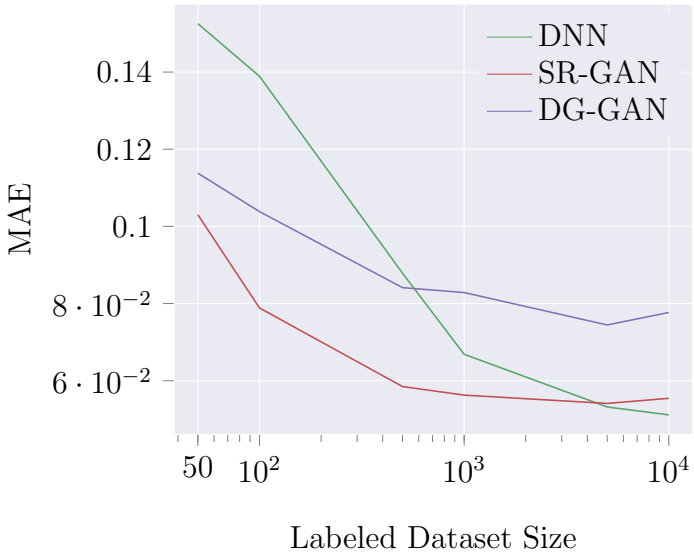


Figure 5.2: The resultant inference accuracy of the coefficient estimation network trained with and without the SR-GAN for various quantities of labeled data.

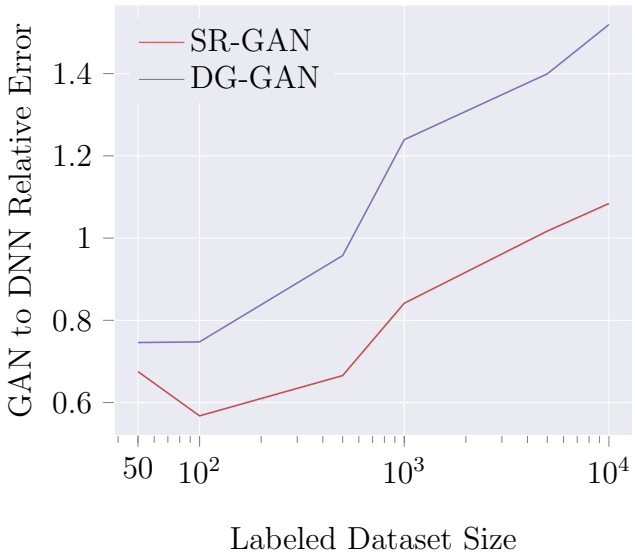


Figure 5.3: The relative error of the GAN model over CNN model for various quantities of labeled data for the coefficient model.

code and hyperparameters can be found at `https://github.com/golmschenk/srgan`. The training dataset for each experiment was randomly chosen. The seed for the random number generator is set to 0 for the first experiment, 1 for the second, and so on. The same seeds are used for each set of experiments. That is, the SR-GAN compared with the DNN use the same training data for each individual trial. Additionally, for experiments over a changing hyperparameter, the same seeds are used for each hyperparameter value.

In these experiments, we demonstrate the value of the SR-GAN on polynomial coefficient estimation. Using a simple fully connected neural network architecture, we have tested the DG-GAN and SR-GAN methods compared to a plain DNN on various quantities of data from the generation process described above. The results of these experiments can be seen in Figure 5.2. In each of these experiments, an unlabeled dataset of 50,000 examples was used, while various quantities (from 50 to 10,000) of labeled data were used. Each point on the plots is the average of three training runs randomly seeded to contain different training and test sets on each experiment. The relative error between the DNN and the GAN methods can be seen in Figure 5.3. We see a significant accuracy improvement in lower labeled data cases for the GAN methods. The SR-GAN error is 68% of what the DNN error is at with 50 labeled examples. With 50 examples, the DG-GAN also has a significant advantage with 75% the error the DNN has. However, the DG-GAN quickly loses its advantage over the DNN as the data size increases. As the amount of labeled data becomes very large, SR-GAN does not perform better than the DNN. This diminishing return is expected, as we can consider the case of infinite labeled data, where unlabeled data could then provide no additional useful information. We note that for the simple problem of coefficient estimation, 10,000 examples is a very large dataset for training. In each of the real-world applications we tested our SR-GAN method in, we did not see a detriment in using the SR-GAN with larger numbers of labeled examples.

5.2 Loss Function Analysis

5.2.1 Overview

Using the polynomial dataset experimental setup, a series of trials were run with various loss functions to determine the optimal functions to be used by the SR-GAN. The influence of the choice of loss functions and the influence of the weight of each type of loss was analyzed. The simplified key findings of this section are

1. The SR-GAN is robust to the choice of matching and contrasting loss functions, with varying functions resulting in an only minor impact on the predictive accuracy.

2. An abundance of weight on the matching loss relative to the contrasting loss results in overfitting. An abundance of weight on the contrasting loss relative to the matching loss causes longer training times but results in comparable accuracy to ideal training values. Therefore, it would seem best to error on the side of extra contrasting loss.

For the purpose of this discussion, the feature distance vector is given by

$$\boldsymbol{d}_f = |\mathbb{E}_{\boldsymbol{x}\sim p_1} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}\sim p_2} f(\boldsymbol{x})| \tag{5.2}$$

where p_1 and p_2 are the appropriate labeled, unlabeled, or fake data distributions depending on if the d_f is being used in the $L_{matching}$, $L_{contrasting}$, or L_G terms as given in Equations (4.3) to (4.5). With this, trials with three feature contrasting and three feature matching loss functions were performed. Every permutation of these loss functions was tested. First, a description of each loss function and why it was considered is given. Each loss function is given a shorthand notation to be used in the remainder of the discussion. It should be noted that matching functions have previously been used in GANs by other works, while contrasting functions are a novel concept designed for the SR-GAN specifically. The matching functions

are used both by the discriminator in $L_{matching}$ (matching the labeled image statistics to the unlabeled image statistics) and by the generator in L_G (matching the fake image statistics to the unlabeled statistics). The contrasting functions are only used by the discriminator in $L_{contrasting}$ (contrasting the unlabeled images from the fake images).

5.2.2 Matching and contrasting functions

The first matching loss function is given by

$$f_{square} = \|\boldsymbol{d}_f\|_2^2. \tag{5.3}$$

This matching function corresponds to the matching function used by Salimans et al. [21]. It represents the Euclidean squared distance between the two mean feature vectors in feature space. Both its intuitive meaning and its usage in existing works justifies the value of experimenting with this function.

The second matching loss function is given by

$$f_{abs} = \|\boldsymbol{d}_f\|_1. \tag{5.4}$$

Compared to f_{square}, this matching function allows for an individual feature to differ by a greater degree if multiple other features can be made more similar. In this case, the similarity of any feature is weighted equally rather than the feature with the greatest dissimilarity. This may reduce the impact of outlier features.

The third matching loss function is given by

$$f_{norm} = \|\boldsymbol{d}_f\|_2. \tag{5.5}$$

This is similar to f_{square} except the norm is not squared, and so the loss is on the raw

Euclidean distance of the feature vectors. This distance may reduce the impact of outlier distance vectors. Note, the distance vector comes from comparison the unlabeled data to the labeled data (or fake data), so the "outlier" in this case is a batch of images.

Next, we have the contrasting functions. The first contrasting function is given by

$$f_{-abs} = -\|\boldsymbol{d}_f\|_1. \tag{5.6}$$

There are two reasons this contrasting function is chosen. The first, and more obvious, reason is that it is simply the negated form of the matching function f_{abs}, providing it with direct adversarial loss. Here the goal is of the cost function is to make the two vectors as dissimilar as possible. The second reason, can be seen when comparing f_{-abs} to the matching function f_{norm}. The naive counterpart to f_{norm} would be the negated f_{norm}. However, this creates a contrasting function which places the most emphasis on the features which are already the most dissimilar. This quickly leads to the discriminator only focusing on a single feature's difference (the feature which already has the largest difference between the fake and unlabeled data). The discriminator then virtually ignores all other features (which the generator can then match its fake data closely to the unlabeled data in). In comparison, f_{-abs} will try to increase the difference of any feature. This provides a balanced adversarial goal to the matching f_{norm} function, as the matching will try to reduce the largest difference, while the contrasting will increase any value it can. As the matching will focus on the largest difference, the generator will increase the difference of all other features until they reach the same difference as the largest, at which point the discriminator will divert effort toward that feature as well. All feature differences will then be expanded equally, and an equilibrium between the feature differences is held.

The second feature contrasting function is given by

$$f_{-sqrt} = -\left\| \sqrt{|\boldsymbol{d}_f| + 1} \right\|_1. \tag{5.7}$$

Notably, compared to f_{-abs}, this contrasting function is not linear. For f_{-abs}, the cost decreases as a feature's difference grows; however, the slope of the cost remains the same—the relative cost decrease per unit difference remains constant. With this linear cost, under the right conditions, the discriminator may gain continual benefits by moving to increase a feature difference regardless of the cost incurred from the matching function. f_{-sqrt} creates a diminishing return. Increasing the difference between closely matched features provides relatively large benefits, while features with a large difference provide relatively little. As the difference grows larger, the discriminator will incur little cost from the contrasting function compared to the matching function. Such a non-linear function provides a built-in mechanism to prevent the discriminator from only optimizing a single goal.

The third feature contrasting function is given by

$$f_{-log} = -\|\log |\boldsymbol{d}_f| + 1\|_1. \tag{5.8}$$

The reasoning behind this function is similar to that of f_{-sqrt}, with the primary difference being how drastic the effects of the diminishing returns are.

In practice, the resulting value for f_{abs}, f_{-abs}, f_{-sqrt}, and f_{-log} is divided by the number of elements in $\boldsymbol{d}_f$ so that the scale of the cost per feature is independent of the feature vector size. Of course, a case could be made for numerous other functions; however, the choices here provide a diverse baseline of comparison.

5.2.3 Cost multiplier grid search

Each combination of matching and contrasting functions was tested and compared. To prevent benefits from the scale of the values of any combination, so that the interplay between the functions could be examined, a scalar multiplier was applied to the cost resulting from each function. A logarithmic grid search on these multipliers was performed. That is, each multiplier was independently varied by a factor of 10, and the network was retrained using the newly scaled loss. Implicitly, these multipliers are relative to the labeled loss, which is not scaled in any way. Figure 5.4 shows the validation curve during training for the optimal multipliers for each combination of matching and contrasting functions. It also shows the validation curves for the surrounding 8 positions in the multiplier grid search (i.e., the curves for each combination of multipliers increased and decreased by a factor of 10). Other multipliers tested are omitted to reduce clutter in the plots. Each of these trials was run for 100,000 training steps with the optimal multipliers being determined by the lowest final value.

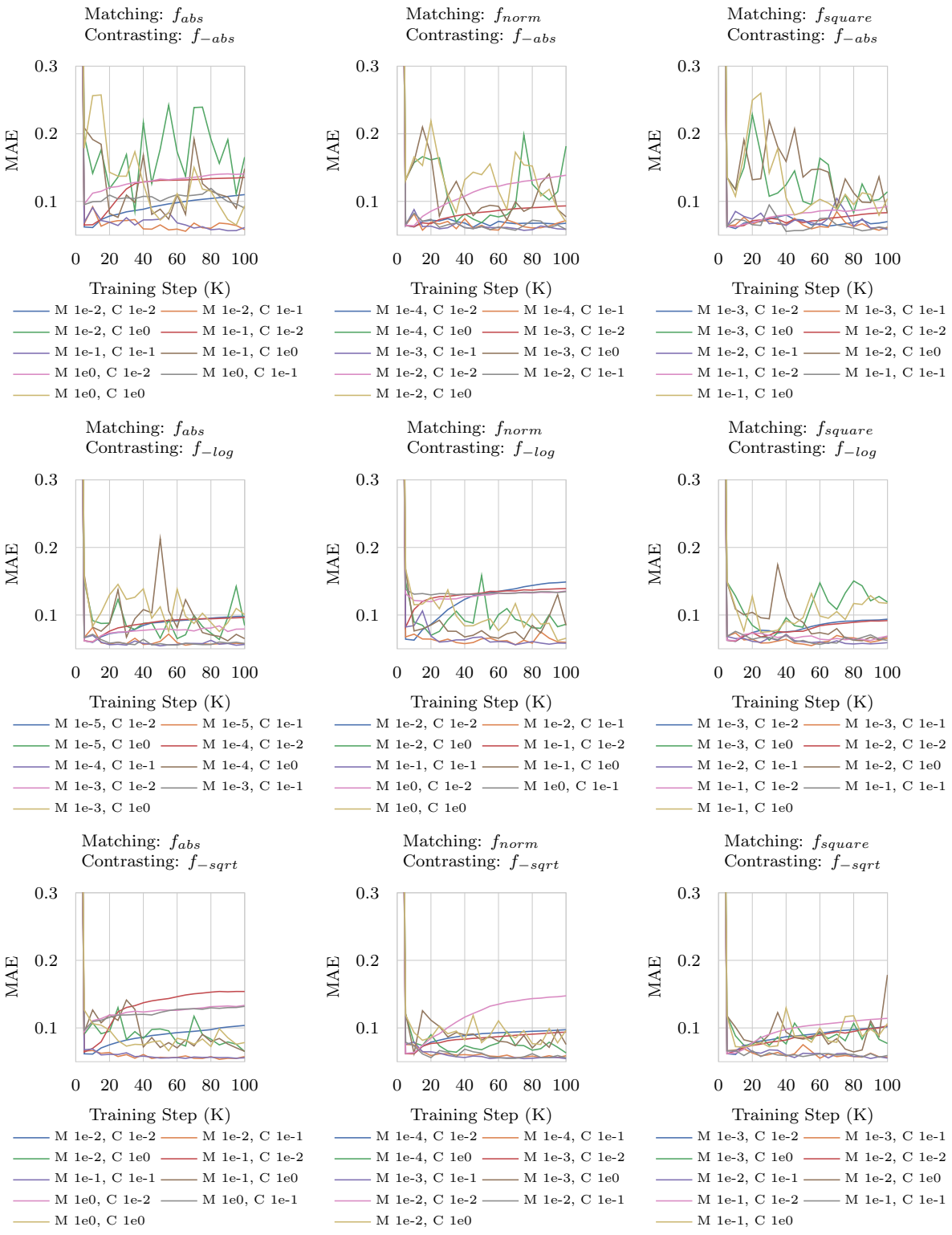


Figure 5.4: Trials with each combination of matching and contrasting functions. For each set of functions, the matching and contrasting loss multiplier was varied by factors of 10 until the optimal combination of multipliers was found. The plots show the optimal multipliers and all multipliers surrounding it.

Figure 5.4 provides several insights. First, it provides multiplier values which produce the lowest error for each combination of functions, allowing for the next part of the analysis, which is described shortly.

Second, it shows which functions are most affected by a change to the multipliers. For example, all 9 displayed validation curves of f_{abs} with f_{-log} end at ~0.1 or lower. In contrast, several of the curves of f_{abs} with f_{-abs} end at ~0.15. This suggests f_{abs} with f_{-log} is more robust to changes in multipliers, and therefore requires less parameter tuning for specific problems. It should be remembered that the range of multipliers shown in these curves is a factor of 100.

