Understanding classification failure in Machine Learning using CNNs and GANs

Sanket Mehrotra
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
sanketm@colostate.edu

Rachit Dalal
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
rdalal@colostate.edu

ABSTRACT

The question of fooling neural networks has been around for a long time. Ever since deep neural networks were shown to perform better than competing conventional methods for image classification in 2012 by AlexNet[1] in the ImageNet competition, researchers have pursued the scenario that such powerful classifiers are not perfect and may be fooled. Though this may be applied to neural nets across various applications, such as text or even audio trained networks, it is explicitly visible in the field of image and video classification. The call to place such networks in ubiquitous technology such as phones (face recognition) and autonomous vehicles pushes the need of the hour toward trustworthy deep learning models that may not be fooled by any factors, obvious to humans or otherwise. We explore both manual and generative adversarial methods to explore the effect of noise on trained classifiers. We prepare several different experiments varying image size, resolution and type and magnitude of noise and note the drop in classification accuracy i.e. the noise induced misclassification on CNN, Pre-trained and GAN models trained to high accuracy.

CCS CONCEPTS

• Image Classification • Machine Learning Failure • Neural Networks • Convolutional Neural Networks • Generative Adversarial Networks

KEYWORDS

Image Misclassification, Adversarial Attacks, CNNs, Conditional GANs

ACM Reference format:

Sanket Mehrotra and Rachit R Dalal. 2021. Understanding classification failure in Machine Learning using CNNs and GANs. In *Proceedings of the Genetic and Evolutionary Computation Conference 2021 (GECCO '21). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1. Introduction

Understanding how and why neural networks can be fooled into misclassifying images is an interesting topic that has been widely explored. This topic has gained much attention of researchers as we are rapidly entering the world of self-driving cars and these kinds of failures can causes major hurdles in bringing technologies to the mainstream society. Some papers [2] show that one-pixel change is enough to force a misclassification. This can be done using CNNs or GANs [2], and even uses evolutionary algorithms in some cases [3]. We feel like though a one-pixel attack may be easy, a more common phenomenon in our experience is dealing with more general noisy images when training or testing image classification networks.

Deep neural networks have been widely used for classifying images with the high accuracy. This project is our quest to explore the tolerance and identify weaknesses of image classifying networks to different types and degrees of noise. The addition of noise is almost indistinguishable to the human eyes but surprisingly it can completely fool neural networks. Moreover, another important point which drives this research is its application in a real-time scenarios e.g. If a self-driving car just ignores or misclassifies a stop-sign or a pedestrian because its neural networks have mis-classified its sensors' input images. With this we hypothesize that:

1. It should be possible to identify certain noise thresholds beyond which a network starts regularly failing,
2. Similarly, certain noise patterns may be more disruptive to the classification process of trained networks and even pre-trained networks.
3. Training GANs on structured noise augmented datasets may help us understand the failure of the machine learning models or the solution to overcome this problem.

2. Background

2.1 Supervised learning

Supervised learning can be defined as the process of building a function that maps an input to and output based on previous examples. The neural network learns in a supervised manner by tuning its parameters to adapt to known examples so that it may then generalize to other samples of data previously unknown to it.

2.2 Deep Neural Networks

Deep Neural Networks, or DNNs, can be defined as non-linear approximation functions. Given an input, they can combine several non-linear transformations onto weighted sums to provide an output mapping in a higher or lower dimensional space.

2.3 Convolutional Neural Network (CNN)

A class of an deep neural network mostly used to analyze visual imagery based on the shared weight architecture of filters and convolutional kernels which extract the features instead of crafting hand-designed feature set from the dataset[9].

2.4 Pre-trained Networks

As advanced CNNs get larger and deeper, the resources and compute time taken to train these models becomes larger than most regular machines can handle. One way to be able to use these powerful models in our experiments is to re-use the model weights from models already trained on large image classification datasets like ImageNet[4-5].

