Captcha Generation and Identification using Generative Adversarial Networks

Hardik Ajmani[*]
SRM Institute Of Science and Technology,
Chennai, India
HardikAjmani@gmail.com

Mrinal Wahal[†]
Amity University
Noida, India
MrinalWahal@gmail.com

Abstract

Adversarial attacking is an emerging worrying angle in the field of AI, capable of fooling even the most efficiently trained models to produce results as and when required. Inversely, the same design powering adversarial attacks can be employed for efficient white-hat modeling of deep neural networks. Recently introduced GANs *(Generative Adversarial Networks)* serve precisely this purpose by generating forged data. Consequently, authentic data identification is a crucial problem to be done away with, considering increased adversarial attacks. This paper proposes an approach using DCGANs *(Deep Convolutional Generative Adversarial Networks)* to both - generate and distinguish artificially produced fake captchas. The generator model produces a significant number of unseen images, and the discriminatory model classifies them as fake (0) or genuine (1). Interestingly enough, both the models can be configured to learn from each other and become better as they train along.

Keywords - *Deep Convolutional Generative Adversarial Networks, CAPTCHA, Deep Learning*

1 Introduction

CAPTCHA, which stabds for **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part, was an innovation to avoid and obstruct the attempts to automate the digital human-specific authentication operations. In 1999, the students of CMU and MIT deployed automated scripts to ram the online voting ballots with more than a thousand votes [35].

Both the schools had $21,000+$ votes, whereas the other schools ended up with a meager score of fewer than 1000 votes. This incident was a significant factor in the development of captchas. However, captchas were developed in order to stop earlier generations of what may now be termed *Artificial Intelligence*, or more so *Early Automated Intelligence*.

Presently, using advanced machine learning designs, any level of automation may be deemed within sight to engineers. These tasks are as straightforward as cat vs. dog prediction or as complex as painting an image provided a text/voice based description.

A recently significant talk of the town has been GANs (Generative Adversarial Networks) [1]. His paper proposed a class of specialized neural networks capable of producing fake data which stands astonishingly close to real data. Contained within it are two interconnected networks, a discriminatory network and a generatory network. The discriminator is essentially a binary classifier i.e., the output is delivered as a boolean value (True or False) and distinguishes between artificial and real data. The generator initiates itself with random noise and then attempts to reform it during training

[*] HardikAjmani.com
[†] MrinalWahal.com

into look-alike-original data, such that the discriminator classifies it as real, or as close to real as possible. Considering both the networks are trained together, they aid each other while training.

A DCGAN (Deep Convolutional Generative Adversarial Network) integrates a deep convolutional network in both the networks. Hence, DCGANs are specialized GANs for improved image processing. These networks, with some improvements in the architecture, manage to replicate the images efficiently, and also produce a new combination of unseen images with superior results [3]. The new data is employed to assist the training of secondary networks, like human face detection algorithms where lack of data is a notable concern. We also share novel generated images, created by permutation of the existing data.

However, this also introduces an issue of creating counterfeit data, capable of deceiving neural networks and are called adversarial attacks [10] on neural nets. Therefore, developing a robust discriminator which can differentiate between authentic and fraudulent data would assist in avoiding such attacks by adding a supplementary layer of defense.

2 Literature Review

With the ongoing surge in the complexity of neural networks, online captchas are turning effortless to be cracked with automated scripts powered with deep neural nets. Thereby, the pitfall of security remains unfixed and orthodox captcha generation methods are unsubstantial and hence obsolete. This parchment proposes a new technique to bridge the existing limitations to certain extent by utilizing DCGANs.

This paper uses GANs, proposed initially in [1]. It describes the use of two interconnected networks which engage in a min-max activity. As aforementioned, the discriminator network endeavors to maximize the error-free classification of original and generated data. Whereas the generator network tries to diminish the loss of the produced output from impulsive noise. Deep generative networks' impacts were inconsiderable because of its inability to use all of its benefits. The paper resolves this issue by combining a discriminator and generator.