Third, these experiments show how relative changes from the optimal multipliers affect the training process. We see that when the matching loss multiplier is relatively high compared to the contrasting multiplier, the validation curve is much smoother, but overfitting occurs. Here, it is worth noting that the DNN ends with a value of ~0.77, which this overfitting often rises above. The easiest way to understand what is happening here is that the discriminator is ignoring the contrasting loss, then it sacrifices the supervised loss in favor of the matching loss, resulting in an error worse than that of the DNN. On the other end of the spectrum, with a relatively high contrasting loss compared to the matching loss, the validation curve is quite noisy. This is due to the generator competing with the discriminator. In each of these cases, the training does not seem to have converged within 100,000 steps, and it may be that this setup will lead to a lower final value with longer training even than the optimal after 100,000 training steps. If this is true, a relatively high contrasting loss is preferred, as it prevents overfitting, leads to a better result, and allows for a lenient range of multiplier values, even if it takes longer to train.

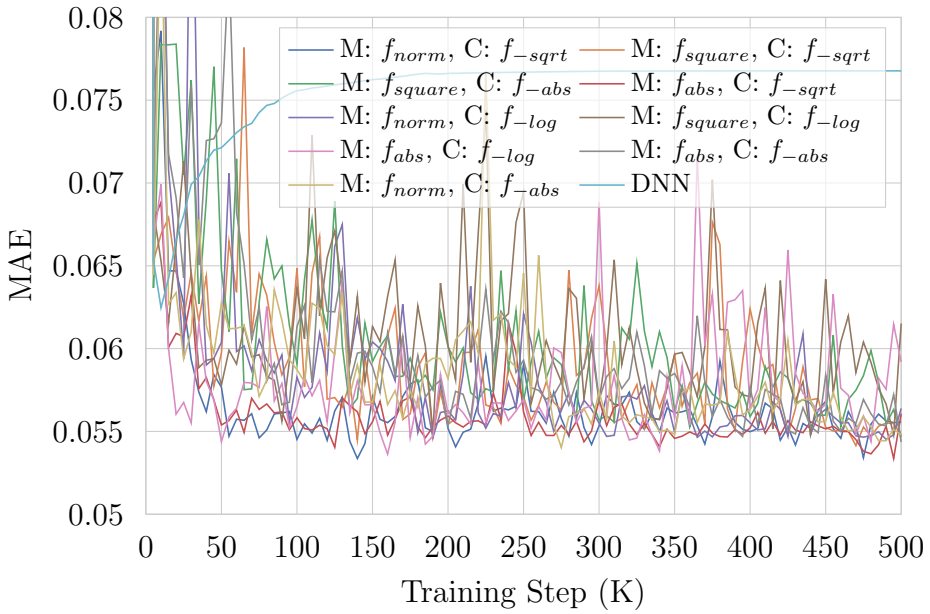


Figure 5.5: A comparison of each combination of loss functions using their optimal loss multipliers. These values come from running each trial for 500,000 training steps.

5.2.4 Optimal long trials

The next set of experiments ran each combination of functions using the optimal parameters for 500,000 training steps to allow for training to convergence. The validation curves of these trials are shown in Figure 5.5, and the final test errors are shown in Table 5.1. Here we see that a combination of f_{abs} and f_{-sqrt} performed the best. However, all the methods had similar test errors, which demonstrates that the SR-GAN method of using a combination of contrasting and matching methods is robust even to the choice of similarity functions.

The final experiment performed uses the combination of functions which performed best in the previous experiment, f_{abs} and f_{-sqrt}, and runs the surrounding logarithmic grid search multiplier values for the full 500,000 steps. This is done to determine how the training of these runs is affected by training toward convergence. The validation curves are shown in Figure 5.6. Here we see the expectation described in Section 5.2.3 fulfilled; Training using higher contrasting multipliers result in longer convergence time; however, a similar error to

Loss Functions		
Matching	Contrasting	MAE
DNN		0.0770
f_{square}	f_{-log}	0.0588
f_{abs}	f_{-log}	0.0579
f_{square}	f_{-abs}	0.0573
f_{abs}	f_{-abs}	0.0557
f_{norm}	f_{-abs}	0.0553
f_{norm}	f_{-sqrt}	0.0552
f_{norm}	f_{-log}	0.0551
f_{square}	f_{-sqrt}	0.0550
f_{abs}	f_{-sqrt}	0.0546

Table 5.1: A comparison of the final errors of each combination of loss functions using their optimal loss multipliers. These values come from running each trial for 500,000 training steps. The final 10 recorded values were averaged. Note that each recorded value is 5,000 training steps apart.

the optimal multipliers is eventually obtained. This suggests that it is advantageous to use a larger contrasting multiplier, as it reaches the same error rates while requiring less finely tuned multipliers.

Notably, these results provide a robust way to choose hyperparameters by selecting a strong contrasting loss compared to the matching loss. At the same time, the choice of loss functions is relatively unimportant, so long as they are stable in that the slope increases the farther from the desired value. This interplay between the contrasting and matching losses provides a form of auto-balancing the weights to be applied to each loss, greater reducing the manual effort and application specific parameter tweaking required.

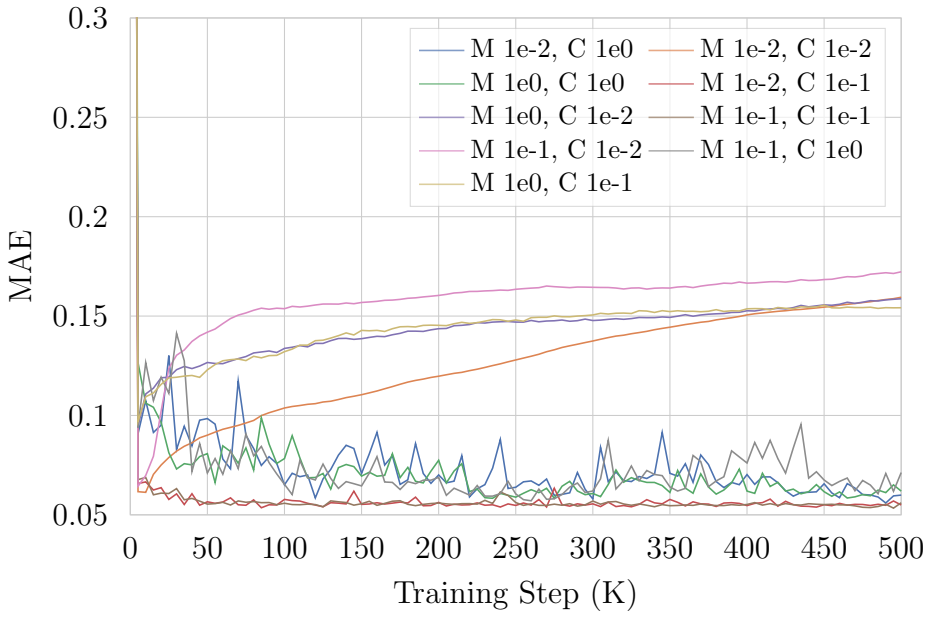


Figure 5.6: A comparison of the best combination of loss functions (f_{abs} and f_{-sqrt}) using optimal and surrounding loss multipliers. These values come from running each trial for 500,000 training steps.

Chapter 6

Comparing SR-GANs to DNNs Through Age Estimation

Age estimation is a well-known regression problem in computer vision using deep learning. In particular, well-established datasets of images of individuals with corresponding ages exist and are widely used by the computer vision community. A notable, large-scale age estimation database is the IMDB-WIKI Face Dataset [42].

For our work, a well-known dataset is particularly important as the deep learning community tends to focus on classification problems and not regression problems. Due to this, well-known regression datasets—ones known even outside their domain—tend to be rare. The age estimation dataset is one of these rare cases. It provides a widely used standard on which we can test our SR-GAN.

The primary focus of this section is on the improvements afforded by using the SR-GAN compared to a DNN. The discriminator in the SR-GAN matches the DNN network throughout these experiments.

6.1 Dataset

The IMDB-WIKI dataset includes over 0.5 million annotated images of faces and the corresponding ages of the people thus imaged. There are 523,051 face images: from 20,284 celebrities, 460,723 face images are from IMDb and 62,328 from Wikipedia. 5% of the celebrities have more than 100 photos, and on average each celebrity has around 23 images.

There are likely many mislabeled images included in the dataset. The image-label pairs were created by searching the top 100,000 actors on IMDb (also known as the "Internet Movie Database"). The actors' IMDb profile and Wikipedia page were scraped for images. Face detection was performed on these images, and if a single face detection is found, the image is assumed to be of the correct individual (which in many cases, it is not). The image timestamp along with the date of birth of the actor is used to label the image with an age. The image is often a screen capture of a movie, which may have taken years to produce or the screen capture may have happened years later.

Additionally, the actor may be purposely made to look a different age in the movie. Despite these many areas of mislabeling, random sampling checks done by human evaluators demonstrate that the dataset consists overwhelmingly of correctly labeled images. To minimize the number of incorrectly labeled images, the database is filtered based on several criteria. The database includes face detection scores (certainty of containing a face) and a secondary face score (containing an additional face). If the first face score was too low, the image was excluded. If there was a secondary face detected it is also excluded (since these are taken from the actor's IMDb page, it is only assumed to be a picture of the actor if there is only one person in the image). Images labeled with an age below 10 or above 95 are also excluded. Primarily, the filtering of images labeled with age 10 or below is important, as low ages are typically from the timestamp calculation being faulty. The age range limitations also remove many clearly mislabeled images, often with negative ages or ages in the thousands.

Figure 6.1: Typical example images from the database.

Finally, only images of 256×256 resolution or higher are used. After this filtering, we are left with ~90K images. Both the labeled and unlabeled data are taken from these images (without overlap), and the labels were not used for the unlabeled data. The data was selected randomly for each trial. Though other face data could be used for the unlabeled data, for these experiments, we wished to ensure that the labeled and unlabeled data came from the same data distribution.

(a) Fake age images.

(b) Real age images.

Figure 6.2: Examples of real and fake images used/generated during training. We note that our approach is not intended to produce realistic looking images, and the fake images are only included for insight.

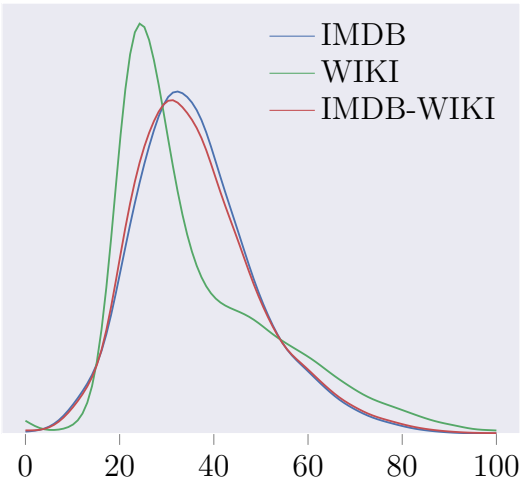


Figure 6.3: The distribution of ages in the IMDB-WIKI database.

6.2 Experimental Analysis

In the age estimation experiments, the DCGAN network architecture [26] is used. All code and hyperparameters can be found at `https://github.com/golmschenk/srgan`. The discriminator of the DCGAN architecture used alone functioned as the CNN baseline model. The network structure can be seen in Figure 6.4. The training dataset for each experiment was randomly chosen. The seed is set to 0 for the first experiment, 1 for the second, and so on. The same seeds are used for each set of experiments. That is, the SR-GAN compared with the CNN use the same training data for each individual trial. This set of experiments used the contrasting loss function of f_{square} and the matching loss function of f_{-log} from Section 5.2. These were chosen before the extensive loss function comparison was performed; however, as noted in Section 5.2, the choice of loss functions has little impact on the final accuracies.

The following experiments demonstrate the value of the SR-GAN on age estimation. Using a DCGAN [26] network architecture, we have tested the SR-GAN method on various quantities of data from the IMDB-WIKI database. The results of these experiments are shown in Figure 6.5. In each of these experiments, an unlabeled dataset of 50,000 images was used, whereas the size of the labeled data samples varies from 10 to 30,000. Each point on this plot is the result of a single randomly seeded training dataset. For each labeled dataset size, five trials were run. The relative error between the CNN and the GAN models is shown in Figure 6.6. We see a significant accuracy improvement in every case tested. At 100 labeled examples, the GAN achieves an MAE of 10.6, an accuracy which is not achieved by the CNN until it has 5000 labeled examples available for training. At 100 labeled examples, the GAN has 75% the error that the CNN does.

The advantage of the SR-GAN drops to near zero as the number of images approaches the number of unlabeled examples being used. There seem to be two likely causes for this.

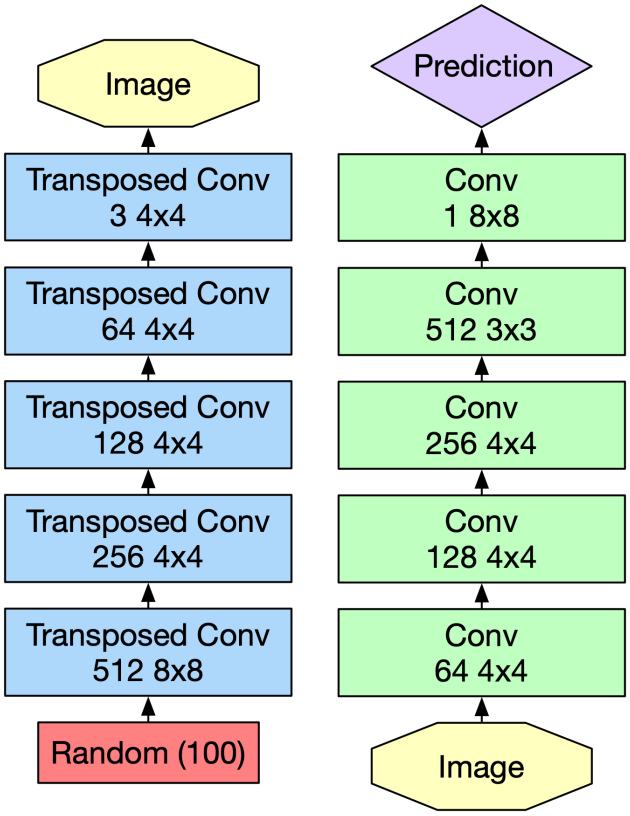


Figure 6.4: The DCGAN structure used for the age estimation experiments. The left network is the generator and the right is the discriminator/CNN.

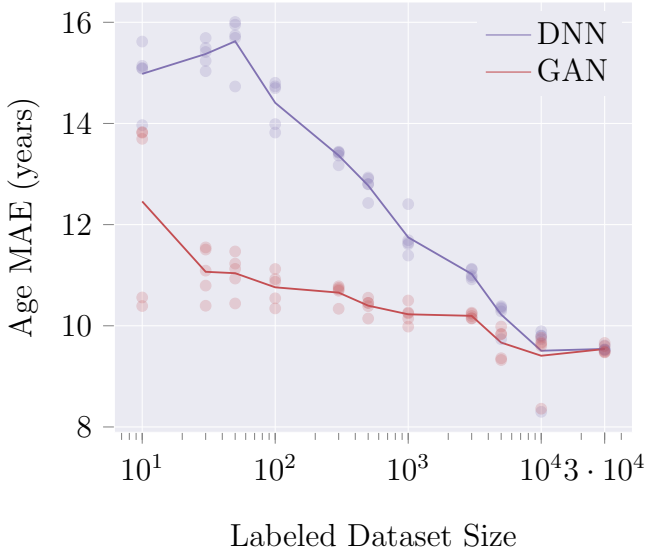


Figure 6.5: The resultant inference accuracy of the age estimation network trained with and without the SR-GAN for various quantities of labeled data. Each dot represents a trial with randomized training data, and the line represents the mean of the trials.