2.5 Generative Adversarial Networks

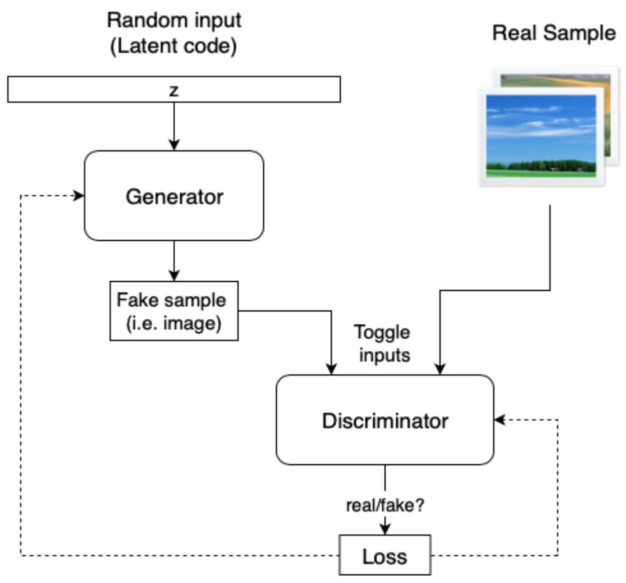


Figure 1: GAN Description: Ref [6]

A Generative Adversarial Network (seen in Figure 1) is a compound neural network architecture designed of two parts. The working of each part is based on a two-player min-max game where the generator G tries to model an input data distribution to generate realistic data to fool the discriminator network while the discriminator D tries to distinguish between real and generated, synthetic data.[7] The value function to be optimized is shown in the below equation (Fig 2.)

$$\min_{G,E}\max_{D} V(G,E,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})} \log D(\mathbf{x}, E(\mathbf{x})) + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log(1 - D(G(\mathbf{z}), \mathbf{z}))$$

Figure 2: MinMax Objective function for a GAN

2.6 Conditional GAN(cGANs)

One drawback of the original GAN[12], we had no control over what output was generated; since the network's input was pure noise. Mahdi and Simon presented the first paper on Conditional GANs in 2014. In the described implementation of this cGAN we can add a conditional input **c** to the random noise **z** so that the generated image is defined by $G(\mathbf{c};\ \mathbf{z})$ [8]. Typically, the conditional input vector **c** is concatenated with the noise vector **z**, and the resulting vector is put into the generator as it is in the original GAN.[7]

2.7 Adversarial Examples

Adversarial examples are examples designed by adding artificial perturbations and noise to a regular image from a dataset with the specific aim to throw off a DNNs classification of that image. Such examples can be made in a simple additive manner by adding random or structured values to pixels in an image. Such modifications can cause the classifier to label the modified image as a completely different class.[2]

2.8 Data Augmentation

Data augmentation is the technique to artificially create the new training data. It helps to deal with the problem of imbalanced classes in a dataset and the performance of deep learning models rely heavily on the amount of data in each class i.e. the class representation. More data in each class will improve both the intra-class variation and inter-class variation, both of which are beneficial to the performance of the deep learning model [10].

2.9 Image Noise

Image noise is the variation in the color/pixel information of in the images. It changes the brightness of the images depending on the type of the noise. Overall, it is an unwanted signal in the image. Depending on the level of the noise added to the images vary the deep learning model performance which is almost indistinguishable to the human eyes but it can fool the neural networks [11].

2.10 Transfer Learning

Transfer learning is a technique of machine learning in which model that is trained on one task can be applied to the related task. So, it is a technique which helps in increase performance of the second task by just fine tunning few layers of the first deep model. This way the training time and compute resources requirements reduced a lot for the related task [13].

3. Motivation

When we started this project, we wanted to understand misclassification of supervised learning algorithms and neural nets. Over time we started work and focused on a subset of this problem, namely the effect of noise and its types on trained and pre-trained networks. Later, we stumbled upon the field of Adversarial attacks and generative adversarial networks. This allowed us to start experimenting with GANs and Conditional GANs in code.

4. Experiments

In this section we describe our dataset and methodology. After a short introduction covering our implementation details, we write about our DNN models, approach to adding noise and our experiments. We also show the results of using both GANs and Conditional GANs to synthesize traffic sign images.

