**GAN Obstruction Detection**

**Introduction**

The aim of the project is to improve the classification process of deciding if the area displayed in an image is traversable by a car. The solution is one postulated by Stanford University in their goNet project and involves making use of GAN images for feature extraction. The project aims to achieve a high accuracy on a test set by training on a small training set. This is because of the risk of danger and potential injury that comes with collecting untraversable images. Another aim is to train on a small enough network. This is likely to be used in a real time application (self-driving cars most likely) and should have a high throughput. For reference, the best results for this problem using no feature extraction uses ResNet 152 and reaches an accuracy of 92%[1]. It was trained on a larger data set than what is provided in the goNet database. The best achieved by our implementation with a limit of 10 layers and our available data was 85% accuracy. We hope to represent our findings in the following manner. The second section is a summary of our data source and the details surrounding that data source. The third section explains our feature extraction process which is the center of our algorithm. The fourth section goes over all the classifiers used and the fifth section is used to present implementation details and results of each classifier and the reason for such results. Section six deals with issues and takeaways and section seven will serve to conclude.

**Data source**

Our project is an implementation of the idea used in goNet. The data we used was from the goNet website which was taken using a robot rover to traverse buildings and hallways at Stanford and using a fisheye camera to take pictures. The data set is made up of a massive 80,000 image set of positive images. The aim of this set is to help train the GAN network. More about this will be revealed in the next section. It is also made up of a smaller set of labeled images with 1200 positive images and 1200 negative images. These are used to train the classifiers. They are separated equally into sets of training images, validation images and testing images with 800 images each. Each image is a 128 x 128 x 3 colored jpeg image[1].

**Feature Extraction**

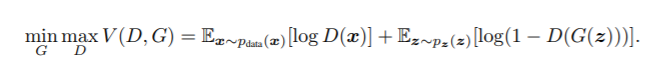
Generative adversarial networks are neural networks that enable computers to generate data that belongs to a distribution from noise. This is done by setting up a generative network that produces images from noise and setting up a discriminative network that tries to distinguish generated images from real images. Then you train both networks using the following min-max function:[2]



Fig 1. Generated Images from GAN

We essentially train the discriminator to distinguish between the generated images and the real images in our data, but we train the generator to trick the discriminator into thinking generated images are real. The ideal result is that the network converges with the accuracy of the discriminator being about half and the distribution of the generator being roughly the distribution of the real images.

Using this technology, the aim is to amplify the discriminative features of the data and downplay features that do not retain any discriminative images. This is achieved by training a GAN with a second constraint; that each generated image is similar to the input image passed into the generator. These networks are then trained on a large set of positive images. The result is a network that tries to replicate the positive image that gets passed through it. The intuition is that, if a negative image is passed through the network the generator will try to create a positive image that is as close as possible to the input negative image. This task will fail and there will be a massive difference between both images. The expectation is that the combined real and generative images of the positive and negative images will hold more discriminative features than just the real images. This is because the difference between the positive real and generated will be a lot smaller than the difference between the negative real and generated.

So, for this network the first constraint is that the generated image is from the distribution of positive training images and the second is that the generated image is similar to the input image. A number of approaches were explored to achieve this double constrained GAN network and ended up with an inverse generator that takes an image input and generates the random distribution input to the generator. The inverse generator was then trained to minimize the difference between

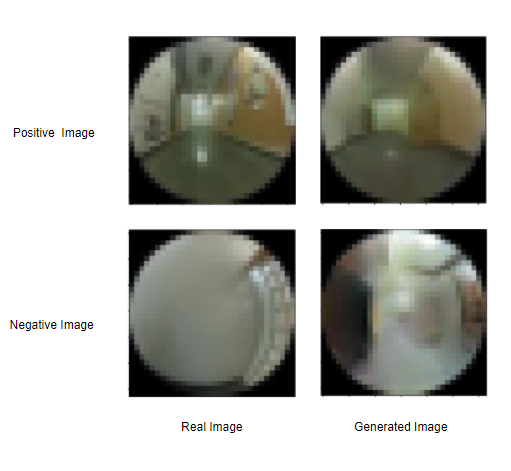


Fig 2. Sample Generated and Real Images

the input image and the generated image. Some of the configurations that we tried are seen below:

1. First attempt:

a. Grey scale images

b. 128 x 128 x 1 dimensions

c. No inverse generator

Results:

· No convergence ( terrible generated images)

Reasons:

· Massive search space i.e., takes too long to converge.