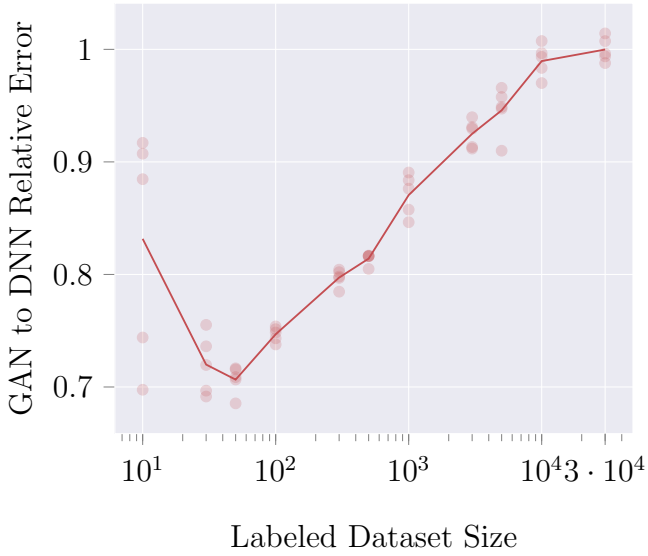


Figure 6.6: The relative error of the GAN model over CNN model for various quantities of labeled data for age estimation. Each dot represents a trial with randomized training data, and the line represents the mean of the trials.

Either, there are enough training images that the network is at capacity (additional images will not further improve the results), or the ratio of labeled to unlabeled images is too small for the generator to be of more benefit to the discriminator. Unfortunately, the number of images available in the IMDB-WIKI dataset makes it difficult to pursue a larger number of training examples further.

Chapter 7

Comparing GAN Methods for Steering Angle Prediction

This section focuses on another real-world application regression task: driving steering angle prediction. In this case, the network predicts the steering angle of a car given a front-facing image from the windshield of a vehicle. Such an approach can be considered a form of basic partial self-driving/auto-pilot capabilities using a single image [43]. Specifically, the method allows an algorithm to mimic the steering of a human by attempting to choose the same steering wheel turning angle a human driver chose given the same upcoming road segment. While this is far from a complete self-driving solution, it provides an application simple enough to focus on the semi-supervised methods, while still being relevant to developing state-of-the-art technologies.

The primary focus of this section is to demonstrate the superiority of the SR-GAN compared to other potential semi-supervised regression GAN methods, especially those expounded in Section 2.3.

7.1 Related Work

For the application of driving angle steering estimation from a single image, Rezagholiradeh and Haidar [30] presented two semi-supervised regression GAN approaches. As of this writing, this is one of the only cases where semi-supervised GANs have been applied to regression tasks beyond our work. The first approach they present is a dual goal GAN (DG-GAN) approach, as was described in more detail in Section 2.3.2. In their work, they refer to this approach as Reg-GAN Architecture 1. The DG-GAN outputs two labels: a regression value prediction of the driving steering angle and a fake/real classification prediction. The idea is that the network must learn both how to distinguish between real and fake examples, and how to predict the correct value for a labeled example. However, this approach does not enforce that these two predictions be related. Part of the network may learn the task of identifying real/fake images, while another portion of the network learns the task of predicting regression values. A visual depiction of this split learning is shown in Figure 2.5. If the objective of distinguishing being real and fake examples is weighted strongly enough, the network may devote larger portions of the network to the real/fake classification task, thereby reducing its effectiveness in the regression prediction. The experiments show the proposed SR-GAN method outperforms the DG-GAN, both in the implementation proposed here and in that of Rezagholiradeh and Haidar [30].

Rezagholiradeh and Haidar [30] also present a second semi-supervised regression GAN method which they refer to as Reg-GAN Architecture 2 and is a specific version of the Explicit Label GAN described in more detail in Section 2.3.1. This method only outputs the single driving angle regression value, and then attempts to label this value as fake or real depending on if the value lies within the range of real values from the dataset. This method has two significant limitations. 1) If the full range of the unlabeled dataset is unknown, a correct angle prediction would be incorrectly labeled as fake data. Rezagholiradeh and Haidar [30]

assumes the range of the unlabeled data is known. 2) A bias is introduced, as values near the boundary between fake and real are preferred. This is because a generator which can exactly duplicate unlabeled data would cause the discriminator to pick a value halfway between fake boundary and real steering angle value as the best possible answer.

Lastly, as a baseline comparison, Rezagholiradeh and Haidar [30] also tested converting the regression task into a discrete classification task by binning the regression values. For this baseline approach, the range of angles was discretized into bins, and the discriminator then uses a standard multinomial classification. For each bin, the predicted label is then the center value for that bin, and this is used to determine the error in angle. The general limitations of this approach are described in Section 2.3.3. In this specific application, it requires manual selection of the (unknown) boundaries of the steering angle range and an arbitrary selection of the number of bins. The precision and training issues of the binning regression to classification method apply here, as with any regression task.

7.2 Experimental Analysis

Here, we performed the experiments presented by Rezagholiradeh and Haidar [30] using our SR-GAN approach and compared the results to the methods presented by Rezagholiradeh and Haidar [30]. In these experiments, the dataset [44] consists of 45,567 images from a dashboard-mounted camera. A device attached to the steering wheel of the vehicle measured the angle of the steering wheel rotation. A human driver operated the vehicle during the recording process. Each image is paired with its corresponding measured steering angle. The task of the network is then to predict, from a single image of the forward-facing camera, what steering angle the human driver selected. Varying numbers of labeled images randomly selected from the entire dataset are used for training (up to 7,200 images) and testing (9,000 images). The remaining images are used as the unlabeled data. We use the DCGAN network

architecture [26], which matches the architecture presented by Rezagholiradeh and Haidar [30]. This network structure (both generator and discriminator) is shown in Figure 7.1. All code and hyperparameters can be found at `https://github.com/golmschenk/srgan`. We note that we cannot precisely duplicate the experiments by Rezagholiradeh and Haidar [30], as the images used for training and testing were randomly chosen, and the specific images selected were not provided by Rezagholiradeh and Haidar [30]. However, multiple trials with different random sets of images demonstrated that the change in accuracy is negligible. We similarly randomly selected our datasets. Our random selections were seeded for reproducibility, and the code at our repository can be used to retrieve the dataset selection for our experiments. Examples of the images, both real and fake, used and generated during training are shown in Figure 7.2.

We also note that an entirely random image selection has limited evaluation value for this dataset. The images are part of a video sequence with each image have only minor differences from the previous image. Even a small percentage of the images, when randomly chosen, would contain the primary attributes of a large portion of the dataset. However, for comparison purposes, we have followed the experimental procedure used by Rezagholiradeh and Haidar [30]. In addition to repeating the configurations used by Rezagholiradeh and Haidar [30], we have provided results for significantly lower numbers of labeled images.

The evaluation metric used is a normalized mean absolute error (NAE) given by

$$NAE = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{y_{max} - y_{min}} \times 100\%. \tag{7.1}$$

This metric was chosen to match the metric used by Rezagholiradeh and Haidar [30].

The results of our method in comparison to the methods presented by Rezagholiradeh and Haidar [30] are shown in Table 7.1. In these experiments, we show that our SR-GAN method significantly outperforms the Reg-GAN method for any number of labeled examples.

	Labeled examples					
Method	100	500	1000	2000	4000	7200
Improved-GAN (Binning classification GAN)	-	-	4.38%	4.22%	4.07%	4.06%
Reg-GAN-1 (DG-GAN)	-	-	2.43%	2.40%	2.39%	2.36%
Reg-GAN-2 (Explicit Label GAN)	-	-	3.81%	3.58%	2.23%	2.21%
SR-GAN	3.12%	2.32%	2.02%	1.89%	1.37%	1.16%

Table 7.1: Steering angle prediction NAE compared to existing approaches for various amounts of labeled training examples.

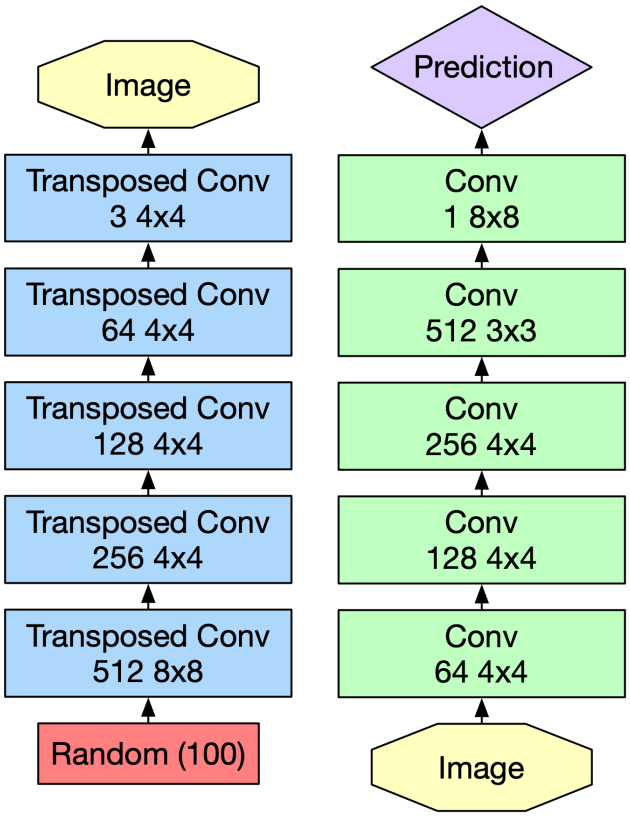
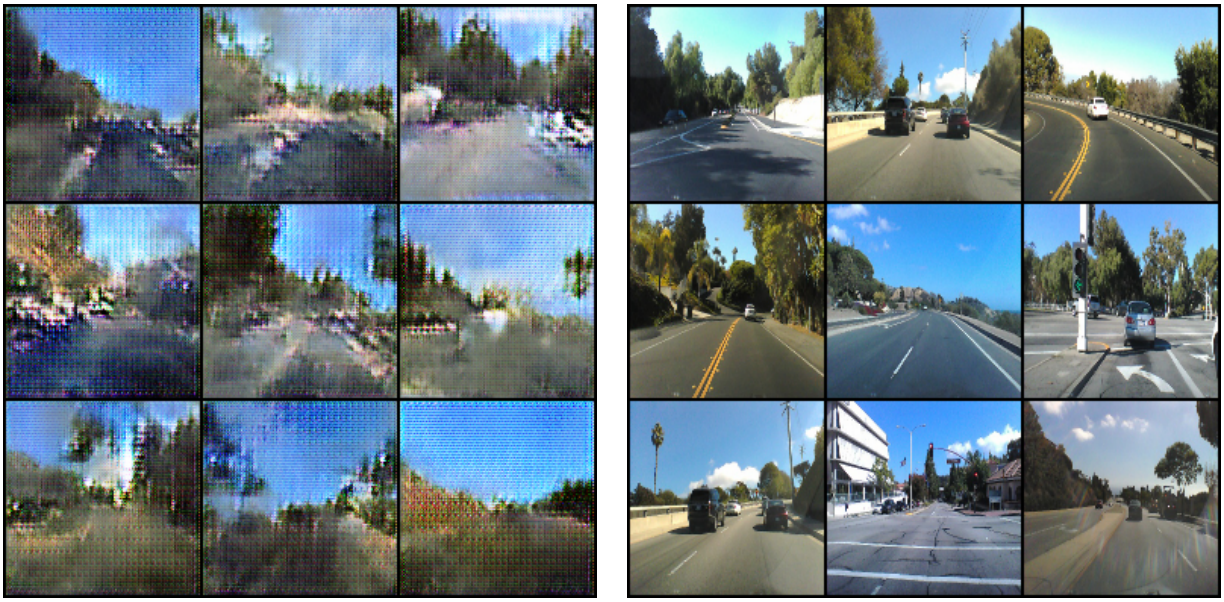


Figure 7.1: The DCGAN structure used for the driving steering angle experiments. The left network is the generator and the right is the discriminator/CNN.



(a) Fake steering angle images. (b) Real steering angle images.

Figure 7.2: Examples of real and fake images used/generated during training. We note that our approach is not intended to produce realistic looking images, and the fake images are only included for insight.

As it requires no arbitrary or manually chosen specifications based on the specific task of steering angle estimation, the Reg-GAN Architecture 1 (DG-GAN) is the more generalized approach of Reg-GAN and provides the comparison of the most interest. However, each comparison provides interesting insights.

In these results, we see that the SR-GAN method outperforms all other approaches in every case. In particular, it outperforms all three of the potential alternative GAN approaches for regression presented in Section 2.3.

Converting the regression task into a classification task through binning provided the worst results. This method suffers from two competing limitations. First, large bins result in less precision, as the angle predicted for any bin is the center value of that bin range. The second and more drastic issue is the lack of affiliation between adjacent bins in this approach. Though adjacent bins represent nearby values in the regression value, the loss function for classification does not acknowledge any such relation. If the correct label is in bin 1, and

the network receives the same loss for predicting either bin 2 or bin 10, despite the fact that regression value of bin 2 is much closer to the correct answer. As the binning becomes smaller, these problems become more detrimental, and to avoid selecting the incorrect bins the network must heavily overfit the data. Such training losses do not promote the network finding an optimal solution. Further, this binning choice is arbitrarily chosen, both in the size of the bins and the range of the bins, requiring manually tuning for each use case.

Next is the Reg-GAN Architecture 1, which is a method equivalent to our DG-GAN. The limitations of the DG-GAN have already been discussed and demonstrated in Chapter 5. We see a similar result here, where the DG-GAN performs well initially, but seems to plateau, again, likely due to the network splitting the weights toward two separate goals rather than using all weights for both goals.

The Reg-GAN Architecture 2, uses an explicit fake label, where the researchers chose a specific range of values to be considered a "real" label and anything outside this range was considered the "fake" label. This introduces a bias in the predicted value to move the prediction toward the edges of the range. This is most easily understood by considered an ideal generator which can exactly generate images from the original dataset. In this case, the best answer the discriminator can pick is halfway between the correct answer and the edge of the range (as the image may be a real image or a fake image that exactly duplicates the real image). Obviously, this bias does not improve the answer in this case.

The SR-GAN outperformed each of these alternatives with every sized labeled dataset. Notably, the SR-GAN using only 500 images performs on par with the best of the other methods using 7200 images. For labeled dataset sizes of 1000 and more, the SR-GAN significantly outperforms any other method using the full 7200 labeled images.

Chapter 8

Applying SR-GANs to State-Of-The-Art Crowd Counting Methods

In this section, the open and rapidly developing field of dense crowd counting is the target application. Here our method provides immediate gains to state-of-the-art network architectures. While both the age estimation and driving steering angle cases were real-world scenarios, they are relatively simple tasks. In contrast, dense crowd counting is a challenging problem which is still rapidly developing.

Every year, crowds of thousands to millions gather for protests, marathons, pilgrimages, festivals, concerts, and sports events. For each of these events, there is a myriad of reasons to desire to know how many people are present. For those holding the event, both real-time management and future event planning is determined by how many people are present, their current locations, and the intervals at which people are present. For security purposes, evacuations planning and where crowding might be a potential harm to individuals is

Figure 8.1: Examples of moderately crowd images and corresponding density labels. Note, the images include regions of interest in the labeling, and as such may not have density labeled in some parts that appear crowded in the image (particularly for distant individuals).

dependent on the size of the crowds. In journalistic pursuits, the size of a crowd attending an event is often used to measure the significance of the event.

