CS230

CS 230 Final Report – Spring 2022

Avi Gupta
Department of Computer Science
Stanford University
avigupta@stanford.edu

Roshini Ravi
Department of Computer Science
Stanford University
roshini@stanford.edu

Abstract

We develop a strong classifier and a deep convolutional generative adversarial network for the Street View House Numbers Dataset. We employ a deep convolutional architecture to produce a highly accurate classifier for recognizing digits in the dataset. We apply the learnings from the classifier to create a DCGAN that generates synthetic address number images, with mixed results. Finally, we present preliminary results of attempting to classify (using our classifier) the synthetic images generated by the DCGAN.

1 Introduction

Our project focuses on a specific application of optical character recognition (OCR) – recognizing the address numbers of homes. This project is motivated by the potential of automated recognition systems to significantly reduce inefficiencies in package deliveries and other navigation applications. For example, deliveries of packages to private homes and businesses could be rendered more efficient by precise address number recognition to ensure that the package is delivered to the appropriate residence. This may be particularly salient in contexts where GPS is unavailable or insufficiently accurate. Therefore, we leverage the Street View House Numbers (SVHN) Dataset to build a classifier that can detect digits in natural images. This work could have more general implications outside of address numbers such as object detection and optical character recognition. Therefore, this classifier could also be extended to more difficult OCR problems such as recognizing English text from images. We have also developed a preliminary deep convolutional generative adversarial network (DCGAN) inspired by the learnings from developing the classifier that produces synthetic address number images from random noise. Finally, we demonstrate the robustness of the classifer and the DCGAN by feeding synthetically images generated using the DCGAN into the classifier and demonstrating that the classifier identifies the correct digit in the limited cases where there is a clearly legible digit.

2 Related work

Multi-digit-recognition is generally approached in two different ways. Traditionally, the localization, segmentation, and recognition steps are separated. This strategy is well-researched and is essentially identical to the classification task completed on the MNIST dataset. This machine learning task has been accomplished via various techniques such as multinomial logistic regression, a standard CNN, or residual learning models such as ResNet. Although this decoupled approach achieves high accuracy, making predictions with this model entails preprocessing the test data into individual digits. This becomes an increasingly intensive task as the amount of test data and the number of digits within an individual test sample increases. Marking off individual digits would require manual labeling, since automating the digit border identification is itself a complex ML problem. Therefore, the motivation for a model that can successfully identify multiple digits is high.

CS230: Deep Learning, Spring 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

(a) Original

(b) Resized

(c) Greyscale

Figure 1: Data Preprocessing Example

Some studies have used a single digit classifier and transfer learning to build final multi-digit classification model. This approach has proven less accurate than some of the others that were explored. [2] State-of-the-art performance on this task is currently 0.99% error and was achieved through the application of Sharpness Aware Minimization [6], a parameter search technique on WideResNets with ShakeShake regularization. [22] Other promising implementations focus on increasing direct connections between layers of a CNN [10] and building a CNN inspired by an RNN. [19]

Although GANs are a relatively recent innovation, their broad array of applications has catalyzed a robust body of literature we drew upon. Vanilla GANs were proposed in 2014 in "Generative Adversarial Nets," which outlines the adversarial training model, shows important theoretical results, and compares GANs to existing models.[8] Building on this work, DCGANs were proposed in 2015 as a unsupervised technique for learning image representations.[15] DCGANs introduce the innovation of removing pooling layers in a GAN, an approach inspired by a 2014 paper, "Striving for Simplicity: The All Convolutional Net."[17] This paper proposes to replace pooling layers with strided convolutions to create a all convolutional CNN. We replicated this approach in our DCGAN. Lastly, because training GANs is particularly challenging, there is also literature on training tips, including one 2016 paper co-authored by GAN creator Ian Goodfellow entitled: "Improved Techniques for Training GANs."[16]

3 Dataset and Features

We are working with the SVHN Dataset which is administered by deep learning researchers here at Stanford.[13] The dataset contains "73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data." The dataset is available in two formats (MNIST style and original images with bounding boxes), both of which we hope to investigate in our project.