4.1 Dataset

The German Traffic Sign Recognition Benchmark Dataset [2] was published by researchers at the Ruhr-Universität Bochum, Germany in 2011 for the International Joint Conference on Neural Networks (IJCNN). The dataset has the following properties:

- Single-image, multi-class classification problem.
- Dataset contains 43 classes of different traffic size.
- Training and test sets of 39209 and 12630 respectively.

The images in this dataset are of uneven size and range between 25x25 to 225x225 images of road signs. The class dataset distribution graph is shown in the Fig. 3. Additionally, to mitigate this problem of imbalanced classes we have performed different data augmentation techniques like Random rotation of 40 degrees, Random contras of 0.6 and Horizontal and vertical flip. we have Fig 2 shows a few samples of the original dataset without augmentation is being applied.

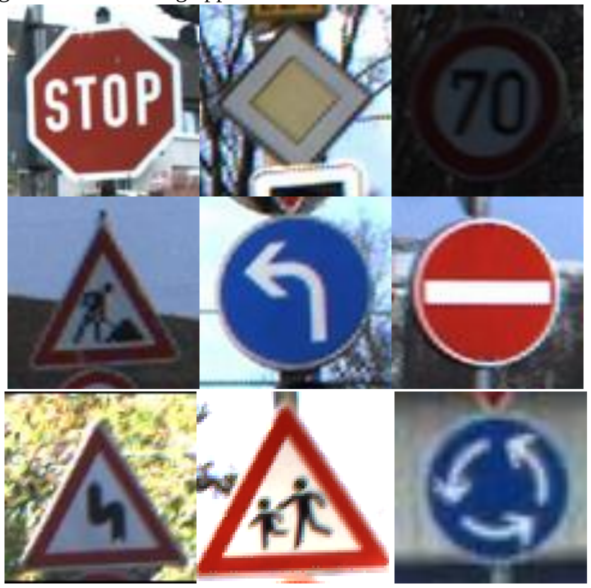


Figure 3: Some samples from the GTSRB dataset

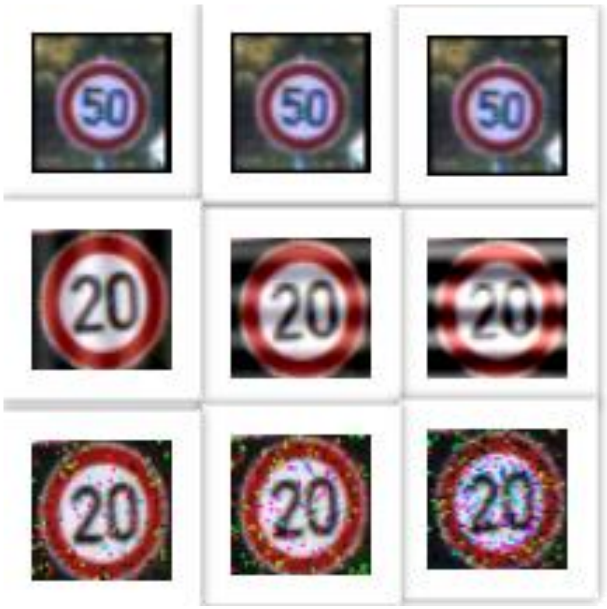


Figure 4: Types and levels of noise introduced to the dataset. From the top, Gaussian Noise, Periodic Noise and Salt and Pepper Noise.

Table 1: Sample Class Labels

ClassId	SignName	ClassId	SignName
0	Speed limit (20km/h)	22	Bumpy road
1	Speed limit (30km/h)	23	Slippery road
2	Speed limit (50km/h)	24	Road narrows on the right
3	Speed limit (60km/h)	25	Road work
4	Speed limit (70km/h)	26	Traffic signals
5	Speed limit (80km/h)	27	Pedestrians
6	End of speed limit (80km/h)	28	Children crossing
7	Speed limit (100km/h)	29	Bicycles crossing
8	Speed limit (120km/h)	30	Beware of ice/snow
19	Dangerous curve to the left	41	End of no passing

4.2 Preprocessing

Using these training and testing images to train our CNNs, pre-trained networks and GANs required some modification. Since all the images were not of uniform shape/size (in terms of width and height), we generated copies of the dataset, each in different sizes to support the specific neural network. The images were regularized by cropping or padding them to the expected size, while making sure that most of the traffic sign still remained in the center of the image. e,g, The InceptionV3 was built to accept inputs of 75*75 while the ResNet50 was designed to allow a minimum image size of 50*50; The GANs we built took an input of 28*28 so that processing time would be quicker and we also trained the CNNs on the 40*40 dataset.