· No attempt to make input similar to the generated image.

Note: Attempt 1.5 was trying to sample the random distribution from the input image in an attempt to achieve the second constraint. This was pretty naïve and did not give the desired results.

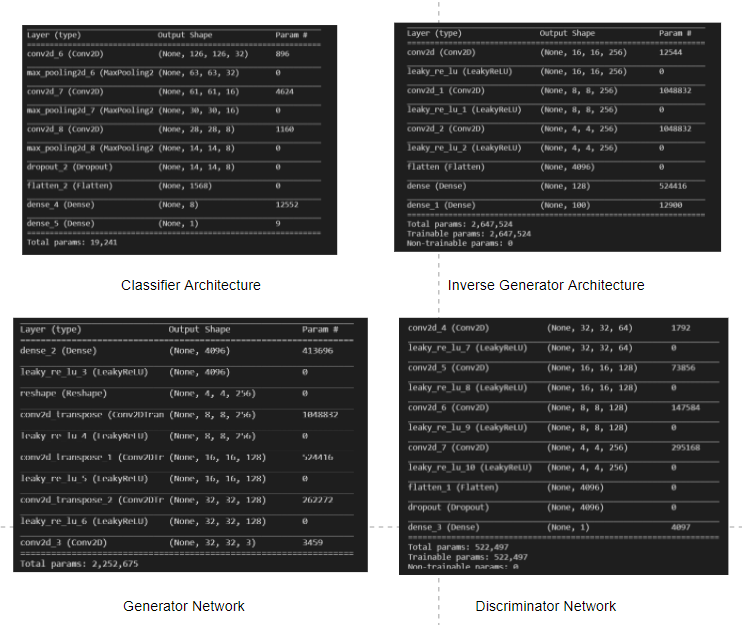


Fig 3. Neural Network Architecture Summaries[3]

2. Second attempt:

a. Grey scale images

b. Inverter generator introduced.

c. 128 x 128 x 1 dimensions

Results:

· Better images (blurry)

Reasons:

· Massive search space i.e., takes too long to converge.

· Over fitting to similarity constraints.

· Possibility of data lost to color.

3. Third attempt:

a. Color images

b. 32 x 32 x 3 dimensions

c. Inverse generator

Results:

· Best attempt. Images that match the requirements where observed.

Reasons:

· Manageable search space. Converged in two hours.

**Classification Algorithms and Methodology**

The main algorithm used for classification was the convolutional neural network. A network with three convolutional layers, each paired with pooling layers was used alongside a dropout layer and two dense layers. This was then trained on the training set of 800 images and tuned using a validation set of the same size. The network was then trained on both training and validation sets and tested on the test set. The equal size distribution showed that the images can be trained on a smaller set than is conventional, which was one of our goals earlier. Other classifiers explored were SVM’s, Decision trees, Gaussian Naïve Bayes, and Decision trees. We also used PCA’s to reduce the dimensions of the data for the SVM algorithm. The bagging ensemble method was also applied to the CNN method to improve the results from the CNN method. The results of the algorithms are seen below.

**Results**

1. GAN MODEL:This network was set up solely for feature extraction. The architecture of each component network is seen in figure 3. There are two operations that take place, The first is training the GAN network. This is made up of the discriminator and the generator.

The discriminator takes in inputs of size 32\*32\*3 and gives out a single output of 0 or 1 indicating, generated or real. It is trained using the binary cross entropy function and the adam optimizer with a learning rate of 0.0002 and a momentum value of 0.5. It is trained on a bootstrapped batch of real images, each labeled 1 and a set of generated images each labeled 0. The set is 64 images large and is made up of 32 real images and 32 generated images.