Many variations of GANs have been adapted since then, and [2] the paper presents some pragmatic instances of CGANs (Conditional GANs). This adaptation offers both the networks to accept input conditionally. It helps in more accurate and objective defined outputs. A new attribute y is added to the inputs, and it helps in mapping of the dataset to the significant distinct classes. Captcha's dataset also constitutes multiple classes; each different captcha is a new class. Since our goal was to generate new captchas using the existing ones, we have not implemented the CGANs.

This [3] paper explains the unsupervised variant of GANs. It foregrounds how unsupervised learning was unchartered with CNNs (Convolutional Neural Networks) and its promising advantages. Hence, it proposes the architecture of DCGANs for an effectual blend of unsupervised learning and CNNs. The networks can be trained with unlabeled data and learned feature representations could be reused at a significant number of places. DCGANs tame the constraints of GANs, thereby finding a notable role in captcha generation.

The obstacle in scaling are a result of limited hardware, and consequently hinders the ability to train on higher resolution images directly. Therefore, the images have to be down-scaled before training. SRGANs [4] (GANs for super-resolution images) is a novel technique to solve the issue of low resolution in GANs; it offers up to 4x times scalable images. Referenced document proposes the use of perceptual loss for producing high-resolution images.

[37] propose another successful attempt using Conditional GANs to produce high-quality images. They generate 2048 X 1024 realistic images employing new adversarial loss and new multi-scale architectures. The paper also explains the extension of their work to incorporate two additional features. First, object manipulation, which allows addition/subtraction of new objects and changing the type of existing ones. Second, they install feature of one-to-many mapping, allowing different results from the single input, helping users to edit the images interactively.

3 Technical Review

Elaboration and background of technical terminologies used excessively in this parchment to aid the reader's comprehension of the further work.

3.1 Adversarial Attacks and Learning

The deep learning models are spectacularly good in the classification of audio and visual. However, as explained in the paper of [10], these networks are highly vulnerable to adversarial examples. They explain two significant flaws for the same. First, despite being individual or in randomized linear combinations, high-level units are indistinguishable. It implies that space contains the most semantic knowledge in the upper layers of the network rather than the individual units. Second, they explain the discontinuous input-to-output mapping of the network, which causes the model to misclassify the picture with minimal perturbation by maximizing the network's error. An optically similar image of a *panda* with noise is misclassified as *Gibbon* with 99% accuracy.

The paper [36] asserts linearity of networks as a primary vulnerability to adversarial examples. It incorporates adversarial training applying the fast gradient sign method based function as an efficient regularizer. After increasing the number of units to 1600 layers, they reduced the test set error from 1.14% to 0.782%. Moreover, the model depicted resistance to adversarial examples.

3.2 Generative Adversarial Networks

Generative models accomplish unsupervised learning tasks with core objective to comprehend the unlabelled distribution of data [5]. Such networks try to regenerate the probability distribution and are implemented as graphs [6], where the nodes represent a random variable, and the arcs represent relations between these nodes. Their eye-catching success [7, 8] has been attributed to Backpropagation and dropout algorithms using piece-wise linear units [9]. Taking into account their inefficiency due to difficulty in the approximation of complex probability distribution and inability to get benefits of piece-wise linear units [1], created grounds for introduction of adversarial networks. In GANs, the generative model has to learn along with discriminator model.

We can interpret it as an analogy of a fraudulent painter who is trying to sell his work to a critic. The critic can judge the difference between real and fake work and hence labels the output as $\mathbf{0}$ or $\mathbf{1}$. With each iteration, the painter tries to update his painting in order to deceive the critic. This game continues and shoulders both of them to improve until the critic is no longer able to declare the difference between the authentic and forged painting.

The Generator learns distribution p_g over data x. The noise variable is defined by z, and the mapping is defined as $G(z;\theta_g)$, where G is a generator and a differentiable function with parameters θ_g. Discriminator, another differentiable function is defined by $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ is the probability over real data x. GANs tries to maximize D's probability of assigning correct labels to x and p_g. Parallelly G tries to minimize the loss over D. These networks hence enages in the min-max game with value function $V(G,D)$ [1]:

$$min_G max_D V(D,G) = \mathbb{E}_{x\sim p_{data(x)}}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \tag{1}$$

3.3 Deep Convolutional GANs

Classical methods, like K-means, can be used for hierarchical clustering of image patches in [11]. DBN (Deep belief networks) were also proved to perform well in learning hierarchical representations [13]. Another popular method was to use the autoencoders, for stacked convolution layers trained to denoise the input like in [12]; it bridged the performance gap with the DBN.