The noisy image datasets were prepared by adding perturbations to the original images. The main three categories of the noise we have taken core are Salt & Pepper Noise, Periodic Noise and Gaussian Noise. We wrote python scripts to re-generate the datasets according to the type and level of the noise. In total we have generated 9 different datasets. Fig. 4 shows different types of noise based on the level. The top row of the grid belongs to salt & pepper noise with the level MAX < AVERAGE < MIN. The second row follows the same level of noise but it is in the Periodic Noise category. The last row falls into Gaussian Noise.

Initially due to memory constraints, we made several copies of the noisy datasets, each with its own version of training and testing images. Later in the experiments this proved useful to prevent load on the RAM which was already being strained under the pressure of training a GAN.

When memory was free, we developed pipelines to transform the images in memory so that we would not have to continuously read and save datasets from the filesystem.

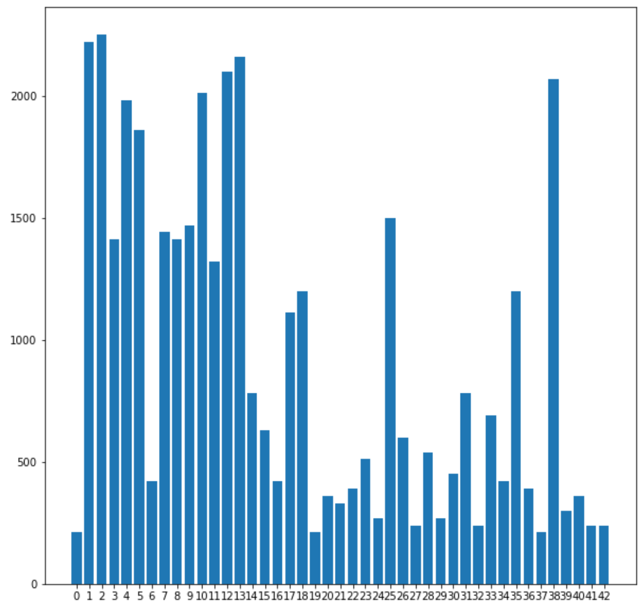
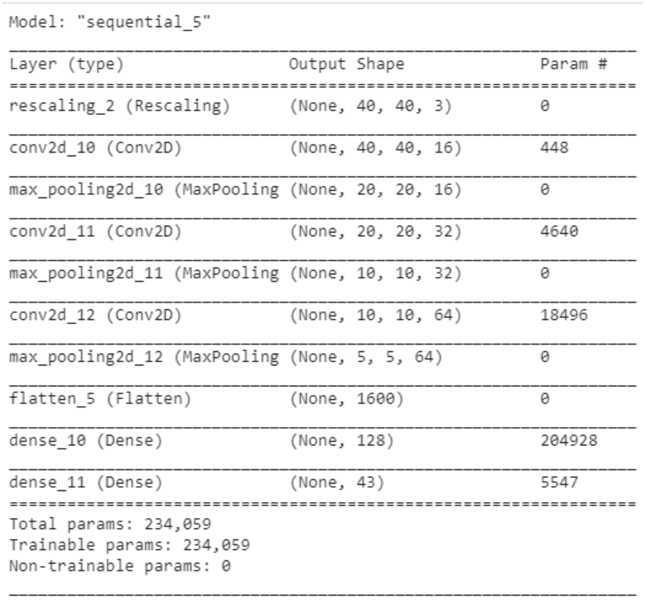


Figure 5: Class Distribution of Training Samples

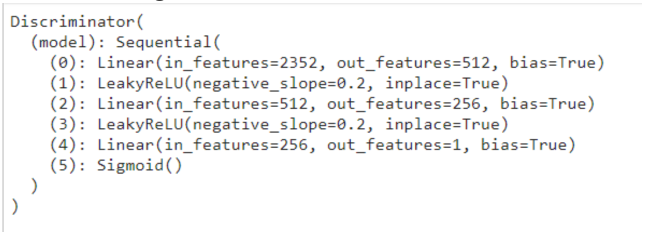
4.3 Implementation

Our implementations of a simple image classification model used as baseline were built with TensorFlow and Keras. After failing to build working networks in Tensorflow, we found some helpful resources that guided us to use PyTorch to build and test our GAN and cGAN[14]. The structures of our networks can be seen in the below figures.