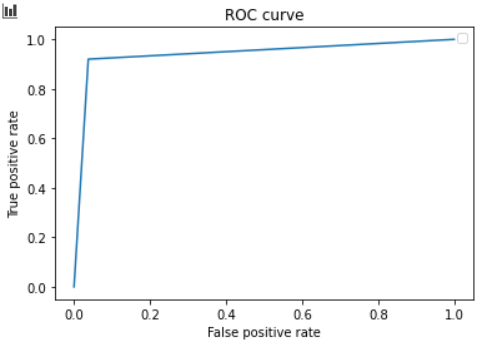
This generator is trained alongside the discriminator. This takes in a one dimensional vector of 100 random variables and gives out a 32\*32\*3 image. It is combined with the discriminator and trained using the same parameters above.This training occurs with the discriminator weights frozen. The training data is a 64 sized batch of random vectors each sampled from a normal with zero mean and variance of one. Each item in the batch is labelled with a one in an attempt to deceive the discriminator.

The above processes are done alternatively making up one epoch. The GAN is trained for 100,000 epochs. Results of the training are seen at image fig 1. The inverse generator network is then connected to the generator and trained to minimize the distance between the input and the generated image. The generator network is frozen during this training. The network takes in a 32\*32\*3 image and outputs a 100 sized noise vector. This network is trained using the mean squared error loss function and the adam optimizer with a learning rate of 0.0003 and a momentum value of 0.5. The data passed into the network is the 80000 image set used as both input and label. Since the label is the same as the input image, the generated image will try to emulate this as close as possible while retaining the weights of the generator. The combination of the inverse generator and generator is termed the encoder. Results of the encoder are seen in figure 2.

1. CNN: For the CNN implementation the network used is seen in figure 3. It was trained using the binary cross entropy loss function and the adam optimizer with a learning rate of 0.0003, a momentum value of 0.5. It is trained for 350 epochs with a batch size of 64. The results are as follows:
   1. Accuracy: 94 %
   2. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Gold Positive | Gold Negative |
| Pred Positive | 368 | 15 |
| Pred Negative | 32 | 385 |

* 1. Precision: 0.96
  2. Recall: 0.92
  3. F1-score: 0.94
  4. ROC curve:

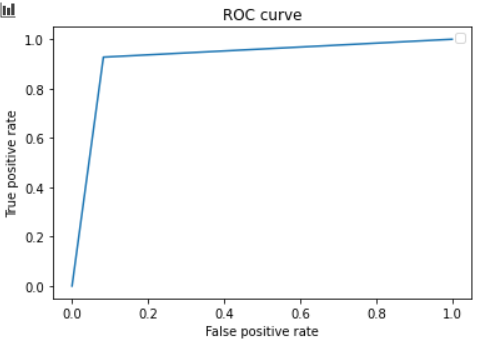


This gives the best results of all implemented classifiers. The precision and recall values show that the algorithm is slightly skewed towards negative predictions. The reason for this is likely tied to the GAN. As a result of computational constraints, the generative model is likely to have not converged completely meaning that the generated image is not as close as it possibly can be to the input image. As a result it is likely that the generated image of some positive examples are so different from the real images that they were classified as negative.

1. CNN Ensemble: There was an attempt to run a bagging algorithm on the CNN classifier. It used 10 trained networks and took a bootstrap of 1550 images to train each classifier. The parameters used are the same as those in the CNN implementation. The results are seen below.
   1. Accuracy: 92.3 %
   2. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Gold Positive | Gold Negative |
| Pred Positive | 371 | 33 |
| Pred Negative | 29 | 367 |

* 1. Precision: 0.92
  2. Recall: 0.93
  3. F1-score: 0.92
  4. ROC curve:

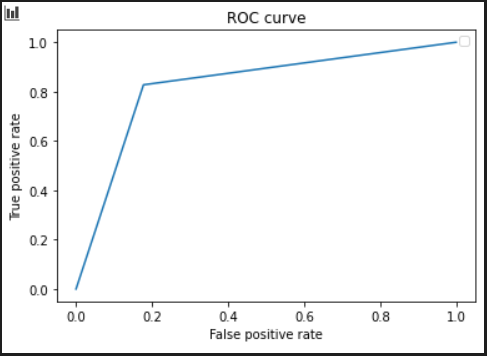


The bagging algorithm doesn’t do as well as the CNN implementation, even though we expect it to. This might indicate a low error contribution from variance. The more likely issue is the instability of the learning method due to the use of dropout for regularization.