For the purposes of the multi-digit-recognition model, the training data from the original SHVN dataset was first split into train and validation subsets, with a 4:1 ratio. The train data was augmented with 10% of the extra data as was the validation data. In total this constituted 46955 training examples, 26916 validation examples, and 13068 test examples. Each image in the dataset was cropped to the bounds of the individual digits in the image and then expanded by 12.5% in each direction and resized to a 64x64 size. The images were also converted to a grayscale, i.e. converted from three channels to one. The images were also normalized by dividing the float values by 255. With respect to the labels the images were assigned, the original labels which were set to be the numbers shown in the image, were split into individual digits and expanded to lists of 5 digits each where any digit that did not have a value in the original image was set to be 10. For example, the image in Figure 1 which originally had label 19 was converted to [1, 9, 10, 10, 10].

The same data preprocessing steps were applied to the images generated by the adversarial network.

An example of this preprocessing is presented in Figure 1.

4 Methods

4.1 Classifier

While there exist a plethora of different strategies and architectures for solving this problem, in implementing our base case, we focused on a model with a high performance that was also straight-

forward in its implementation. Goodfellow et al combine a series of data preprocessing steps, data augmentation, and the DistBelief implementation of deep neural networks [7]. We benefited from their general preprocessing techniques and architecture structure.

Specifically, beyond the data preprocessing enumerated in the section above, we executed the model structure shared in Figure 2.

Differences between our architecture and that proposed by Goodfellow et al, include: the activation functions we chose to apply - consistenly ReLU, wherein Goodfellow et al also used maxout units; no predictions on the length of the digit sequence; and dropout only on the final layer. We found that these changes simplified our model and its running time without drastically affecting its performance.

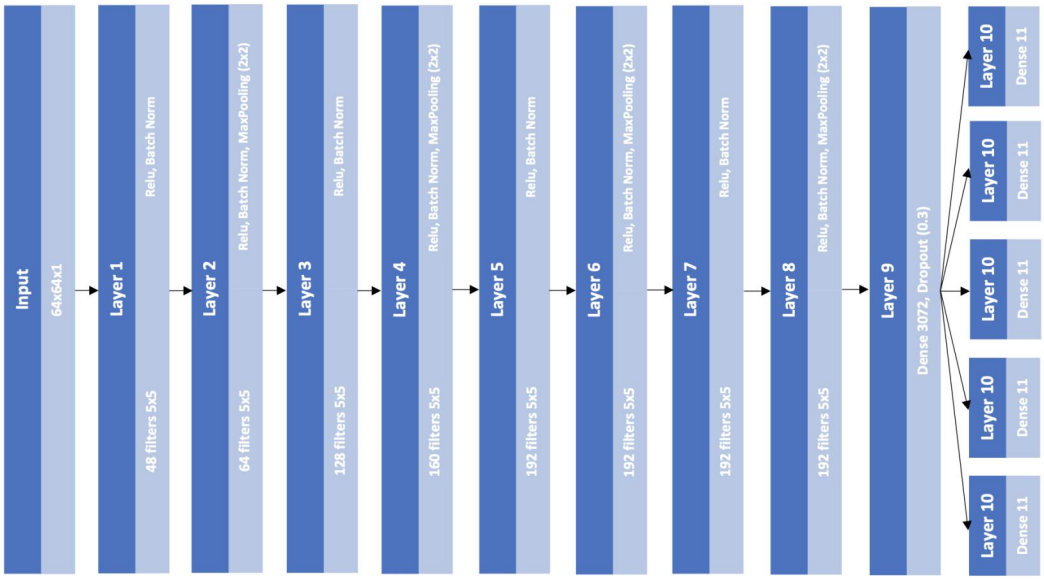

Figure 2: Model Architecture

4.2 GAN

An additional component of our project was the creation of a DCGAN to produce synthetic images mirroring those in the SVHN dataset. Our approach was inspired by the seminal 2015 paper and best practices throughout the GAN literature.[15] [16]

The discriminator architecture we ultimately settled upon consisted of four stacked convolutional layers interspersed with batch normalization and Leaky ReLU activations. A linear layer collapses the output down to a single output neuron. This was inspired by our classifier and tweaked to reflect DCGAN best practices. We employed Leaky ReLU based on the literature and on our experimentation, which suggests it is a superior choice for DCGANs in particular. This marks a contrast with the 2013 paper on GANs which employed a maxout function.[8] Moreover, the DCGAN paper suggests using batch normalization for both the discriminator and generator networks.