However, there was a gap for CNNs [3], they were unexplored for unsupervised learning. There comes a new set of CNNs called DCGANs, which proved to be stable for training as compared to GANs, the trained discriminator could contend against other unsupervised learning algorithms, it could visualize filters, and smooth manipulation of the generated images' semantic qualities.

Generative parametric models are widely explored. Original GANs [1] generated images that suffered from noise and were unintelligible. Other approaches included a recurrent network approach like in [14] called DRAW (Deep Recurrent Attentive Writer). However, to integrate GANs and

CNNs several earlier made attempts were unsuccessful. This lead to the development of LAPGAN (Laplacian Generative Adversarial Networks) [15] and later DCGANs [3].

The core changes adopted in DCGANs were [3]:

- Replaced all spatial pooling functions (like max-pooling) with strided convolutions for both the generator and the discriminator like the All Convolution Network in [16].
- Eliminated all of the fully connected layers from the architecture. A strong example of this is global average pooling like in [17]. A middle ground between stability and inability to converge was obtained by directly connecting the output of the generator to the discriminator.
- Batch Normalization [18] of each layer helped in stabilizing the learning and helped in avoiding the mode collapse, a renowned problem in GANs. The instability was circumvented by not using batchnorm to the generator's and the discriminator's outputs and inputs, respectively.
- The ReLU [19] activation for generator (the output layer used the Tanh activation) and the Leaky ReLU activation for all the layers of the discriminator. It allowed quick color saturation of the image [3].

4 Architectural Design

4.1 Adversarial Network

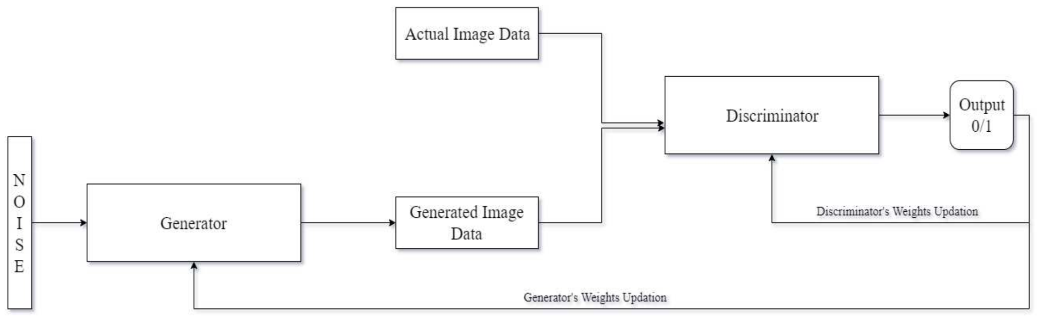


Figure 1: Adversarial model

The adversarial network is constructed by connecting the generator model and the discriminator model, as depicted in figure 1. Note, that the discriminator is compiled separately in order to update the weights individually, and the generator's weights are updated using the results of the discriminator.

Similar to the discriminator network, for compilation the RMSProp optimizer [26] is employed with the (learning_rate = 1e-4) and the (decay = 3e-8) [25]. The binary crossentropy is used for the loss and the evaluation metrics is accuracy [27].

4.2 Discriminator

Figure 2 portrays the architecture of the discriminator, the input is the captcha image (explained in section ahead), and the output is between 0 and 1. It is a conventional deep CNN architecture. The input is an image with dimensions 30 * 150.

Each layer (except for output) has a LeakyReLU activation with alpha = 0.2. Initial layer has depth = 64, and it doubles for every layer. Each layer also has a dropout = 0.4 [22], to prevent overfitting in the network. Typical values for dropout rates range from 0.5 - 0.8. Padding is padding = same for every convolution layer. The final layer is a flatten layer with dimensions from previous convolution layer 4 * 19 * 512 = 38912.