With the advent of deep learning, convolutional neural networks have been the most successful means of analyzing crowd images. However, a major limitation of CNNs is the extensive data required. Particularly with the manual effort required to label such data, where each person needs to be labeled individually, and ideally have their entire body segmented as an entire person (while taking into account they may be partially occluded). This is where

the semi-supervised GANs can help in the reduction of data requirements. Security footage is plentiful; only the labeled version of it is difficult to come by.

In particular, we tackle the problem of counting the number of people and the density of people per pixel in an image. Especially crowded scenes provide the most challenging situations, where people are represented only by a few pixels, and individuals are highly occluded. Furthermore, we approach the problem of how crowded each portion of the image is on a per-pixel basis. This requires that the person density per pixel is determined. An example of what this data looks like can be seen in Figure 8.1.

Crowd analysis is a particularly worthwhile application to test our GAN methods on. This is because it provides a real-world situation that has immediately usable applications. It is a very complex and challenging problem, even when attempted with sufficient data to train a normal CNN. A successful application of the SR-GAN in this realm demonstrates the generalizability of the method, it's capability to handle even complex situations, and can be used to illustrate the challenges which need to be overcome in such a complex application. Furthermore, this particular area is in need of such a data requirement reducing method, as labeling this kind of data is a challenge of its own, to the point that simulated datasets [45] and annotation tools [46] have been created specifically to ease the effort required in labeling data such as this.

Specific properties of crowd analysis also lend themselves to demonstrating the properties of various SR-GAN implementations. For example, in the case of crowd counting, there can never be negative people in an image, only a minimum of zero. This means a negative number could be used to represent a fake image from the generator in the explicit label GAN. However, such a representation would introduce bias into the model, as previously noted. That is, predicting a low count for a fake image would be more correct (closer to negative) than predicting a high count for a fake image. The direct use of the output label may be able to provide more accurate overall training, but the bias may counteract this gain.

8.1 Related Work

Zhang et al. [4] provides one of the first uses of convolutional neural networks (CNNs) as a method for crowd counting, especially across multiple scenes. While other works have made valuable alterations to the approach given in this paper to produce improved state-of-the-art results (such as multiple-scale CNNs [47], or residual network skip connections [48]), the primary functionality of these approaches, namely the ability to use CNNs for crowd counting, is still similar to that which is presented by Zhang et al. [4]. In our work, the discriminator is similar to the CNN of Zhang et al. [4], notably in the use of the joint optimization goal of two outputs of the CNN, one for count prediction and the other for density prediction.

In Li et al. [49], a GAN is used in crowd counting. The GAN in Li et al. [49] is used to improve the accuracy of the crowd counting prediction. The GAN in this work consists of a conditional generator with an input of true images and an output of generated density maps. A discriminator network then attempts to distinguish between the generated density maps and the true density maps. Once this GAN is trained, the generator is used to produce density maps of the test images, and these results are used as the predicted density maps. That is, the generator is used for inference in the testing phase. The formulation of this GAN is significantly different from our approach. Most notably, the GAN in Li et al. [49] is not designed to train with unlabeled data or to specifically reduce the amount of data required. Li et al. [49] expects the training images have the corresponding labels which can be used for training the discriminator. Our approach allows for the use of unlabeled data to train, with the goal of requiring significantly less labeled data. The difference in the goal of the GANs is also reflected in the structure of the networks being completely different. These differences include but are not limited to the generator outputting a density label vs an image, the generator predicting the labels vs the discriminator predicting the labels, and the generator using true images as input vs only using random noise as input.

8.2 Explicit label GAN Design for Crowd Counting

Previously to developing the SR-GAN method, we developed an explicit label GAN as described in Section 2.3.1 for the purpose of crowd counting and density prediction. This early work led to the recognition that the explicit label GAN had flaws, and eventually produced the goal of the SR-GAN. As this background insight is important, this section describes the early explicit label GAN formulation. However, it is important to note that the rapidly evolving field of dense crowd counting resulted in new network architectures and datasets by the time the SR-GAN was developed. For the SR-GAN experiments, we opted to compare with state-of-the-art networks rather than our earlier explicit label GAN approach. Thus, this first set of experiments are primarily to provide insight into the limitations of the explicit label GAN. In particular, we can note the various domain-specific considerations that need to be accounted for using the explicit label GAN for crowd counting.

8.2.1 Discriminator

A label in this model is a crowd density map. This map consists of an array of real numbers summing to the total number of individuals shown in the image. The density for each individual person is spread over the array using a 2D Gaussian kernel. Such density maps are shown in Figure 8.1. This spreading of values provides a smoother training gradient for the network across the density map label. We define our supervised loss using an $L_{1,1}$ loss (equivalent to L_1 if the matrix is flattened) over the elements of the true density labels compared with the predicted ones,

$$L_{supervised} = \mathbb{E}_{\boldsymbol{x},y\sim p_{data}(\boldsymbol{x},y)} \left\| y - D(\boldsymbol{x}) \right\|_1 . \tag{8.1}$$

Note that our supervised loss function omits a logarithm, as we are using a Wasserstein GAN [41]. GANs have been shown converge more consistently using a loss function based on an Earth Mover's distance (or Wasserstein distance) even among classification problems [41]. Our network also uses a discriminator training gradient penalty of Wasserstein GANs as opposed to the weight clipping proposed by Arjovsky, Chintala, and Bottou [41], as weight clipping was shown to result in a GAN which produces pathological behavior while gradient penalty results in more consistent convergence over a wider range of network architectures [37].

In the classification case, there were two terms to the unsupervised loss. For clarity, in the following description of the proposed approach, we define these two terms separately with

$$L_{unsupervised} = L_{matching} + L_{generated}. \tag{8.2}$$

This way we can define the intuition for the definition of $L_{matching}$ and $L_{generated}$ individually. For the case of loss from the generated images, we wish to punish the discriminator for seeing any amount of people (any density), as the generated images contain no true images of people. So here our loss is given by

$$L_{generated} = \mathbb{E}_{\boldsymbol{x}\sim G} \left\| D(\boldsymbol{x}) \right\|_1 . \tag{8.3}$$

With this, in a sense our equivalent of the $K + 1$th class is zero density. Note however, that even real images contain areas of pixels with zero density.

As we do not know the true label for the unlabeled images, training the discriminator toward an exact value can be detrimental toward the overall accuracy. For example, consider a case where we assume, based on the labeled images from the camera, that the unlabeled image has a label $\hat{y}$, but the true (unknown) label is y. If the discriminator predicts y, and our loss function has no leniency, the discriminator would be trained to move away from

the correct answer it predicted towards $\hat{y}$. Instead, we use a loss function which allows for a range of "correct" answers as we do not know which is true. Specifically, we use a loss function for which a range of input values produces zero loss. That is, if the difference of the predicted from the true value is small enough, no loss is produced. However, beyond a given difference threshold, the loss is non-zero. Specifically, our unlabeled loss is

$$\begin{aligned} L_{unlabeled} = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})} \Big[& \\ & \max\left(\frac{1}{\alpha}\sum y_e - \sum D(\boldsymbol{x}), 0\right) + \\ & \max\left(\sum D(\boldsymbol{x}) - \alpha \sum y_e, 0\right) \Big] \end{aligned} \tag{8.4}$$

where α is an experimentally chosen hyperparameter and is greater than 1. y_e is average labeling of known count labels for the labeled images being used for training (in implementation, these are the images for a given step of training). Note here that the values are summed before differences are computed. This is because there is no information on the locations of the person densities in unlabeled images, and only the approximate count is useful (i.e., comparing density maps would not be).

8.2.2 Generator

The generator is trained on a modified version of the feature matching loss described by Salimans et al. [21]. Once again, the main difference in our generator as compared to Salimans et al. [21] come from training in the case of regression targets, and more specifically, our crowd counting case. In this case, we want the generator's goal to be able to produce highly crowded examples (as interpreted by the discriminator). Here we use feature matching as the generator goal as described by Salimans et al. [21]. Normally, feature matching is simply used to match the features that arise in the discriminator from the real images. Instead, we want features which result in the highest predictions for density and count to be the goal of

the generator's matching. That is, we don't want the generator to match the features for floors or walls, but instead to match the features in areas the discriminator sees as containing crowds of individuals. If this were not the case, the discriminator and generator could "agree" to have the generator produce realistic, uncrowded images, as it could meet both networks' goals. To force the generator to work toward features that represent crowded portions of the images, each feature vector on the intermediate layer is weighted based on the predicted output value of that vector by the discriminator. In this way, the generator tries to produce images whose feature vectors in the discriminator match the feature vectors in the real data that have the largest count and density predictions. With $f(\boldsymbol{x})$ denoting the activations on the final layer before the output layer and $\boldsymbol{z}$ being noise to input into the generator, the generator loss is given by

$$L_G = \left\| \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \left[\frac{D(\boldsymbol{x}) \odot f(\boldsymbol{x})}{\sum D(\boldsymbol{x})} \right] - \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}} f(G(\boldsymbol{z})) \right\|_1 \tag{8.5}$$

The second term of this loss function is simply getting the expected values for the activations of an intermediate layer of the discriminator for fake images. That is, the mean features of fake data. The first term does something similar for real data. The only difference is that the real features are weighted by how much crowd density they correspond to. Thus, the generator tries to produce images that have features similar to more crowded images.

8.2.3 Dual optimization goal

One additional complication comes from training a network with a second optimization goal. As shown by Zhang et al. [4], better crowd counting results can be achieved by training a network to produce both a density map and a separate total count value. The last layer of our network is actually two layers in parallel, one for density and one for count. The density layer is trained to produce a density map which matches the true values of the label. The

count layer is trained to produce values which, when summed, match the true summed value of the label. Because of this, we actually have two losses for both discriminator and generator. Luckily, other than which output of the discriminator is used, all the losses are identical in both cases except the labeled loss of the discriminator for the case of the count. That loss is given by

$$L_{supervised} = \mathbb{E}_{\boldsymbol{x},y\sim p_{data}(\boldsymbol{x},y)}\left[\sum y - \sum D(\boldsymbol{x})\right]. \tag{8.6}$$

Note that the only difference is that rather than the norm distance between the arrays, the loss is the difference of the sum of the arrays. We use the same network for our CNN as Zhang et al. [4].

8.3 Experimental Setup and Results

For both the explicit label GAN results and the SR-GAN results, the primary metric we are interested in is the standard in the dense crowd counting field, namely, the mean absolute count error (MAE). However, several other metrics have been put forward as interesting by the research community. For the SR-GAN, we also consider the normalized absolute count error (NAE) and the root mean squared count error (RMSE). These are given by the following equations:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{C}_i - C_i\right| \tag{8.7}$$

$$\text{NAE} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|\hat{C}_i - C_i\right|}{C_i} \tag{8.8}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{C}_i - C_i)^2} \tag{8.9}$$

where C_i is the true count for a given image, and $\hat{C}_i$ is the predicted count.

8.3.1 Explicit label GAN experiments

The Shanghai Jiao Tong University WorldExpo'10 crowd counting dataset is used for our experiments. The dataset includes images and videos from 109 cameras and consists of a training dataset and a testing dataset. We used the WorldExpo'10 training data, and from it generated a set of training datasets, and a single validation dataset, and a single testing dataset (in addition to the original WorldExpo'10 testing dataset). Our training datasets (as a group), validation dataset, and each testing dataset are entirely disjoint from one another, in particular, disjoint in their use of cameras/scenes. CNN and GAN models are trained using a varying number of cameras (1, 3, 5, 10 and 20) and varying number of images (1, 3, 5, 10 and 20) per camera in order to systematically compare the performance of the two models; details of the selections are provided in Section 8.3.1. However, the same testing datasets and validation dataset are used for all the trained models.

One testing dataset is simply the WorldExpo'10 testing dataset, and it consists of 600 images from 5 cameras. We want to note that no images from cameras in the testing dataset are included in the training dataset and vice versa. That is, training and testing are performed across scenes. The datasets are disjoint in camera source selection. Each camera from this test dataset has 120 images. This dataset contains 2 of the most crowded scenes (in the entire WorldExpo'10 database), 2 of the least crowded scenes (and 1 additional scene). While this provides an excellent, challenging case to test on, it is not very representative of the training data. As such, we prepared the additional testing dataset. This provides better insight into how accurate the methods would be when testing on data whose statistics more closely match the training data statistics (i.e., this dataset is specifically chosen randomly from the same data generating distribution, rather than hand-chosen to be a challenging case). The second testing dataset is from a subset of the WorldExpo'10 training dataset. This testing dataset consists of 370 images from 10 cameras. These cameras were randomly

chosen from among the full dataset before any models were trained. No images from cameras in the validation dataset are included in the training dataset and vice versa. Again, the two datasets are disjoint camera source selection as well. We note that in general results are better on our randomly chosen testing dataset than on the original testing dataset. Finally, the validation dataset consisted of a different 307 images from 10 cameras and was only used to try different models and examine how well they generalize without comparing against the final testing datasets.

In each of the following experiments, we train both a CNN model and a GAN model on the same set of labeled data (for each of the camera-image number combinations). The GAN additionally uses the unlabeled images from the cameras in the trial's training dataset (the CNN has no way to profitably use this data). Each resulting model is used to predict the person count of each image in both test datasets. The accuracies between the two methods are compared. In every case, the GAN uses as its discriminator a network identical to the CNN. As we compare the two methods using a varying number of cameras and a varying number of images per camera, this gives both an understanding of the amount of total image data required by each network as well as the distinct scene information (number of cameras) required by each network.

The unlabeled data from the GAN model consists of video from which the labeled image data was taken for training. The GAN allows for any number of these frames to be used with no additional labeling (which is the primary advantage of the GAN). As such, in these experiments, we used all (unlabeled) video data for any camera within the training set (that is, the number of cameras the GAN has access to is still limited). Though the video lengths vary, on average there are approximately 2 minutes of video for each camera with 50 FPS. It should be noted that as this is video data, many of the frames are near identical to others.

The process of selecting cameras is done entirely randomly, and this random selection process was not repeated based on trial results. For each set of experiments, the cameras

and images chosen are consistent between experiments. That is, when 10 cameras are used, the first 5 of these cameras are the ones used in the 5 camera trial. The same is true for the images used. Specifically, in the table, the training dataset used for any trial is a subset of the training dataset for the trial to the right and below it on the table. Although cameras were chosen randomly, to allow for reproducibility, the list of cameras and images used is provided in the appendix (supplementary material). In the GAN cases, the unlabeled image data which is used is only from the cameras which are in the labeled set for that trial (i.e., no cameras are included which are not included in that training set). In all the training cases listed, the networks are trained for 7000 epochs.

We tested the trained models using a varying number of cameras with a varying number of images. For each case, training using the CNN (discriminator) alone and training with the GAN is compared. These results can be seen in Tables 8.1 and 8.2. For each experiment, the number of cameras is given along with the number of images used per camera. The data is explicitly limited so that we can experiment on how much data is needed to train the system to different levels of accuracy and to compare how much data is needed with the generator to how much is needed without it. The numbers of images and cameras used are referring to the training dataset only. In all trials, the entirety of both testing datasets is used. Table 8.1 shows the results on the randomly chosen test dataset and Table 8.2 are the results on the original WorldExpo'10 test dataset. Again, there is no overlap of cameras between the test datasets and the training datasets (or between the two test datasets).

When trained using the entire database of available data, the CNN and GAN reach an accuracy of 11.1 and 14.2 for the random test dataset, and 19.7 and 23.9 for the original test dataset. Training using the entire dataset, the CNN performs better. The CNN outperforming the GAN, in this case, is likely due to the GAN introducing a bias and the labeled data samples for training CNN is already sufficient.