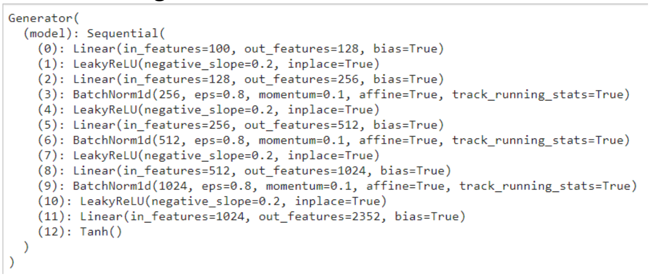
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_2 (Rescaling)      (None, 40, 40, 3)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 40, 40, 16)        448
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 20, 20, 16)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 20, 20, 32)        4640
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               204928
_________________________________________________________________
dense_11 (Dense)             (None, 43)                5547
=================================================================
Total params: 234,059
Trainable params: 234,059
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Baseline CNN Architecture



```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=2352, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

Figure 7: Discriminator Architecture



```
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=2352, bias=True)
    (12): Tanh()
  )
)
```

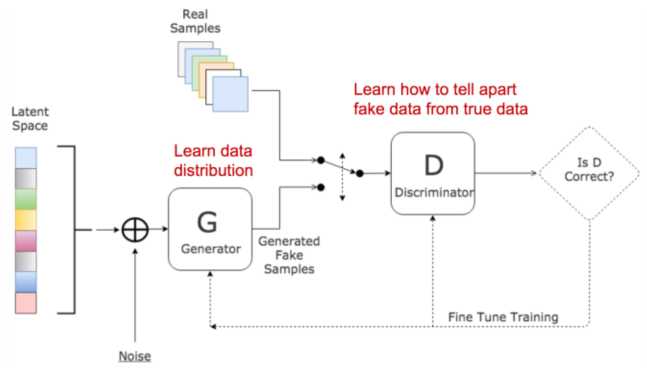
Figure 8: Generator Architecture

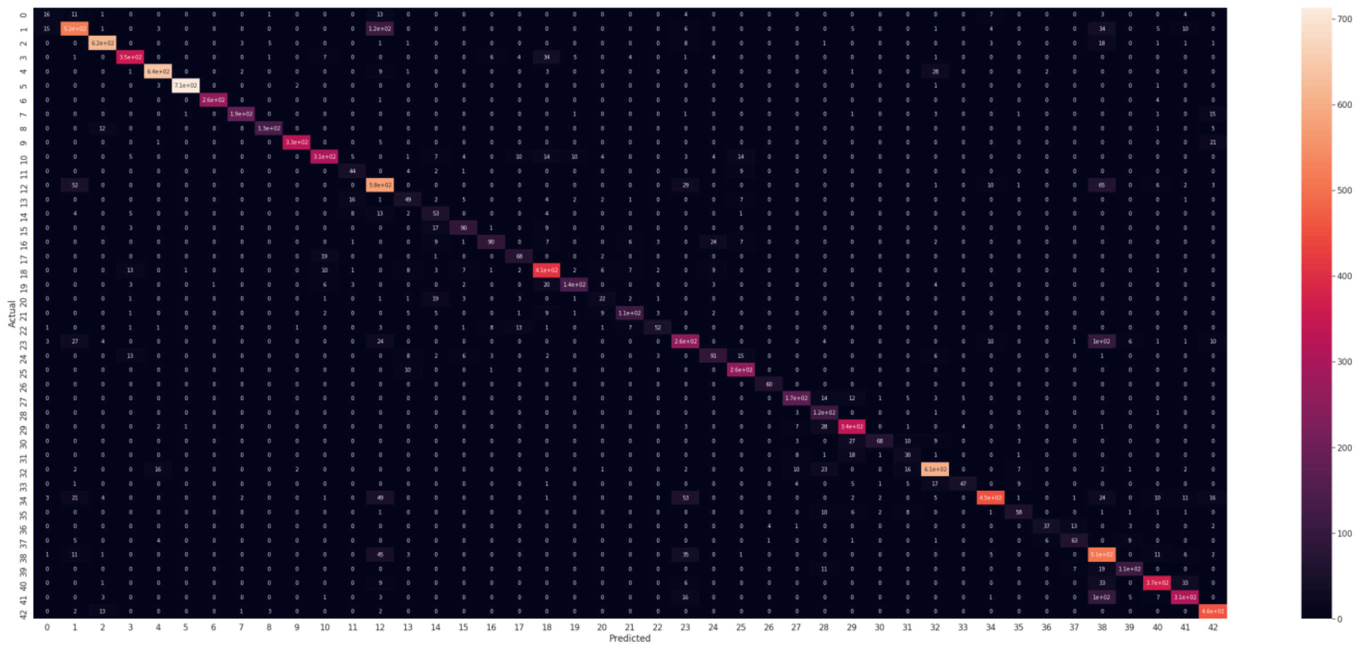
4.3.1 CNN

4.3.2 Pre-trained Models

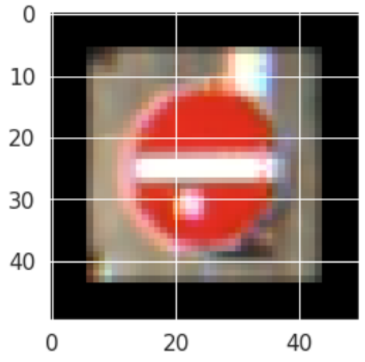
We chose to use largely successful and widely studied networks for our pretrained focused experiments, namely Google's InceptionV3[] and the ResNet50[]. Both these networks are huge in size compared to our baseline CNN model and their architectures are many times deeper.

4.3.3 GANs and cGANs

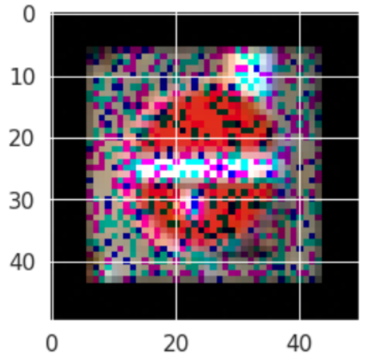

Real Samples
Latent Space
Learn how to tell apart fake data from true data
Learn data distribution
D
Discriminator
Is D Correct?
G
Generator
Generated Fake Samples
Fine Tune Training
Noise


Actual
Predicted
0
100
200
300
400
500
600
700

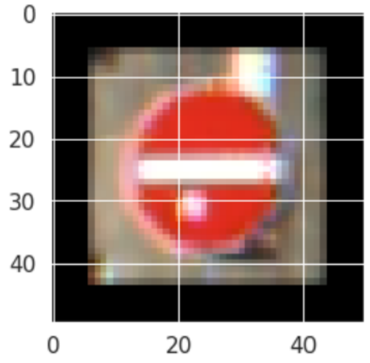
```
Real Label:Ground Truth: 17
Prediction Before Noise: 17
```



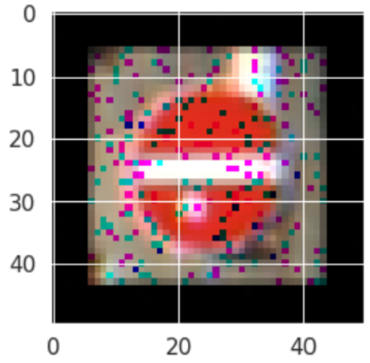
```
Real Label:Ground Truth: 17
Prediction After Salt & Pepper Noise: 12
```