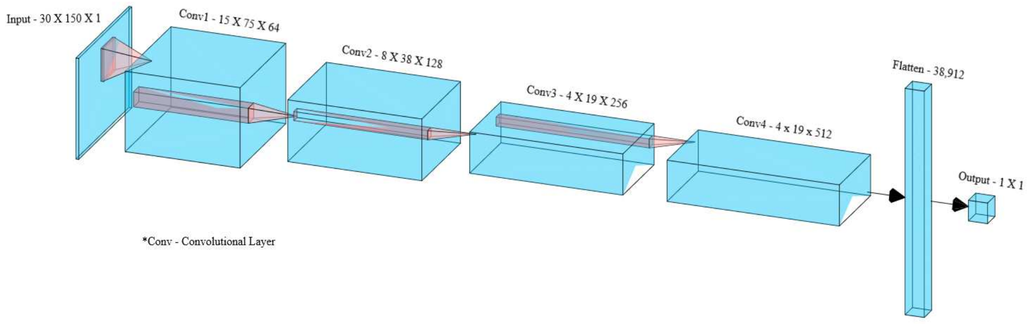


Figure 2: Discriminator model

Finally, the last component is a dense layer with one node, and the Sigmoid activation [23] function applied to it, and ensures the output lies between 0 - 1. For compiling, initially, the Adam optimizer [24] (with learning_rate = 2e-4 and decay = 6e - 8 [25]) was used similar to [3], but after experimentations and referring to [20] the RMSprop optimizer(with learning_rate = 2e-4 and decay = 6e - 8 [25]) was tried and it portrayed better stability and learning.

The RMSProp (Root Mean Square Propagation) adapts the learning rate for each of the parameters. It divides the learning rate by calculated average of the recent rates [26].

The binary crossentropy loss is used for the back-propagation and accuracy metrics for the evaluation.

Figure 3 gives the architectural details. It is generated using Keras' summary() method, explained in [21].

4.3 Generator

Figure 4 depicts the architecture of the generator model. It is a DDCN (Deep De-Convolution Network) [28], and it is deployed to perform semantic segmentation on CNN.

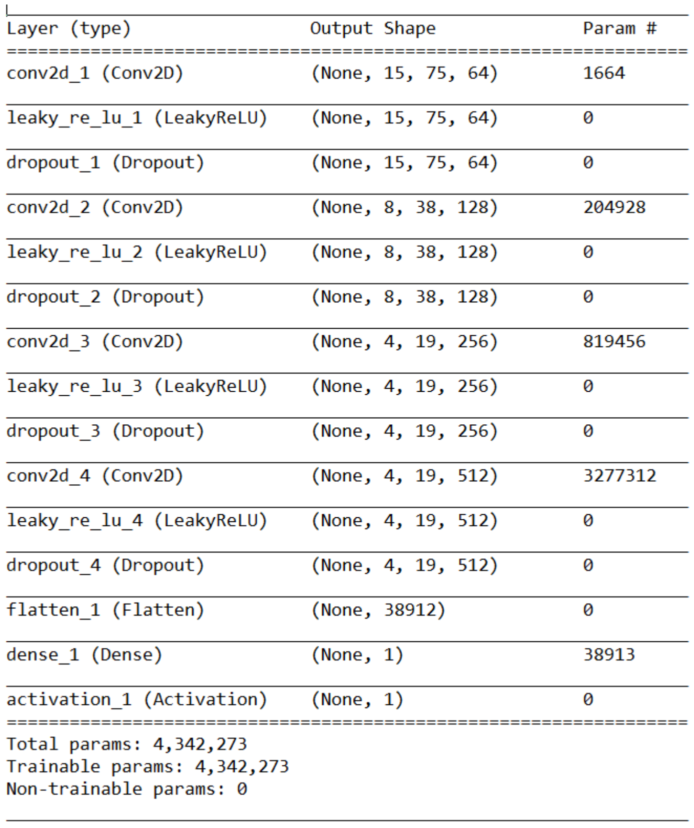
We input random noise in the generator, and 30 * 150 dimensioned image is the output, which is in accordance to the actual data sent to the discriminator network as input.

Similar to the discriminator network, each layer has dropout = 0.4 [22] and activation function as ReLU [19](except for output layer). Also, each layer has Batch Normalization with momentum = 0.9 [18].