1. SVM Implementation: The SVM implementation was run on a flattened version of the training data. It was trained on a C value of 9 using a gaussian kernel. These were obtained through hyperparameter tuning. The results are as follows:
2. Accuracy: 82.5 %
3. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Gold Positive | Gold Negative |
| Pred Positive | 331 | 71 |
| Pred Negative | 69 | 329 |

1. Precision: 0.82
2. Recall: 0.83
3. F1-score: 0.825
4. ROC curve:



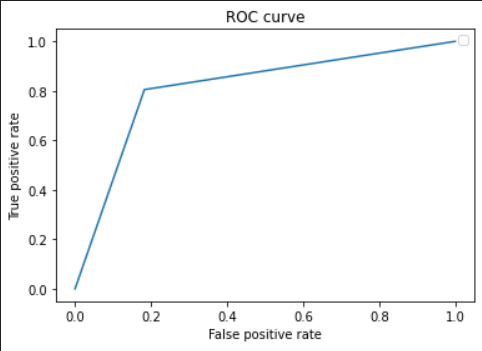
This performs considerably worse than the CNN implementation. We speculate that this is as a result of the absence of spatial and temporal info that is retained in the CNN implementations. We also notice that the skew towards negative examples is not displayed here. This also hints at the fact that the SVM is not using the similarities between the real and generated images for its classification.

An implementation with PCA dimensionality reduction was also attempted in a bid to curb the curse of dimensionality and give better results. The aim was to retain 99% of the data and it resulted in a reduction from 6144 features to 474 features. This however did not reach the same accuracy as the first implementation. This means that we lose some discriminating features in the 1% of data that was lost. The results of this implementation is seen below.

1. Accuracy: 81 %
2. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Gold Positive | Gold Negative |
| Pred Positive | 322 | 73 |
| Pred Negative | 78 | 327 |

1. Precision: 0.82
2. Recall: 0.81
3. F1-score: 0.81
4. ROC curve:

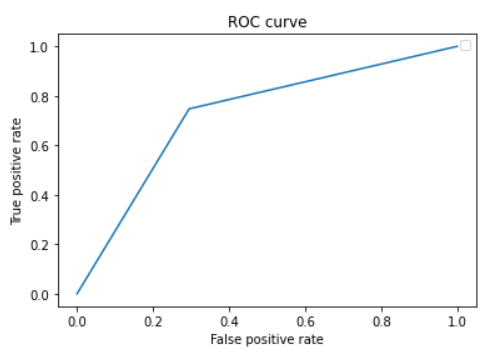


5. Decision Tree: A decision tree based on the entropy criterion was trained on the flattened data. Through testing an optimal tree depth of 16 was obtained. The evaluation metrics are listed below:

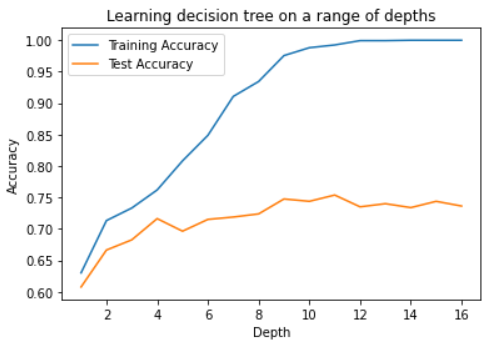
1. Accuracy: 71%
2. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Gold Positive | Gold Negative |
| Pred Positive | 299 | 118 |
| Pred Negative | 101 | 282 |

1. Precision: 0.72
2. Recall: 0.75
3. F1-score: 0.73
4. ROC curve:



The decision tree does not perform as well as the other classifiers. This is because the model easily overfits the training data and so cannot generalize to other examples. One reason for the model overfitting is because our data consists of too many features which increases the model complexity and which can also lead to spurious correlations, hindering the primary objective of the learner, namely to identify whether there are any obstacles present in the centre of the image or not. To rectify this we at first tried to limit the growth of the tree by setting the maximum depth of the decision tree and then increasing it in gradual intervals.