While these early *explicit model* results provided level of improvement, the fluctuation of

Random Test Dataset Error						
Number of training cameras		Number of training images per camera				
		1	3	5	10	20
1	CNN	93.2	116.8	131.5	25.2	32.0
	GAN	134.0	70.1	37.5	32.4	19.9
3	CNN	46.8	29.1	17.4	25.0	21.7
	GAN	26.1	28.0	28.8	22.4	19.9
5	CNN	31.3	29.2	48.9	13.5	
	GAN	22.2	24.4	27.9	18.2	
10	CNN	31.0	17.8	17.4		
	GAN	29.4	24.3	17.5		
20	CNN	26.1	25.8			
	GAN	22.7	24.1			

Table 8.1: A table of the mean absolute error when the network is trained with varying amounts of data with or without the generator. For each experiment, the number of cameras is given along with the number of images used per camera. Additionally, it is shown whether the GAN or the plain CNN is used. The test dataset is the same for every case

the percentage of improvement was greater than the improvement itself. This makes these early results difficult to claim as successful. As previously noted, the explicit approach to the semi-supervised regression GAN has several potential pitfalls, which seem to be encountered here. Specifically, the error in the predictions above come primarily from under predicting the number of people in an image. This is the expected bias given the explicit training goals used.

8.3.2 SR-GAN experiments

The above early experiments led to the development of the SR-GAN to overcome the explicit model limitations. These later experiments were performed on different datasets and with different network architectures from the explicit model to keep up with the latest

Original WorldExpo'10 Test Dataset Error						
Number of training cameras		Number of training images per camera				
		1	3	5	10	20
1	CNN	65.8	168.5	189.2	32.8	45.9
	GAN	169.5	74.3	53.5	36.0	27.9
3	CNN	73.3	60.3	35.0	29.6	30.2
	GAN	112.6	36.4	30.5	32.3	32.9
5	CNN	58.0	55.5	68.1	23.7	
	GAN	33.6	40.1	42.8	30.2	
10	CNN	52.3	25.7	24.6		
	GAN	40.0	31.5	26.8		
20	CNN	46.6	38.7			
	GAN	39.5	32.7			

Table 8.2: A table of the mean absolute error when the network is trained with varying amounts of data with or without the generator. For each experiment, the number of cameras is given along with the number of images used per camera. Additionally, it is shown whether the GAN or the plain CNN is used. The test dataset is the same for every case

state-of-the-art implementations.

Idrees et al. [50] showed that a vanilla DenseNet [51] outperformed many application-specific networks for crowd counting. Though Idrees et al. [50] then provides an application-specific version of DenseNet, we chose to use the vanilla version of DenseNet201 as the discriminator in our experiments, which is shown in Figure 8.2. The full details of the network as given in Table 8.3. This is done to avoid application specific nuances that distract from the main focus of our work, while still providing a network comparable to the state-of-the-art in terms of accuracy. For the generator, we use the same DCGAN generator architecture as was used in our age and steering angle experiments.

For our experiments, we used the UCF-QNRF dataset [50]. This dataset contains 1535 total images split as 1201 training images and 334 testing images. These training and testing distributions are provided by the dataset provider. The dataset contains 1,251,642 total

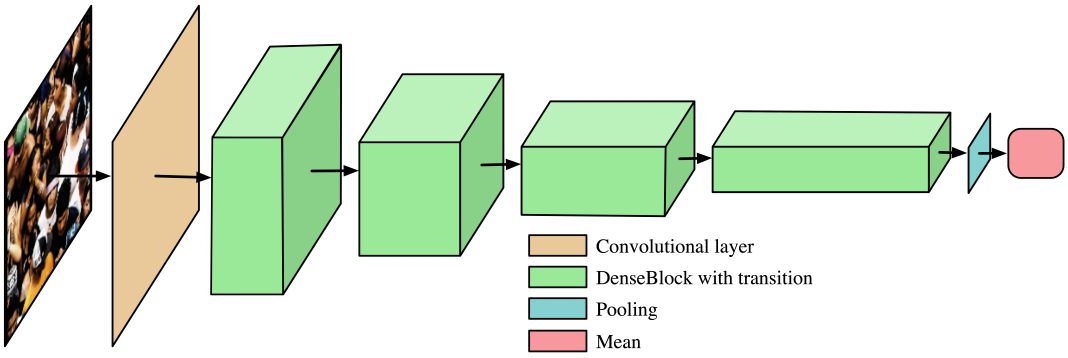


Figure 8.2: A conceptual overview of the DenseNet architecture [51] which is used as the discriminator network in the SR-GAN crowd counting set of experiments.

Layers	Output Size	Filter Types
Convolution	112×112	7×7 conv, stride 2
Pooling	56×56	3×3 max pool, stride 2
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv
	28×28	2×2 average pool, stride 2
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv
	14×14	2×2 average pool, stride 2
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Transition Layer (3)	14×14	1×1 conv
	7×7	2×2 average pool, stride 2
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$
Regression Layer	1×1	7×7 average pool

Table 8.3: The discriminator architecture based on DenseNet201. The growth rate for the filters is $k = 32$ (which defines the number of channels per DenseNet201 specifications). Each convolution is followed by a leaky ReLU, except the final convolution.

Figure 8.3: Typical examples of crowd scene images from the UCF-QNRF dataset with various levels of densities, illuminations, and distributions.

headcounts, with a median of 425 (per image), and a mean of 815.4. To the best of our knowledge, the UCF-QNRF dataset is currently the largest dataset in terms of the number of head counts. With a minimum of 49 and a maximum of 12,865, the dataset contains an enormous variety of crowd density levels. Furthermore, the camera perspective angles range drastically from near parallel to ground level to nearly perpendicular with the ground. Lighting conditions, environmental scenes, pixel size of individuals, and levels of occlusion are similarly widely varied. Typical example images from the dataset are shown in Figure 8.3. The UCF-QNRF dataset provides a challenging and extensive dataset for dense crowd counting.

Images in the UCF-QNRF vary widely in resolutions. Our network accepts image patches of 224×224. The network is trained to predict the number of headcounts within random training image patches. During the test phase, an overlapping sliding window is used to make a prediction of the number headcounts in each patch of the image. Overlapping values are averaged, and the final prediction for an image under is given by the sum of these values. Examples of the patches, both real and fake, used/generated during training are shown in Figure 8.4.

In all experiments, the network architecture of the CNN and the discriminator are identical.

(a) Fake crowd counting images. (b) Real crowd counting images.

Figure 8.4: Examples of real and fake images used/generated during training. We note that our approach is not intended to produce realistic looking images, and the fake images are only included for insight.

Notably, these are the only portions of the network used during the evaluation of the test data (the generator is only used during training). This emphasizes the value gained by the SR-GAN training method rather than the architecture of the inference network.

The first set of experiments uses a limited set of labeled training images. The unlabeled training images consist of any available training image not included in the labeled set, resulting in a number of unlabeled images being 1201 minus the number of labeled images. The results from this set of experiments are shown in Table 8.4.

We see from this set of experiments that the SR-GAN significantly outperforms the CNN in every case for MAE and RMSE. NAE results are less consistent. For MAE, the SR-GAN often outperforms the CNN even when it uses two or four times as much data. For example, the SR-GAN using 80 labeled examples significantly outperforms the CNN using 320 labeled examples. In some cases, the difference is less significant, such as when 40 examples were used. However, the model with the simple addition of the generator and the losses to the

Examples		MAE		RMSE		NAE	
Labeled	Unlabeled	CNN	GAN	CNN	GAN	CNN	GAN
10	1191	422.9	298.7	705.2	501.8	0.5369	0.5338
20	1181	390.1	288.0	643.6	473.5	0.7076	0.4994
40	1161	285.0	255.0	487.2	388.3	0.4823	0.5296
80	1121	260.2	196.4	432.4	305.7	0.4853	0.3434
160	1041	237.4	193.2	385.4	289.5	0.4735	0.3887
320	881	227.6	186.2	362.4	294.0	0.4228	0.3522

Table 8.4: Results using varying levels of labeled training examples and using all remaining training examples as unlabeled data. In each experiment, the CNN and the discriminator network architectures are identical.

Examples		MAE		RMSE		NAE	
Labeled	Unlabeled	CNN	GAN	CNN	GAN	CNN	GAN
10	100	422.9	331.8	705.2	530.9	0.5369	0.5311
20	200	390.1	295.8	643.6	456.1	0.7076	0.6240
30	300	323.3	278.4	525.2	414.0	0.5878	0.6398
40	400	285.0	258.4	487.2	416.4	0.4823	0.4592
50	500	277.4	228.1	489.2	372.9	0.4881	0.4682

Table 8.5: Results using varying levels of labeled training examples and ten times as many unlabeled examples. In each experiment, the CNN and the discriminator network architectures are identical.

CNN always outperforms the ordinary CNN. Similarly, the RMSE of the SR-GAN always outperforms the CNN using the same amount of data. These results mirror the MAE case, with the SR-GAN with 80 labeled examples once again outperforming the CNN using 320 labeled examples.

The second set of experiments consists of using both a limited number of labeled examples and a limited number of unlabeled examples. In particular, each experiment uses ten times as many unlabeled examples as labeled examples. The results from this set of experiments are shown in Table 8.5. The readers may compare Table 8.4 and Table 8.5 for the performance of under the same numbers of labeled examples (e.g., 10, 20, and 40) but with different numbers

of unlabeled examples (e.g., 1201-10 compared to 100, 1201-20 compared to 200, 1201-40 compared to 400). It should be noted, the greatest number of labeled examples used in this set of experiments is far smaller than in the previous set of experiments. This is due to the limited total dataset size (large numbers of labeled examples would not have sufficient proportional amounts of unlabeled examples to train with).

Once again, this set of experiments clearly show the advantage of adding the generator and SR-GAN to the original CNN. In every case, the MAE and RMSE of the SR-GAN significantly outperform the CNN, often even versions of the CNN with more data. For example, the SR-GAN with 20 examples achieves an MAE which outperforms the CNN with 30, 40, and 50 labeled examples.

SR-GAN challenges with density map predictions

Notable from the above experiments is a lack of the density map prediction commonly used within dense crowd counting. Unfortunately, this is due to a failure to successfully train the SR-GAN using the multi-target goal with both count prediction and density map prediction. Initially, this was interpreted as being due to a lack of effective training gradients provided by the density map labeling scheme. In Chapter 9, we discuss a labeling scheme which leads to an improved training configuration and results in higher predictive accuracies for state-of-the-art crowd counting models. Despite providing a significant advancement in the area of crowd counting, this method did not result in a successful SR-GAN using the map prediction methods. This issue is discussed in more detail in Section 9.5.

Chapter 9

Crowd label topology and upsampling

Many systems have been proposed for crowd counting purposes, with most recent state-of-the-art methods being based on convolutional neural networks (CNNs). To the best of our knowledge, every CNN-based dense crowd counting approach in recent years relies on using a density map of individuals, primarily with a Gaussian-based distribution of density values centered on individuals labeled in the ground truth images. Often, these density maps are generated with the Gaussian distribution kernel sizes being dependent on a k-Nearest Neighbor (kNN) distance to other individuals [52]. In this chapter, we explain how this generally accepted density map labeling is lacking and how an alternative inverse kNN (ikNN) labeling scheme, which does not explicitly represent crowd density, provides improved counting accuracy. We show how a single ikNN map provides information similar to the accumulation of many density maps with different Gaussian spreads, in a form which is better suited for neural network training. This labeling provides a significant gradient spatially across the entire label while still providing precise location information of individual pedestrians (with the only exception being exactly overlapping head labelings). We show that by simply replacing density map training in an existing state-of-the-art network with our ikNN map

training, the testing accuracy of the network improves. This is the first major contribution of this chapter.

Additionally, coupling multi-scale upsampling with densely connected convolutional networks [51] and the proposed ikNN mapping, we provide a new network structure, MUD-ikNN, which performs favorably compared to existing state-of-the-art methods. This network uses multi-scale upsampling with transposed convolutions [53] to make effective use of the full ground truth label, particularly with respect to our ikNN labeling scheme. The transposed convolutions are used to spatially upsample intermediate feature maps to the ground truth label map size for comparison. This approach provides several benefits. First, it allows the features of any layer to be used in the full map comparison, where many existing methods require a special network branch for this comparison. Notably, this upsampling, comparison, and following regression module can be used at any point in any CNN, with the only change being the parameters of the transposed convolution. This makes the module useful not only in our specific network structure but also applicable in future state-of-the-art, general-purpose CNNs. Second, as this allows features which have passed through different levels of convolutions to be compared to the ground truth label map, this intrinsically provides a multi-scale comparison without any dedicated additional network branches, thus preventing redundant parameters which occur in separate branches. Third, because the transposed convolution can provide any amount of upsampling (with the features being used to specify the upsampling transformation), the upsampled size can be the full ground truth label size. In contrast, most existing works used a severely reduced size label map for comparison. These reduced sizes remove potentially useful training information. Although some recent works use full-size labels, they require specially crafted network architectures to accomplish this comparison. The proposed upsampling structure can easily be added to most networks, including widely-used general-purpose networks, such as DenseNet. This proposed network structure is the second major contribution of the chapter.

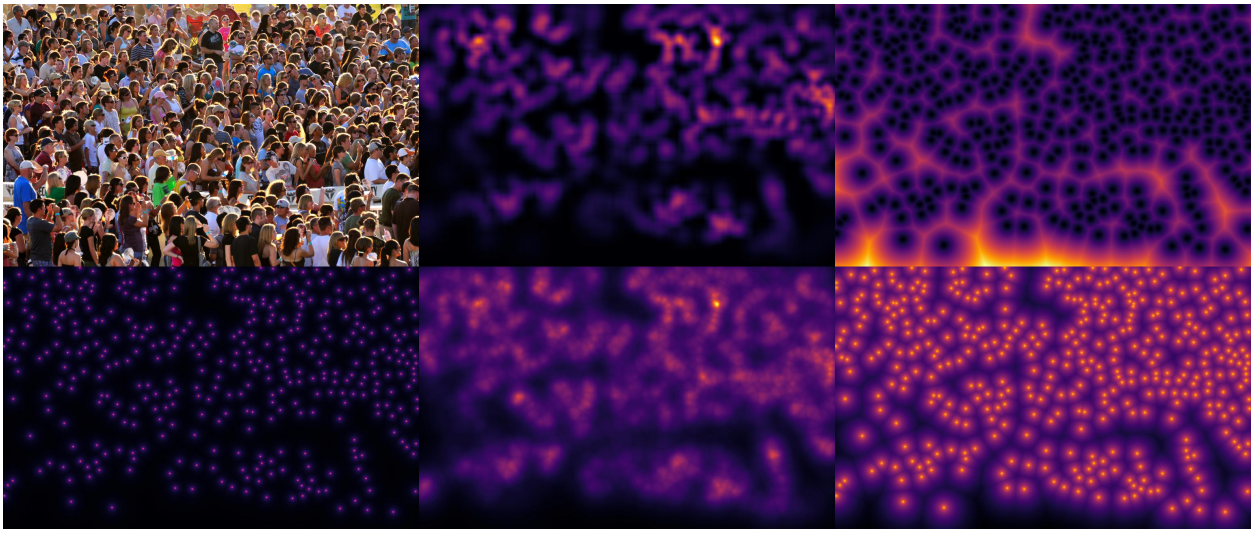


Figure 9.1: An example of a crowd image and various kinds of labelings. From left to right, on top, there is the original image, the density map, the kNN map with $k = 1$. On bottom, the inverse kNN map with $k = 1$, the inverse kNN map with $k = 3$, and the inverse kNN map with $k = 1$ shown with a log scaling (for read insight only). Note, in the case of the density map, any values a significant distance from a head labeling are very small. This is in contrast to the inverse kNN map, which has a significant gradient even a significant distance from a head position.