We have chosen initial depth as 64 * 4 = 256, and it will be halved with each layer. The first layer (fig 4) is a dense layer of dimension 8 * 38 * 64 * 4 = 77824, the 8 * 38 is necessary for upscaling the final output image to 30 * 150. After applying batch_norm, this layer is reshaped into a 3-D vector of 8 * 38 * 256, and a dropout is applied. In the next layer, the image is upscaled (8 * 38) -¿ (16 * 76), then a deconvolution layer is applied, which makes the final dimensions as 16 * 76 * 128. It is followed by the batch_norm and the ReLU activation. The same is repeated until the last deconvolution layer with dimensions of 32 * 152 * 16.

The final layer is only a deconvolution layer, which reduces the depth parameter to 1, and converts it into a single 2-D vector of dimensions 32 * 152. Before activation, cropping is applied to convert image into 30 * 150. At last, the Tanh activation function, necessary for the image's pixels to lie in the range of [-1, 1].

Contrary to the discriminator model, the generator model is not compiled separately, it is incorporated into the adversarial network, along with the discriminator, explained in the section ahead.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 15, 75, 64)        1664
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 15, 75, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 75, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 38, 128)        204928
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 8, 38, 128)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 38, 128)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 19, 256)        819456
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 4, 19, 256)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 19, 256)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 19, 512)        3277312
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 4, 19, 512)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 4, 19, 512)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 38912)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 38913
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,342,273
Trainable params: 4,342,273
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Discriminator Architectural Details

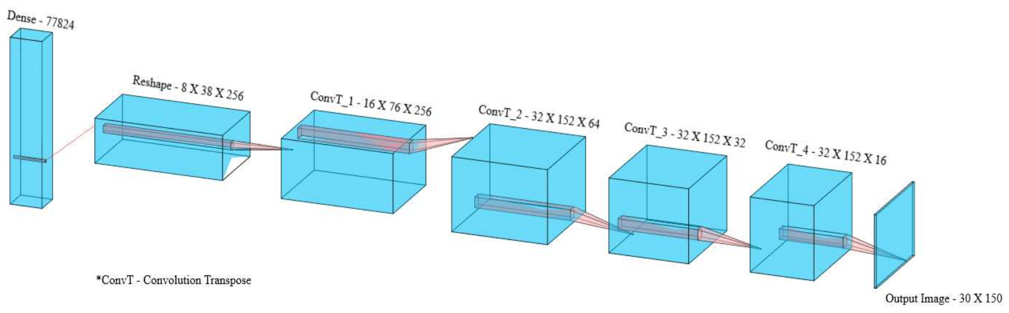
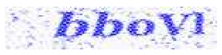


Figure 4: Generator model

5 Data and Pre-procesing

The data is 3,600 pictures of 30 different captchas. We scrapped it from [36]. All the images preprocessing functions are performed using the OpenCV library. Figure 5 shows an example of raw and processed data. Here is the sequence of steps:

1. The images are accessed in grayscale format.
2. The Binary Threshold is applied to the images [30]; it set a predefined value if the pixel is higher than a threshold.
3. The images are then blurred (to get rid of noise) using MedianBlur [31].
4. Finally, the images are normalized from (0 ,255) to (-1 , 1) for the Tanh activation function.


(a) Raw Captcha

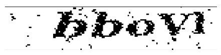

(b) Processed Captcha

Figure 5: Data: Before and After preprocessing

6 Training and Evaluation

6.1 Algorithm

```
$iter = 0, epochs = 10000, batch\_size = 256$;
while $iter \leq epochs$ do
    fake_Image ← generator(noise);
    data ← concatenate(real_Image, fake_Image);
    output[real_Image] ← 1;
    output[fake_Image] ← 0;
    discriminator.train_on_batch($x, y$);
    output[fake_Image] ← 1;
    adversarial.train_on_batch(noise, $y$);
    print epoch, loss, accuracy;
    $iter \leftarrow iter + 1$
end
```

GANs are somewhat troublesome to train, due to their instability, as explained in [1, 3]. We trained the model for 10000 epochs with a batch size of 256. Algorithm 1 explains all the steps performed while training.