Our final generator used a linear encoding of the noise vector followed by repeated transposed convolutions interspersed with batch normalization and ReLU activations. We avoided pooling layers (which would typically be used in a CNN) based on the results presented in the DCGAN paper and other references.[17] Crucially, we concluded our DCGAN with a tanh activation function, as suggested in the paper. The full details of both architectures are available in the code submission accompanying this report.

4.3 Feeding GAN Images to Classifier

Using the generator network developed during DCGAN training, synthetic images were generated from random noise. They were then preprocessed in an identical manner to the images that were earlier fed into the classifier (resizing, grayscale, normalization, etc.) to produce labeled results. This

	Single-Digit	Full-Sequence
Train	99.17%	96.32%
Validation	97.43%	90.15%
Test	97.02%	88.08%

Table 1: Accuracies

work is preliminary and was therefore only performed on a small number of GAN images, but serves to illustrate the correctness of some of the GAN images and the robustness of the classifier to changes in input.

5 Experiments/Results/Discussion

5.1 Classifier

After training the model, accuracy was measured in two ways. Given each image had 5 components to its label, single-digit accuracy represents how many of those 5 components were accurately for the total number of predictions made, i.e. for a single image with label [1, 9, 10, 10, 10], if the model correctly predicted the first, fourth, and fifth digits, single-digit accuracy would be 0.6. Full-sequence accuracy encapsulates the number of images within a set of predictions wherein the classifier correctly predicted each digit of the image. In the prior example, since the model did not correctly classify the second and third digits of the model, full-sequence accuracy would be 0.

The original Goodfellow et al model performed at 97.84% single-digit accuracy and 96.03% full-sequence accuracy. Based on the accuracies shown in Table 1, it appears that our model has overfit to the train data. This is potentially a consequence of not incorporating sufficient dropout. While our model does do well on the validation and test sets with respect to the full-sequence accuracy, its performance is still lower than Goodfellow et al's. We did not perform any data augmentation in our work which likely would have improved our model's ability to generalize.

5.2 GAN

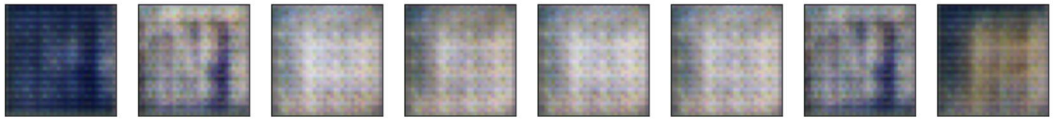

Figure 3: Example of noisy GAN images

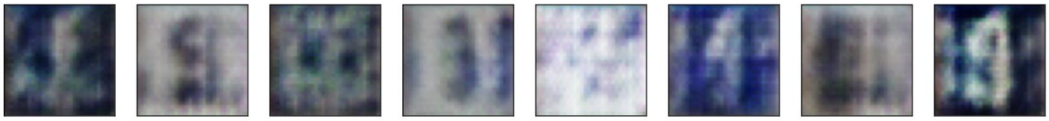

Figure 4: Examples of more legible GAN images

The results of our DCGAN were mixed but fairly positive. GANs are notoriously difficult to train, and ours was no exception. Some training runs led to good results, while others diverged. For example, Figure 1 depicts a DCGAN output with no intelligible numbers and a great deal of artifacts and random noise. In contrast, Figure 2 has several images with legible digits. For example, the rightmost number clearly contains a 4 while the 4th image from the left contains a 3.

We employed many training techniques, including varying the hyperparameters of the Adam optimizer, changing weight initialization, and modifying the number of training epochs. Our approach could be improved by deepening the model architectures, enhancing the training time (we were limited by our one AWS instance), and exploring different training techniques, such as differentially updating the discriminator and generator. We could also have explored adding dropout to the generator function, which has produced good results in some studies.[12] [20]

These results exemplify the randomness inherent in DCGANs. Both of these results were produced using the same model, but one shows much clearer numbers compared to the other. Therefore, improving this model may not require architectural changes, but rather hyperparameter tuning and extended training. We initialized weights from a normal distribution with mean 0 and variance 0.04 and trained the discriminator and the generator networks with an Adam with $\alpha = 0.0002$ and $\beta_1, \beta_2 = 0.9, 0.999$.