Importantly, these contributions are largely complementary to, rather than alternatives to, existing approaches. Most approaches can easily replace their density label comparison with the proposed ikNN map comparison and upsampling map module, with little to no modification of the rest of their method or network architecture. As the ikNN label does not sum to the count, the ikNN label and map module should go hand-in-hand.

9.1 Related Work

Many works use explicit detection of individuals to count pedestrians [54, 55, 56]. However, as the number of people in a single image increase and a scene becomes crowded, these explicit detection methods become limited by occlusion effects. Early works to solve this problem relied on global regression of the crowd count using low-level features [57, 58, 59].

While many of these methods split the image into a grid to perform a global regression on each cell, they still largely ignored detailed spatial information of pedestrian locations. [60] introduced a method of counting objects using density map regression, and this technique was shown to be particularly effective for crowd counting by [4]. Since then, to the best of our knowledge, every CNN-based crowd counting method in recent years has used density maps as a primary part of their cost function [50, 61, 62, 4, 52, 63, 64, 65, 66].

A primary advantage of the density maps is the ability to provide a useful gradient for network training over large portions of the image spatially, which helps the network identify which portion of the image contains information signifying an increase in the count. These density maps are usually modeled by representing each labeled head position with a Dirac delta function, and convolving this function with a 2D Gaussian kernel [60]. This forms a density map where the sum of the total map is equal to the total count of individuals, while the density of a single individual is spread out over several pixels of the map. The Gaussian convolution allows a smoother gradient for the loss function of the CNN to operate over, thereby allowing slightly misplaced densities to result in a lower loss than significantly misplaced densities.

In some works, the spread parameter of the Gaussian kernel is often determined using a k-nearest neighbor (kNN) distance to other head positions [52]. This provides a form of pseudo-perspective which results in pedestrians which are more distant from the camera (and therefore smaller in the image) having their density spread over a smaller number of density map pixels. While this mapping will often imperfectly map perspective (especially in sparsely crowded images), it works well in practice. Whether adaptively chosen or fixed, the Gaussian kernel size is dependent on arbitrarily chosen parameters, usually fine-tuned for a specific dataset.

In a recent work [50], the authors used multiple scales of these kNN-based, Gaussian convolved density maps to provide various levels of spatial information, from large Gaussian

kernels (allowing for a widespread training gradient) to small Gaussian kernels (allowing for precise localization of density). While this approach effectively integrates information from multiple Gaussian scales, thus providing both widespread and precise training information, the network is left with redundant structures and how the various scales are chosen is fairly ad hoc. Our alternative ikNN labeling method supersedes these multiple scale density maps by providing both a smooth training gradient and precise label locations (in the form of steep gradients) in a single label. Our new network structure utilizes a single branch CNN structure for multi-scale regression. Together with the ikNN labeling, it provides the benefits of numerous scales of these density maps.

Nearly all these CNN-based approaches use a reduced label size. Some recent works [63, 64] have begun using full resolution labels. In contrast, even to these works, we provide a generalized map module which can be added to existing network structures allowing them to take advantage of larger resolutions. The proposed network is based off the DenseNet201 [51], with the novel map module added to the end of each DenseBlock. This map module can be added to most CNN architectures with little or no modification to the original architecture.

Our ikNN mapping is somewhat related to a distance transform, which has been used for counting in other applications [67]. However, the distance transform is analogous to a kNN map, rather than our ikNN. To our knowledge, neither the distance transform nor a method analogous to our ikNN labeling has been used for dense crowd counting.

9.2 Inverse k-Nearest Neighbor Map Labeling

We propose using full image size ikNN maps as an alternative labeling scheme from the commonly used density map explained in Section 9.1. Formally, the commonly used density

map [50, 61, 62, 4, 52] is provided by,

$$D(\boldsymbol{x}, f(\cdot)) = \sum_{h=1}^{H} \frac{1}{\sqrt{2\pi} f(\sigma_h)} \exp\left(-\frac{(x - x_h)^2 + (y - y_h)^2}{2f(\sigma_h)^2}\right), \tag{9.1}$$

where H is the total number of head positions for the example image, σ_h is a size determined for each head position (x_h, y_h) using the kNN distance to other heads positions (a fixed size is also often used), and f is a manually determined function for scaling σ_h to provide a Gaussian kernel size. For simplicity, in our work we define f as a simple scalar function given by $f(\sigma_h) = \beta\sigma_h$, with β being a hand-picked scalar. Though they both apply to head positions, the use of kNN for σ_h in the density map is not to be confused with the full kNN map used in our method, which is defined by,

$$K(\boldsymbol{x}, k) = \frac{1}{k} \sum \min_{k} \left(\sqrt{(x - x_h)^2 + (y - y_h)^2}, \forall \boldsymbol{h} \in \mathcal{H}\right), \tag{9.2}$$

where $\mathcal{H}$ is the list of all head positions. In other words, the kNN distance from each pixel, (x, y), to each head position, (x_h, y_h), is calculated.

To produce the inverse kNN (ikNN) map, we use,

$$M = \frac{1}{K(\boldsymbol{x}, k) + 1}, \tag{9.3}$$

where M is the resulting ikNN map, with the addition and inverse being applied element-wise.

To understand the advantage of an ikNN map over a density map, we can consider taking the generation of density maps to extremes with regard to the spread parameter of the Gaussian kernel provided by f. A similar explanation is illustrated in Figure 9.2. At one extreme, is a Gaussian kernel with zero spread. Here the delta function remains unchanged, which in practical terms translates to a density map where the density for each pedestrian is fully residing on a single pixel. When the difference between the true and predicted density

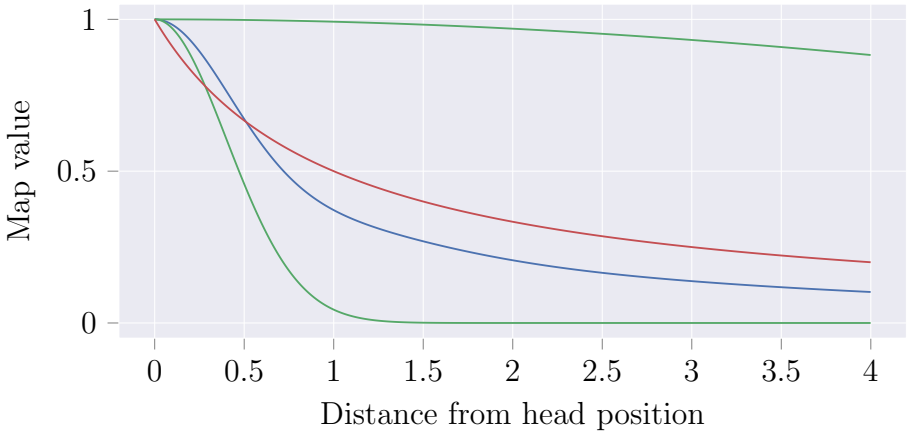


Figure 9.2: A comparison of the values of map labeling schemes with respect to the distance from an individual head position (normalized for comparison). Two Gaussians are shown in green. The narrow Gaussian provides a precise location of the head labeling. However, it provides little training information as the distance from the head increases. The wide Gaussian provides training information at a distance, but gives an imprecise location of the head position, resulting in low training information near the correct answer. The blue line shows a composite of several Gaussians with spread parameters between those of the two extremes ([50] uses 3 Gaussian spreads in their work). This provides both precise and distant training losses. Our approach of the ikNN map shown in red (with $k = 1$) approaches a map function with a shape similar to the integral on the spread parameter of all Gaussians for a spread parameter range from 0 to some constant. Additionally, our method provides both the precise and distant gradient training information in a single map label. Also notable, is that even the large Gaussian shown here approaches near zero much sooner than the ikNN map value.

maps is used to calculate a training loss, the network predicting density 1 pixel away from the correct labeling is considered just as incorrect as 10 pixels away from the correct labeling. Obviously, this is not desired, as it both creates a discontinuous training gradient, and the training process is intolerant to minor spatial labeling deviations. The other extreme is a very large Gaussian spread. This results in inexact spatial information of the location of the density. At the extreme, this provides no benefit over a global regression, which is the primary purpose for using a density map in the first place. Any intermediate Gaussian spread has an intermediate degree of both these problems. Using multiple scales of Gaussian spread, [50] tries to obtain the advantage of both sides. However, the size of the scales and the

number of scales are then arbitrary and hard to determine.

In contrast, a single ikNN map provides a substantial spatial gradient everywhere while still providing steep gradients in the exact locations of individual pedestrians. An example of our ikNN map compared with a corresponding density map labeling can be seen in Figure 9.1. Notably, [50] uses 3 density maps with different Gaussian spread parameters, with the Gaussian spread being determined by the kNN distance to other head positions multiplied by one of the 3 spread parameters. We note that for a single head position, all Gaussian distributions integrated over the spread parameter from 0 to some constant α results in a form of the incomplete gamma function. This function has a cusp around the center of the Gaussians. Similarly, the inverse of the kNN map also forms a cusp at the head position and results in similar gradients of loss given misplaced density/distance values as the spread integrated Gaussian function. In our experiments, we found that an inverse kNN map outperformed density maps with ideally selected spread parameters (and further outperformed general use selected spread parameters).

In one experiment, we use [50]'s network architecture, which utilized DenseBlocks [51] as the basis, but we replace the density maps with ikNN maps and show there is an improvement in the prediction's mean absolute error. This demonstrates the direct improvement of our ikNN method on an existing state-of-the-art network. Note, the regression module from ikNN map to count is then also required to convert from the ikNN map to a count. The difference in error between the original approach in [50] and the network in [50] with our ikNN maps, though improved, is relatively small. We suspect this is because the density maps (or ikNN maps) used during training are downsampled to a size of 28x28 (where the original images and corresponding labels are 224x224). This severe downsampling results in more binning of pixel information, and this seems to reduce the importance of which system is used to generate that label. At the extreme case, when downsampled to a single value, both approaches would only give the global count in the patch (where the ikNN map gives

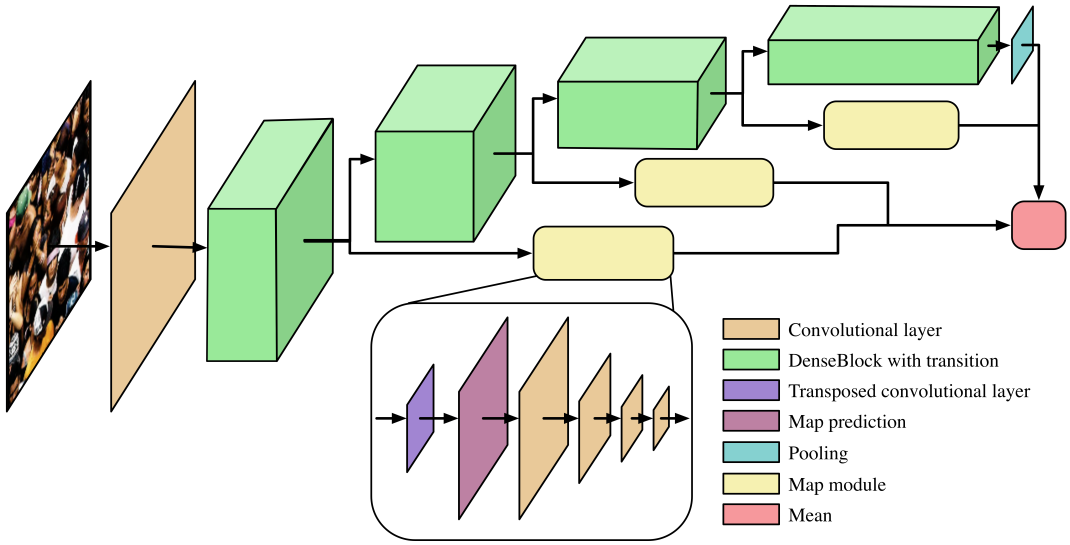


Figure 9.3: A diagram of the proposed network architecture MUD-ikNN: multiscale regression with DenseBlocks and ikNN mapping. Best viewed in color.

the inverse of the average distance from a pixel to a head labeling which can be translated to an approximate count). This downsampling is a consequence of the network structure only permitting labels of the same spatial size as the output of the DenseBlocks. Our network (which is described below) remedies this through transposed convolutions, allowing for the use of the full-size labels.

9.3 MUD-ikNN: A New Network Architecture

We propose a new network structure, MUD-ikNN, with both multi-scale upsampling using DenseBlocks [51] and our ikNN mapping scheme. We show that the new MUD-ikNN structure performs favorably compared with existing state-of-the-art networks. In addition to the use of ikNN maps playing a central role, we also demonstrate how features with any spatial size can contribute in the prediction of ikNN maps and counts through the use of transposed convolutions. This allows features of various scales from throughout the network to be used for the prediction of the crowd.

The proposed MUD-ikNN network structure is shown in Figure 9.3. Our network uses the DenseBlock structures from DenseNet201 [51] in their entirety. DenseNet has been shown to be widely applicable to various problems. The output of each DenseBlock (plus transition layer) is used as the input to the following DenseBlock, just as it is in DenseNet201. However, each of these outputs is also passed to a transposed convolutional layer (excluding the final DenseBlock output). These transposed convolutions are given a stride and kernel size such that output is the size of the ikNN map, and no spatial input dimensions have their output overlap in the produced ikNN map. This form of upsampling allows the feature depth dimensions to contribute to the gradients of the map values in the predicted ikNN map. Both the stride and kernel size of the transposed convolutions of our network are 8, 16, and 32.

The ikNN map generated at each level is individually compared against the ground truth ikNN map, each producing a loss which is then summed,

$$\mathcal{L}_m = \sum_j \text{MSE}(\hat{M}_j, M_j) \tag{9.4}$$

where j is the index of the DenseBlock that the output came from, M is the ikNN map labeling, and $\hat{M}$ is the predicted map labeling.

Each ikNN map is then also used as the input to a small regression module. This module is a series of small convolutional layers, shown in the inset of Figure 9.3. The sizes of these layers are specified in Table 9.1. The regression module then has a singleton output, corresponding to the predicted crowd count.

The mean of all predicted crowd counts from the regression modules, three in Figure 9.3, and the output of the final DenseBlock is used as the final count prediction.

$$\mathcal{L}_c = \text{MSE}\left(\frac{\hat{C}_{end} + \sum_{j=1}^{m} \hat{C}_j}{m+1}, C\right) \tag{9.5}$$

Layer	Output size	Filter
Input from DenseBlock	128x28x28 256x14x14 896x7x7	
Transposed convolution	1x224x224 (map prediction)	(8,16,32)x(8,16,32) stride=(8,16,32)
Convolution	8x112x112	2x2 stride=2
Convolution	16x56x56	2x2 stride=2
Convolution	32x28x28	2x2 stride=2
Convolution	1x1x1	28x28

Table 9.1: A specification of the map module layers. This module is used at 3 points throughout our network as shown in Figure 9.3, so the initial input size varies. However, the transposed convolution always produces a predicted map label which is uniform size (1x224x224).

with C being the ground truth count, $\hat{C}_{end}$ being the regression count output by the final DenseBlock, and $\hat{C}_j$ being the count from the jth map regression module ($j = 1, 2, ..., m; m = 3$ in Figure 9.3). This results in a total loss given by $\mathcal{L} = \mathcal{L}_m + \mathcal{L}_c$.