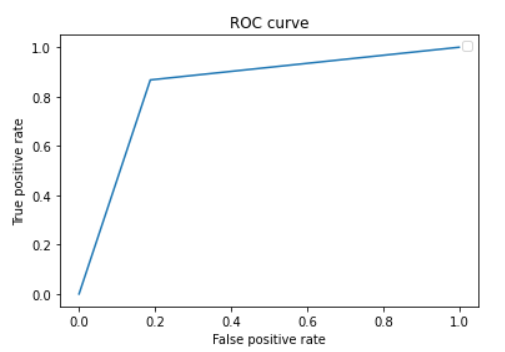
It seems that while it does not overfit on the training data anymore, we also lose classification accuracy for the test set. Next we post-pruned the tree which did not lead to significant improvement in accuracy. Finally running PCA on the dataset actually reduced the accuracy of the algorithm to 63%, meaning that a lot of discriminating features were removed while trying to reduce the number of features.

6. Random Forest: In an effort to improve the accuracy of the decision tree, we now implement a random forest and our results seem to improve significantly

1. Accuracy: 84%
2. Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Gold Positive | Gold Negative |
| Pred Positive | 347 | 75 |
| Pred Negative | 53 | 325 |

1. Precision: 0.82
2. Recall: 0.87
3. F1-score: 0.84
4. ROC



Decision tree is a high variance learner in the sense that its output can change dramatically with a change in the data, leading to low test accuracy. By taking an ensemble of decision trees and averaging their results we can lower the variance, leading to better generalization over test data.

**Issues and Takeaways**

Our major issue was computational ability and time. Training with full sized images was made near impossible and training of the GAN network took an extremely long time. With more processing power and more time, more experiments would have been conducted, and more approaches would have been explored: Some of the things in mind where:

1. Using edge detection on images and training with those

2. Using the GAN network to augment data and increase the training set size.

3. Training on full colored images etc.

The biggest takeaways from the implementation are as follows:

1. The GAN algorithm: Aside from the general idea of the algorithm and the math behind it, one of the takeaways is that most collected data that hold any information are representative of real distributions and the only issue is that we do not explicitly know the parameters of the distribution. This is easy to think about in relation to simple numerical data but is really an eye opener when data like images and sounds come into play.

2. Feature extraction: The intuition gained from using this strategy to improve the algorithm shows reveals the essence of performing feature extraction. The aim of feature extraction is to automatically promote discriminative features in your data before the use of a classifier. So rather than going after error by variance or bias, feature extraction goes after error by noise.

3. Reasons for CNNs: There is a clear difference in the performance of the CNN’s and the other algorithms in this application. The reason for this was not clear until we had to implement them side by side. For every other algorithm there was the need to flatten the images before training. This causes a loss of the spatial dependencies that exist between pixels.

**Conclusion**

This problem is one that can be approached from a lot of different perspectives and this is an approach that was particularly interesting. A lot of experience with ML strategy and usage of different algorithms was gleaned from it. It also gets the results we expected which is a plus. Another perspective that can be taken is trying to train the GAN to learn how to generate negative images so we can generate the hard to obtain negative images and not need to learn from a small set. This will be explored in the future.

References

1. N. Hirose, A. Sadeghian, M. Vázquez, P. Goebel, and S. Savarese, “GONet: A Semi-Supervised Deep Learning Approach For Traversability Estimation,” *arXiv.org*, 08-Mar-2018. [Online]. Available: https://arxiv.org/abs/1803.03254. [Accessed: May-2021].
2. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Networks,” *arXiv.org*, 10-Jun-2014. [Online]. Available: https://arxiv.org/abs/1406.2661. [Accessed: 04-May-2021].
3. J. Brownlee, “How to Develop a GAN to Generate CIFAR10 Small Color Photographs,” *Machine Learning Mastery*, 01-Sep-2020. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-cifar-10-small-object-photographs-from-scratch/. [Accessed: May-2021].