5.3 Feeding GAN Images to Classifier

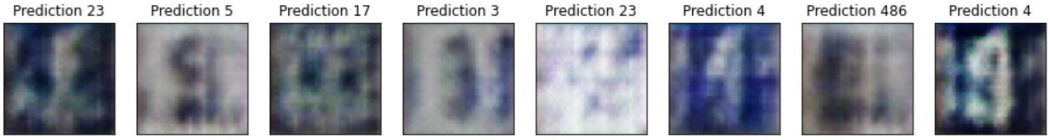

Figure 5: Results predicted by classifier on generated images

Lastly, we present initial results of feeding generated images into the classifier to see whether our GAN-generated images were cognizable by our classifier trained on the original data. Because the GAN seeks to learn the original distribution of the data it is trained on and the classifier performed strongly on that training set. However, because the GAN outputs unlabeled images, our analysis is confined only to those images that clearly depict a number. In the case of Figure 3, the clearest images are the 4th, 6th, and 8th in the top row, which appear to be 3, 4, and 4 respectively. The labels above the images correspond to the predictions of the classifier on the generated images. Multi-digit labels indicate that the classifier believes that multiple numbers are present in the image. For example, a prediction of "23" means both 2 and 3 are in the image. In this case, the only images that have clear labels (the 3, 4, and 4 denoted earlier) appear to be correctly labeled. Therefore, both the GAN and the classifier appear to be performing reasonably well on this limited example. If the GAN were to be improved further (using the methods described in the previous section), we expect that the classifier would easily identify the digits present in synthetic images given its strong performance on the training set and in this example.

6 Conclusion/Future Work

This project has produced a strong classifier using deep convolutional neural networks on the SVHN dataset. The classifier exhibits robust performance and compares favorably to the state of the art on this dataset. We have also presented the initial development of a DCGAN whose architecture was inspired by the classifier to generate synthetic address images. However, despite repeated efforts and variations in architecture and training formula, the results of the DCGAN were inconsistent. This highlights the difficulty in training GANs that is well-documented in the literature. However, our DCGAN was able to produce some legible data that we then ran through our strong classifier to mixed results.

The future directions of the project would lie in improving the performance of the DCGAN and using these results to perform transfer learning on a more sophisticated optical character recognition problem. The DCGAN could be improved through deepening the architecture, tuning the hyperparameters, or exploring more advanced training techniques such as differential updates to the generator and discriminator. Another (simpler) way to improve the GAN would be to simply train a more powerful model on more data for longer. We were resource constrained and therefore limited in our ability to train the model.

7 Contributions

Roshini developed the data pipeline for classification and the classification model. Roshini also developed the data processing to feed GAN images into the classifier network. Avi created the DCGAN and managed the model training using AWS. Avi also recorded the project video submitted. Both authors contributed substantially to conceptualizing the project and debugging all facets. This report was written by both authors jointly.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Bruno Ambrozio and Victor Zacchi. A Convolutional Neural Network model to predict multi-digits over the SVHN dataset., 5 2021.

[3] Francois Chollet et al. Keras, 2015.

[4] Alex Clark. Pillow (pil fork) documentation, 2015.

[5] Andrew Collette. *Python and HDF5*. O'Reilly, 2013.

[6] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *CoRR*, abs/2010.01412, 2020.

[7] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks, 2013.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[9] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[10] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[11] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[12] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. Dropout-gan: Learning from a dynamic ensemble of discriminators, 2018.

[13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[17] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2014.

[18] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[19] Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron C. Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *CoRR*, abs/1505.00393, 2015.

[20] Sabine Wieluch and Dr. Friedhelm Schwenker. Dropout induced noise for co-creative gan systems, 2019.

[21] NVS Yashwanth. Street view house number generation using dcgan, 2020.

[22] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.