This approach has multiple benefits. First, if an appropriately sized stride and kernel size are specified, the transposed convolutional layer followed by ikNN map prediction to regression module can accept any sized input. This means this module of the network is very generalizable and can be applied to any CNN structure at any point in the network. For example, an additional DenseBlock could be added to either end of the DenseNet, and another of these map modules could be attached. Second, each ikNN map is individually trained to improve the prediction at that layer, which provides a form of intermediate supervision, easing the process of training earlier layers in the network. At the same time, the final count is based on the mean values of the regression modules. This means that if any individual regression module produces more accurate results, its results can individually be weighted as being more important to the final prediction.

We note that the multiple Gaussian approach by [50] has some drawbacks. The spread of the Gaussians, as well as the number of different density maps, is somewhat arbitrary. Additionally, without upsampling, a separate network branch is required to maintain spatial resolution. This results in redundant network parameters and a final count predictor which is largely unconnected to the map prediction optimization goal. Our upsampling approach allows the main network to retain a single primary branch and connects all the optimization goals tightly to this branch.

The input to the network is 224×224 image patches. The ikNN maps (or density maps) use the same size patches. Each map regression module contains the layers specified in Table 9.1. At evaluation time, a sliding window with a step size of 128 was used for each patch of the test images, with overlapping predictions averaged.

Network code and hyperparameters can be found at `https://github.com/golmschenk/sr-gan`.

9.4 Experimental Results

9.4.1 Evaluation metrics

For each dataset that we evaluated our method on, we provide the mean absolute error (MAE), normalized absolute error (NAE), and root mean squared error (RMSE). These are given by the following equations:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{C}_i - C_i\right| \tag{9.6}$$

$$\text{NAE} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|\hat{C}_i - C_i\right|}{C_i} \tag{9.7}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{C}_i - C_i)^2} \tag{9.8}$$

In the first set of experiments, we demonstrate the improvement of the ikNN labeling scheme compared to the density labeling scheme. We trained our network using various density maps produced with different Gaussian spread parameters, β (as described in Section 9.2), and compared these results to the network using ikNN maps with varying k. We also analyze the advantage of upsampling the label for both density and ikNN maps. In the second set of experiments, we provide comparisons to the state-of-the-art on standard crowd counting datasets. In these comparisons, the best ikNN map and density map from the first set of experiments is used. Most works provide their MAE and RMSE results. [50] provided the additional metric of NAE. Though this result is not available for many of the datasets, we provide our own NAE on these datasets for future works to refer to. The most directly relevant work, [50], has only provided their results for their latest dataset, UCF-QNRF. As such, their results only appear in regard to that dataset. Finally, we offer a general analysis of the results using our ikNN maps and upsampling approaches. General statistics about the datasets used in our experiments is shown in Table 9.2.

9.4.2 Impact of labeling approach and upsampling

Density maps vs ikNN maps

We used the ShanghaiTech dataset [52] part A for this analysis. The results of these tests are shown in Table 9.3. The density maps provide a curve, where too large and too small of spreads perform worse than an intermediate value. Even when choosing the best value (where $\beta = 0.3$), which needs to manually determined, the i1NN label significantly outperforms the density label.

Dataset	Images	Total count	Mean count	Max count	Average resolution
UCF-QNRF	1535	1,251,642	815	12,865	2013×2902
ShanghaiTech Part A	482	241,677	501	3139	589×868
ShanghaiTech Part B	716	88,488	123.6	578	768×1024
UCF-CC-50	50	63,974	1279	4633	2101×2888

Table 9.2: General statistics for the tested datasets.

Included in the table are experiments, in the fashion of [50], with density maps using 3 different β values. Here β_1 denotes the spread parameter used as the label map for the first map module, while β_2 and β_3 are for the second and third modules. Contrary to [50]'s findings, we only gained a benefit from 3 density labels when the first output had the smallest spread parameter. Even then, the gain was minimal. Upon inspection of the weights produced by the network from the map to the count prediction, the network reduces the predictions from the non-optimal β maps to near zero and relies solely on the optimal map (resulting in a reduced accuracy compared to using the optimal map for each map module).

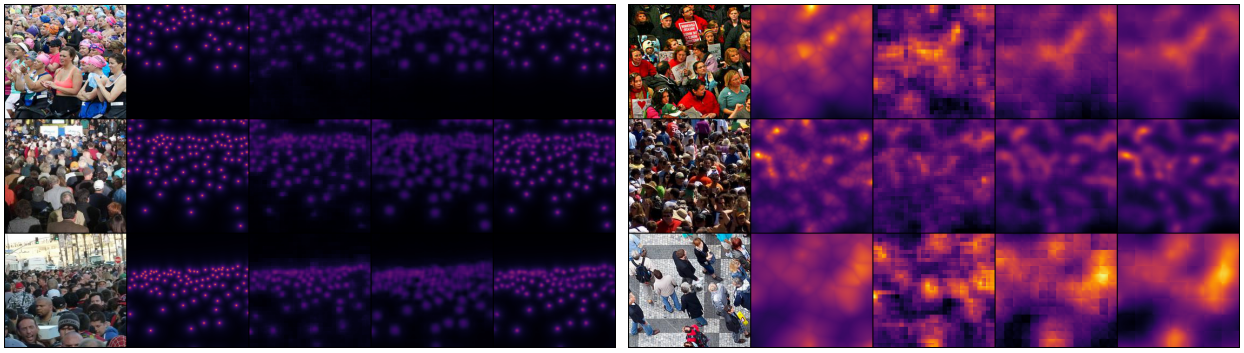
With varying k, we find that an increased k results in lower accuracy. This is likely due to the loss of precision in the location of an individual. The most direct explanation for this can be seen in the case of $k = 2$. Every pixel on the line between two nearest head positions would have the same map value, thus losing the precision of an individual location.

Upsampling analysis

Most existing works use a density map with a reduced size label for testing and training. Those that use the full label resolution design specific network architectures for the high-resolution labels. Our map module avoids this constraint by upsampling the label using a trained transposed convolution, which can be integrated into most existing architectures. Using the ShanghaiTech part A dataset, we tested our network using various label resolutions to determine the impact on the predictive abilities of the network. These results can be seen in Table 9.3. Experiments without no label resolution given are 224×224. From these results,

Method	MAE	NAE	RMSE
MUD-densityβ0.3 28x28	79.0	0.209	120.5
MUD-densityβ0.3 56x56	74.8	0.181	121.0
MUD-densityβ0.3 112x112	73.3	0.176	119.1
MUD-i1NN 28x28	75.8	0.180	120.3
MUD-i1NN 56x56	72.7	0.181	117.4
MUD-i1NN 112x112	70.8	0.166	117.0
MUD-densityβ0.05	84.5	0.233	139.9
MUD-densityβ0.1	76.8	0.189	120.3
MUD-densityβ0.2	75.3	0.175	124.2
MUD-densityβ0.3	72.7	0.174	120.4
MUD-densityβ0.4	75.7	0.176	130.5
MUD-densityβ0.5	76.3	0.182	130.0
MUD-density $\beta_1$0.5,$\beta_2$0.3,$\beta_3$0	78.5	0.205	124.2
MUD-density $\beta_1$0.5,$\beta_2$0.3,$\beta_3$0.05	77.8	0.207	124.9
MUD-density $\beta_1$0.4,$\beta_2$0.2,$\beta_3$0.1	76.7	0.202	122.7
MUD-density $\beta_1$0.1,$\beta_2$0.2,$\beta_3$0.4	75.1	0.191	119.0
MUD-density $\beta_1$0.2,$\beta_2$0.3,$\beta_3$0.4	76.0	0.196	122.1
MUD-i1NN	68.0	0.162	117.7
MUD-i2NN	68.8	0.168	109.0
MUD-i3NN	69.8	0.169	110.7
MUD-i4NN	72.2	0.173	116.0
MUD-i5NN	74.0	0.182	119.1
MUD-i6NN	76.2	0.188	120.9

Table 9.3: Results using density maps vs ikNN maps with varying k and β, as well as the various upsampling resolutions on the ShanghaiTech Part A dataset. If a resolution is not shown, it is the default 224×224. Multiple β correspond to a different Gaussian density map for each of the 3 map module comparisons.



(a) i1NN predictions. (b) i3NN predictions.

Figure 9.4: A small sample of patch predictions for map labels. In each subfigure, from left to right is the original image patch, the ground truth label, and the patches from the three map modules in order through the network.

it is clear that the higher resolution leads to higher accuracy. Note, this results in a minor change to the map module structure, as the final convolution kernel needs to match the remaining spatial dimension. A set of predicted ikNN map labels can be seen in Figure 9.4, where a grid pattern due to the upsampling can be identified in some cases.

9.4.3 Comparisons on standard datasets

The following demonstrates our network's predictive capabilities on various datasets, compared to various state-of-the-art methods. Again, we note that our improvements are expected to complementary to the existing approaches, rather than alternatives.

For these experiments, we used the best k, 1, and best β, 0.3, from the first set of experiments.

The first dataset we evaluated our approach on is the UCF-QNRF dataset [50]. The results of our MUD-ikNN network compared with other state-of-the-art networks are shown in Table 9.4. Our network significantly outperforms the existing methods. Along with a comparison of our complete method compared with the state-of-the-art, we compare with [50]'s network, but replace their density map predictions and summing to count with our

Method	MAE	NAE	RMSE
Idrees et al. [68]	315	0.63	508
MCNN [52]	277	0.55	426
Encoder-Decoder [69]	270	0.56	478
CMTL [62]	252	0.54	514
SwitchCNN [61]	228	0.44	445
Resnet101 [70]	190	0.50	227
DenseNet201 [51]	163	0.40	226
Idrees et al. [50]	132	0.26	191
[50] with i1NN maps	122	0.252	195
MUD-i1NN	104	0.209	172

Table 9.4: Results on the UCF-QNRF dataset.

Method	MAE	NAE	RMSE
ACSCP [63]	75.7	-	102.7
D-ConvNet-v1 [66]	73.5	-	112.3
ic-CNN [65]	68.5	-	116.2
CSRNet [64]	68.2	-	115.0
MUD-densityβ0.3	72.7	0.174	120.4
MUD-i1NN	68.0	0.162	117.7

Table 9.5: Results on the ShanghaiTech Part A dataset.

ikNN map prediction and regression to count. Using the ikNN maps, we see that their model sees improvement in MAE with ikNN maps, showing the effect of the ikNN mapping.

The second dataset we evaluated our approach on is the ShanghaiTech dataset [52]. The dataset is split into two parts, Part A and Part B. For both parts, we used the training and testing images as prescribed by the dataset provider. The results of our evaluation on part A are shown in Table 9.5. Our MUD-ikNN network slightly outperforms the state-of-the-art approaches on this part. The results of our evaluation on part B are shown in Table 9.6. Here our network performs on par or slightly worse than the best-performing methods.

The third dataset we evaluated our approach on is the UCF-CC-50 dataset [68]. We followed the standard evaluation metric for this dataset of a five-fold cross-evaluation. The

Method	MAE	NAE	RMSE
D-ConvNet-v1 [66]	18.7	-	26.0
ACSCP [63]	17.2	-	27.4
ic-CNN [65]	10.7	-	16.0
CSRNet [64]	10.6	-	16.0
MUD-densityβ0.3	16.6	0.130	26.9
MUD-i1NN	13.4	0.107	21.4

Table 9.6: Results on the ShanghaiTech Part B dataset.

Method	MAE	NAE	RMSE
ACSCP [63]	291.0	-	404.6
D-ConvNet-v1[66]	288.4	-	404.7
CSRNet [64]	266.1	-	397.5
ic-CNN [65]	260.9	-	365.5
MUD-densityβ0.3	246.44	0.188	348.1
MUD-i1NN	237.76	0.191	305.7

Table 9.7: Results on the UCF-CC-50 dataset.

results of our evaluation on this dataset can be seen in Table 9.7.

Overall, our network performed favorably compared with existing approaches. An advantage to our approach is that our modifications can be applied to the architectures we're comparing against. The most relevant comparison is between the ikNN version of the MUD network, and the density map version. Here, the ikNN approach always outperformed the density version. We speculate that the state-of-the-art methods we have compared with, along with other general-purpose CNNs, could be improved through the use of ikNN labels and upsampling map modules.

9.5 ikNN Maps and the SR-GAN

While the improvements demonstrated by the ikNN maps alone are a significant development, their main focus in this work was to overcome a challenge presented by the use of semi-

supervised GANs for crowd counting. In particular, the multiple goal aspect commonly used in dense crowd counting, the prediction of both a form of localization of the person density in the image and the total count. Most state-of-the-art approaches to dense crowd counting use such a map prediction as a secondary loss function to improve the accuracy of their predictions, as we have seen throughout this chapter. However, adding such a secondary goal greatly increases the complexity of training the SR-GAN.

With the map prediction, if the matching and contrasting losses are not also applied to the map prediction, the network may sacrifice accuracy in the counting goal to improve the map prediction, as there is no semi-supervised loss to contend with. Applying the semi-supervised metrics to the features of the map prediction cause additional problems, as the number of features (and therefore their weight) are much greater. Manually balancing the weights is challenging (as the magnitude of the map and count features are not intrinsically proportional) and any such manual balancing introduces a new arbitrary parameter that may not generalize well. The results of Section 5.2 provided a way to auto-balance the parameters in the SR-GANs algorithm. However, in this more complex situation, the number of parameters to weigh the losses are doubled. These parameters now affect losses from various parts in the network, with various types of losses being considered. This makes the self-balancing approach intractable.

Preliminary experiments suggested the challenge of training the SR-GAN to handle the map and count optimization goals simultaneously arose from the inconsistent training gradient of the density label scheme. The ikNN labeling method was expected to remedy this training issue. While it likely still had an impact to this effect, our preliminary experiments using the SR-GAN with the ikNN map did not produce better results than the standard DNN approach, due to the issue of loss balancing described above.

This is not to suggest that there is not a solution to this issue, but rather that this work does not present the solution. This is a line of research which should be continued. Likely,

the issue is not with the SR-GAN specifically, but the current map comparison approach in dense crowd counting. Currently, the map comparisons in dense crowd counting are given an arbitrary weight to overall optimization function (which combines both map and count goals). Adding a scalar multiplier to the map or count loss demonstrates that the optimal multiplier is not unity. As of now, the multiplier chosen is manually selected, arbitrary, and changes depending on the dataset. Should a more methodological approach be devised for the selection of this weighting, a generalized SR-GAN implementation with the new mapping topology could be more easily devised.

Chapter 10

Conclusion and Discussion

Throughout this work, we have presented a means by which to train semi-supervised GANs in a regression situation. The new SR-GAN algorithm was explained in detail. A set of optimization rules which allows for stable, consistent training when using the SR-GAN (including experiments demonstrating the importance of these rules) were given. We performed systematic experiments using the SR-GAN on the real-world applications of age estimation, driving steering angle prediction, and crowd counting, all from single images, showing the benefits of SR-GANs over existing approaches. Adding the SR-GAN generator and objectives to a CNN when unlabeled data is available almost always increases the predictive accuracy of the CNN. We believe this work demonstrates a way in which semi-supervised GANs can be applied generally to a wide range of regression problems with little or no change to the algorithm presented here. This work allows such problems to be solved using deep learning with significantly less labeled training data than was previously required.

Additionally, we have presented a new form of labeling for crowd counting data, the ikNN map. We have compared this labeling scheme to commonly accepted labeling approach for crowd counting, the density map. We show that using the ikNN map with an existing state-of-

the-art network improves the accuracy of the network compared to density map labelings. We have demonstrated the improvements gained by using increased label resolutions and provide an upsampling map module which can be generally used by other crowd counting architectures. These approaches can easily be incorporated into other crowd counting techniques, as we have incorporated them into DenseNet, which resulted in a network which performs favorably compared with the state-of-the-art.