```
Real Label:Ground Truth: 17
Prediction Before Noise: 17
```



```
Real Label:Ground Truth: 17
Prediction After Salt & Pepper Noise: 17
```



5. Results

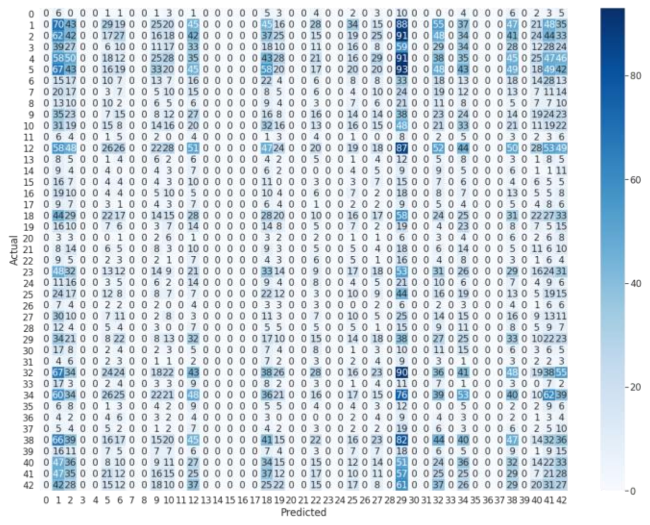
5.1 CNN

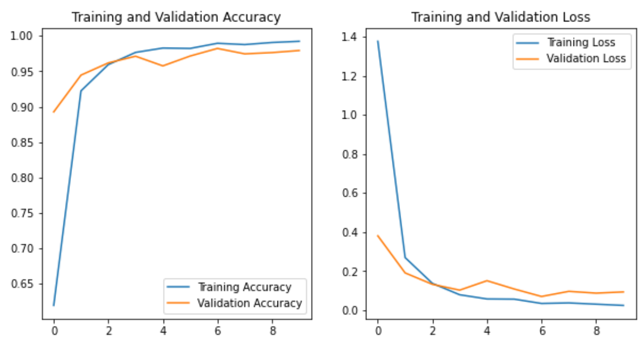
Network	Image Size	Noise Type	Noise Level	Accuracy
CNN				
Pretrained				
GAN				

5.2 Pretrained

5.3 GAN and Conditional GAN

6. Conclusion





ACKNOWLEDGMENTS

Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here. Insert paragraph text here.

REFERENCES

[1] Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” Communications of the ACM, vol. 60, no. 6, 2017, pp. 84–90. Crossref, doi:10.1145/3065386.

[2] Su, Jiawei, et al. “One Pixel Attack for Fooling Deep Neural Networks.” IEEE Transactions on Evolutionary Computation, vol. 23, no. 5, 2019, pp. 828–41. Crossref, doi:10.1109/tevc.2019.2890858.

[3] Nguyen, Anh, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[4] Marcelino, Pedro. "Transfer learning from pre-trained models." Towards Data Science (2018).

[5] Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." arXiv preprint arXiv:1605.07678 (2016).

[6] Horev, Rani. “Style-Based GANs – Generating and Tuning Realistic Artificial Faces.” Lyrn.AI, 3 June 2019, www.lyrn.ai/2018/12/26/a-style-based-generator-architecture-for-generative-adversarial-networks.

[7] Huang, He, Philip S. Yu, and Changhu Wang. "An introduction to image synthesis with generative adversarial nets." arXiv preprint arXiv:1803.04469 (2018).

[8] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

[9] “Convolutional neural network,” 27-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network. [Accessed: 27-Apr-2021].

[10] Jason Brownlee. 2019. How to Configure Image Data Augmentation in Keras. (July 2019). Retrieved April 27, 2021 from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[11] Image noise. (2021, February 04). Retrieved April 27, 2021, from https://en.wikipedia.org/wiki/Image_noise

[12] Goodfellow, Ian J., et al. "Generative adversarial networks." *arXiv preprint arXiv:1406.2661* (2014).

[13] J. Brownlee, “A gentle introduction to transfer learning for deep learning,” 16-Sep-2019. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. [Accessed: 27-Apr-2021].

[14] Eriklindernoren. “Eriklindernoren/PyTorch-GAN.” *GitHub*, 2019, github.com/eriklindernoren/PyTorch-GAN.

Images

1. Dataset introduction grid – example of classes
2. Noise Grid
3. Baseline CNN training graphs
4.