The SR-GAN method provides the opportunity for countless future endeavors. This work has shown its generalizability, by applying the method successfully to four applications, three of which are real-world tasks. As noted in Section 1.1, there are many regression problems which DNNs are currently being employed to solve, and the set of interesting regression tasks is limitless. The first expansion of this work is simply to apply it to more tasks and to reap the benefits of its potential. The number of works which have begun to exploit GANs for semi-supervised learning is quite limited, especially in regression tasks, largely due to the lack of a general method prior to this work. Anyone of the many applications listed in Section 1.1, among others, are candidates for gaining improvement from the SR-GAN.

Additionally, this work has also only explored the use of the SR-GAN on a single layer using a single feature statistic. Specifically, the final feature layer using the mean value of each feature for the contrasting and matching metrics. Though this approach proved to be a successful one, greater benefits may come from exploring other metrics. For example, the statistics of earlier feature layers or jointly from many feature layers may provide additional benefits. Furthermore, only the mean was used as the comparing statistic in this work. Using the variance as a metric may provide additional value. Alternatively, a higher-order moment of the statistics (where mean is the first-order moment, variance the second-order moment, skewness the third-order moment, etc).

Finally, this work provides some fascinating insight into what CNNs are actually finding in their search to improve the regression value prediction. When the generator is trained

to produce a realistic-looking image, the images have a high degree of realism, often being indistinguishable to a human observer from a real image. However, when the generator is trained to improve the performance of a discriminator in the prediction accuracy task, the images from the generator appear less real. This suggests that the characteristics of an image which help the discriminator accomplish a prediction task are not the same as those that make an image appear real. This is an exciting line of inquiry which has not been explored in this work but would be of great interest to peruse, as it brings to the forefront the understanding of how a DNN currently "thinks" differently than the human mind.

Candidate's Peer Reviewed Publications

[1] Greg Olmschenk, Zhigang Zhu, and Hao Tang. "Generalizing semi-supervised generative adversarial networks to regression using feature contrasting". In: *Computer Vision and Image Understanding* (2019).

[2] Greg Olmschenk, Zhigang Zhu, and Hao Tang. "Improving Dense Crowd Counting Convolutional Neural Networks using Inverse k-Nearest Neighbor Maps and Multiscale Upsampling". In: *(Conference unlisted for blinding)*. 2019, Submitted.

[3] Greg Olmschenk, Jin Chen, Hao Tang, and Zhigang Zhu. "Dense Crowd Counting Convolutional Neural Networks with Minimal Data using Semi-Supervised Dual-Goal Generative Adversarial Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

[4] Greg Olmschenk, Hao Tang, and Zhigang Zhu. "Crowd Counting With Minimal Data Using Generative Adversarial Networks For Multiple Target Regression". In: *Applications of Computer Vision (WACV), 2018 IEEE Winter Conference on*. IEEE. 2018, pp. 1151–1159.

[5] Jonas Weiss, Gregory Olmschenk, Qiuwen Lou, Folkert Horst, and Bert Jan Offrein. "Simulation Framework and Hardware-in-the-Loop Validation for Analog Compute Accelerators". In: *Proceedings of the Conference in Cognitive Computing*. 2018.

[6] Vishnu Nair, Manjekar Budhai, Greg Olmschenk, William H Seiple, and Zhigang Zhu. "ASSIST: personalized indoor navigation via multimodal sensors and high-level semantic information". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[7] Vishnu Nair, Christina Tsangouri, Bowen Xiao, Greg Olmschenk, WH Seiple, and Z Zhu. "A Hybrid Indoor Positioning System for Blind and Visually Impaired Using Bluetooth and Google Tango". In: *Journal on technology and persons with disabilities* 6 (2018).

[8] Greg Olmschenk, Hao Tang, and Zhigang Zhu. "Pitch and Roll Camera Orientation from a Single 2D Image Using Convolutional Neural Networks". In: *Computer and Robot Vision (CRV), 2017 14th Conference on*. IEEE. 2017, pp. 261–268.

[9] Jie Gong, Cecilia Feeley, Hao Tang, Greg Olmschenk, Vishnu Nair, Zixiang Zhou, Yi Yu, Ken Yamamoto, and Zhigang Zhu. "Building smart and accessible transportation hubs with internet of things, big data analytics, and affective computing". In: *International Conference on Sustainable Infrastructure 2017*. 2017, pp. 126–138.

[10] J Gong, C Feeley, H Tang, G Olmschenk, V Nair, Z Zhou, Y Yu, K Yamamoto, and Z Zhu. "Building smart transportation hubs with internet of things to improve services to people with special needs". In: *Transportation Research Board (TRB) 96th Annual Meeting* (2017).

[11] Greg Olmschenk, Christopher Yang, Zhigang Zhu, Hanghang Tong, and William H Seiple. "Mobile crowd assisted navigation for the visually impaired". In: *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. IEEE. 2015, pp. 324–327.

[12] Wai L Khoo, Greg Olmschenk, Zhigang Zhu, and Tony Ro. "Evaluating crowd sourced navigation for the visually impaired in a virtual environment". In: *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE. 2015, pp. 431–437.

[13] Greg Olmschenk and Zhigang Zhu. "Mobile real-time single image 3D corridor reconstruction using J-Linkage". In: *Machine Vision Applications (MVA), 2015 14th IAPR International Conference on*. IEEE. 2015, pp. 94–97.

[14] Greg Olmschenk and Zhigang Zhu. "3D Hallway Modeling Using a Single Image". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 158–164.

[15] Karl Isensee, Greg Olmschenk, Lawrence Rudnick, Tracey DeLaney, Jeonghee Rho, JD Smith, William T Reach, Takashi Kozasa, and Haley Gomez. "Nucleosynthetic Layers in the Shocked Ejecta of Cassiopeia A". In: *The Astrophysical Journal* 757.2 (2012), p. 126.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[2] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. "Object recognition with gradient-based learning". In: *Shape, contour and grouping in computer vision* (1999), pp. 823–823.

[3] Samuel Dodge and Lina Karam. "A Study and Comparison of Human and Deep Learning Recognition Performance Under Visual Distortions". In: *arXiv preprint arXiv:1705.02498* (2017).

[4] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. "Cross-scene crowd counting via deep convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 833–841.

[5] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[6] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. "Deep learning for event-driven stock prediction." In: *Ijcai*. 2015, pp. 2327–2333.

[7] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *Advances in neural information processing systems*. 2014, pp. 2366–2374.

[8] Zhenxing Niu, Mo Zhou, Le Wang, Xinbo Gao, and Gang Hua. "Ordinal regression with multiple output cnn for age estimation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4920–4928.

[9] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. "Context encoders: Feature learning by inpainting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2536–2544.

[10] Iftikhar Ali, Felix Greifeneder, Jelena Stamenkovic, Maxim Neumann, and Claudia Notarnicola. "Review of machine learning approaches for biomass and soil moisture retrievals from remote sensing data". In: *Remote Sensing* 7.12 (2015), pp. 16398–16421.

[11] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. "Traffic flow prediction with big data: a deep learning approach". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 865–873.

[12] Jouni Hartikainen, Mari Seppanen, and Simo Sarkka. "State-space inference for non-linear latent force models with application to satellite orbit prediction". In: *arXiv preprint arXiv:1206.4670* (2012).

[13] Daniel L Marino, Kasun Amarasinghe, and Milos Manic. "Building energy load forecasting using deep neural networks". In: *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*. IEEE. 2016, pp. 7046–7051.

[14] S Fabbro, KA Venn, T O'Briain, S Bialek, C Kielty, F Jahandar, and S Monty. "An Application of Deep Learning in the Analysis of Stellar Spectra". In: *Monthly Notices of the Royal Astronomical Society* (2017).

[15] Tiago Prado Oliveira, Jamil Salem Barbar, and Alexsandro Santos Soares. "Computer network traffic prediction: a comparison between traditional and deep learning neural networks". In: *International Journal of Big Data Intelligence* 3.1 (2016), pp. 28–37.

[16] Max Schwarz, Hannes Schulz, and Sven Behnke. "RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 1329–1335.

[17] Lucie M Bland, BEN Collen, C David L Orme, and JON Bielby. "Predicting the conservation status of data-deficient species". In: *Conservation Biology* 29.1 (2015), pp. 250–259.

[18] Yonggang Liu and Robert H Weisberg. "Patterns of ocean current variability on the West Florida Shelf using the self-organizing map". In: *Journal of Geophysical Research: Oceans* 110.C6 (2005).

[19] Ian Goodfellow. "NIPS 2016 tutorial: Generative adversarial networks". In: *arXiv preprint arXiv:1701.00160* (2016).

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[21] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[24] Maurice Sion. "On general minimax theorems". In: *Pacific Journal of mathematics* 8.1 (1958), pp. 171–176.

[25] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. "Testing the manifold hypothesis". In: *Journal of the American Mathematical Society* 29.4 (2016), pp. 983–1049.

[26] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[27] Jost Tobias Springenberg. "Unsupervised and semi-supervised learning with categorical generative adversarial networks". In: *arXiv preprint arXiv:1511.06390* (2015).

[28] Kumar Sricharan, Raja Bala, Matthew Shreve, Hui Ding, Kumar Saketh, and Jin Sun. "Semi-supervised conditional gans". In: *arXiv preprint arXiv:1708.05789* (2017).

[29] Nasim Souly, Concetto Spampinato, and Mubarak Shah. "Semi and Weakly Supervised Semantic Segmentation Using Generative Adversarial Network". In: *arXiv preprint arXiv:1703.09695* (2017).

[30] Mehdi Rezagholiradeh and Md Akmal Haidar. "Reg-Gan: Semi-Supervised Learning Based on Generative Adversarial Networks for Regression". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 2806–2810.

[31] Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4077–4087.

[32] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. "Meta-learning for semi-supervised few-shot classification". In: *arXiv preprint arXiv:1803.00676* (2018).

[33] Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. "Metagan: An adversarial approach to few-shot learning". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2365–2374.

[34] Shabab Bazrafkan and Peter Corcoran. "Versatile Auxiliary Regressor with Generative Adversarial network (VAR+ GAN)". In: *arXiv preprint arXiv:1805.10864* (2018).

[35] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan R Salakhutdinov. "Good semi-supervised learning that requires a bad gan". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6513–6523.

[36] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[37] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of wasserstein gans". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5769–5779.

[38] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[39] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[40] Samuel A Barnett. "Convergence Problems with Generative Adversarial Networks (GANs)". In: *arXiv preprint arXiv:1806.11382* (2018).

[41] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[42] Rasmus Rothe, Radu Timofte, and Luc Van Gool. "Deep expectation of real and apparent age from a single image without facial landmarks". In: *International Journal of Computer Vision* (2016), pp. 1–14.

[43] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. "Virtual to real reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1704.03952* (2017).

[44] Sully Chen. *Sully Chen Driving Dataset*. `https://github.com/SullyChen/driving-datasets`. [Online; accessed 8-February-2019]. 2017.

[45] Ernest Cheung, Tsan Kwong Wong, Aniket Bera, Xiaogang Wang, and Dinesh Manocha. "Lcrowdv: Generating labeled videos for simulation-based crowd behavior learning". In: *European Conference on Computer Vision*. Springer. 2016, pp. 709–727.

[46] Carl Vondrick, Donald Patterson, and Deva Ramanan. "Efficiently scaling up crowdsourced video annotation". In: *International Journal of Computer Vision* 101.1 (2013), pp. 184–204.

[47] Lingke Zeng, Xiangmin Xu, Bolun Cai, Suo Qiu, and Tong Zhang. "Multi-scale Convolutional Neural Networks for Crowd Counting". In: *arXiv preprint arXiv:1702.02359* (2017).

[48] Mark Marsden, Kevin McGuinness, Suzanne Little, and Noel E O'Connor. "ResnetCrowd: A Residual Deep Learning Architecture for Crowd Counting, Violent Behaviour Detection and Crowd Density Level Classification". In: *arXiv preprint arXiv:1705.10698* (2017).

[49] Jiawen Li, Hua Yang, Lin Chen, Jingwei Li, and Cheng Zhi. "An end-to-end generative adversarial network for crowd counting under complicated scenes". In: *Broadband Multimedia Systems and Broadcasting (BMSB), 2017 IEEE International Symposium on*. IEEE. 2017, pp. 1–4.

[50] Haroon Idrees, Muhmmad Tayyab, Kishan Athrey, Dong Zhang, Somaya Al-Maadeed, Nasir Rajpoot, and Mubarak Shah. "Composition Loss for Counting, Density Map Estimation and Localization in Dense Crowds". In: *arXiv preprint arXiv:1808.01050* (2018).

[51] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely Connected Convolutional Networks." In: *CVPR*. Vol. 1. 2. 2017, p. 3.

[52] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. "Single-image crowd counting via multi-column convolutional neural network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 589–597.

[53] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. "Deconvolutional networks". In: (2010).

[54] Bo Wu and Ram Nevatia. "Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors". In: *null*. IEEE. 2005, pp. 90–97.

[55] Zhe Lin and Larry S Davis. "Shape-based human detection and segmentation via hierarchical part-template matching". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.4 (2010), pp. 604–618.

[56] Meng Wang and Xiaogang Wang. "Automatic adaptation of a generic pedestrian detector to a specific traffic scene". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 3401–3408.

[57] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. "Privacy preserving crowd monitoring: Counting people without people models or tracking". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–7.

[58] Ke Chen, Chen Change Loy, Shaogang Gong, and Tony Xiang. "Feature mining for localised crowd counting." In: *BMVC*. Vol. 1. 2. 2012, p. 3.

[59] Ke Chen, Shaogang Gong, Tao Xiang, and Chen Change Loy. "Cumulative attribute space for age and crowd density estimation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2467–2474.

[60] Victor Lempitsky and Andrew Zisserman. "Learning to count objects in images". In: *Advances in neural information processing systems*. 2010, pp. 1324–1332.

[61] Deepak Babu Sam, Shiv Surya, and R Venkatesh Babu. "Switching convolutional neural network for crowd counting". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 1. 3. 2017, p. 6.

[62] Vishwanath A Sindagi and Vishal M Patel. "Cnn-based cascaded multi-task learning of high-level prior and density estimation for crowd counting". In: *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*. IEEE. 2017, pp. 1–6.

[63] Zan Shen, Yi Xu, Bingbing Ni, Minsi Wang, Jianguo Hu, and Xiaokang Yang. "Crowd counting via adversarial cross-scale consistency pursuit". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5245–5254.

[64] Yuhong Li, Xiaofan Zhang, and Deming Chen. "Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 1091–1100.

[65] Viresh Ranjan, Hieu Le, and Minh Hoai. "Iterative crowd counting". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 270–285.

[66] Zenglin Shi, Le Zhang, Yun Liu, Xiaofeng Cao, Yangdong Ye, Ming-Ming Cheng, and Guoyan Zheng. "Crowd counting with deep negative correlation learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5382–5390.

[67] Carlos Arteta, Victor Lempitsky, and Andrew Zisserman. "Counting in the wild". In: *European conference on computer vision*. Springer. 2016, pp. 483–498.

[68] Haroon Idrees, Imran Saleemi, Cody Seibert, and Mubarak Shah. "Multi-source multi-scale counting in extremely dense crowd images". In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE. 2013, pp. 2547–2554.

[69] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.