The dataset is loaded with real images (after preprocessing), and a set of images is produced using the generator. Then initially, the real images are alloted 1, and the generated images are set to 0 for the training of the discriminator.

For, the adversarial network's training, the output is set to 1 for the generated images. Note that, we use the train_on_batch method for training of both the networks [32].

6.2 Results

As explained in the aforemintioned section, we trained the model for a little more than 10000 epochs. Table 1 shows the average losses and the accuracies for both the adversarial and the discriminator models. Table 2 shows the actual raw and the processed data compared with the generated data. Also, table 3 displays some unseen captchas generated. Figure 6 plots the loss versus accuracy graph for both the discriminator and the adversarial network. Note the instability in the training. The centerline plots the *mean* with a window of 1000 epochs to represent the trend in the training.

	Average Loss	Average Accuracy
Discriminator	0.2314	91.977%
Adversarial	8.3181	0.2542%

Table 1: Mean Metrics for both the models

The average accuracy for the discriminator is $\mathbf{91.977}\%$, and for the adversarial network is $\mathbf{0.2542}\%$. We have not trained the model to the ideal discriminator loss of $\mathbf{0.5}$, because we intend to create a robust discriminator and at the same time generate some unseen captchas, which were evident at this number of epochs, and therefore we did not over-train it.

Although, in experimentations, the model was trained up to $\mathbf{15000}$ epochs, which lowered the loss and reduced the number of anew generated captchas.

It proves that the GAN's permute the data like a jig-saw puzzle, moreover, newly generated images are just a failure of fixing the right piece in the right puzzle.

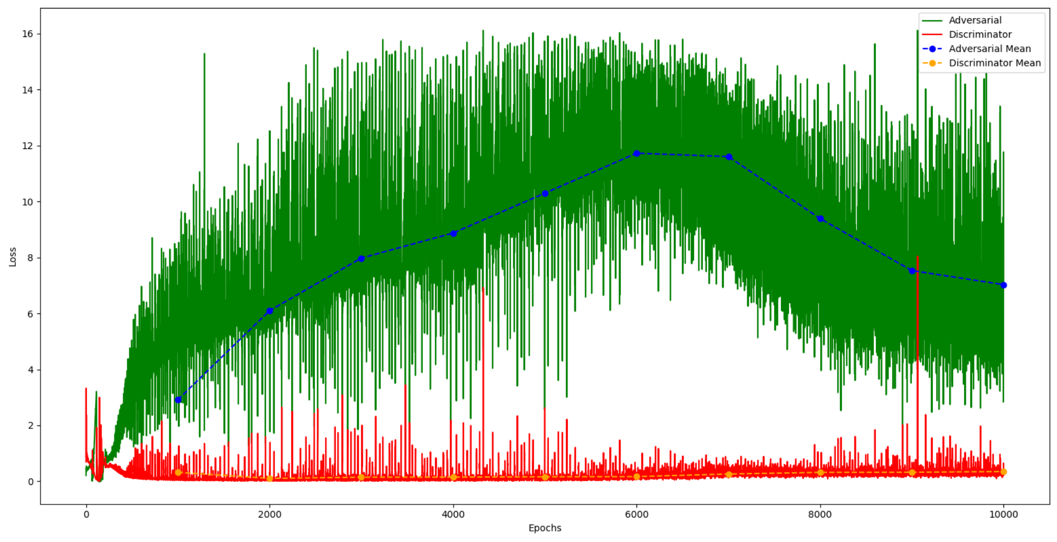


Figure 6: Loss Vs Epoch Graph for Discriminator and Adversarial Newtork

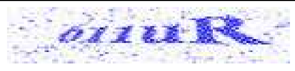
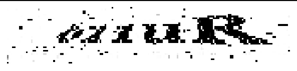
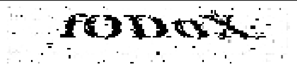
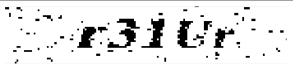
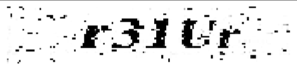
Raw	Processed	Generated
		
		
		

Table 2: Actual and Generated data

7 Limitations

Drawbacks observed in this research stand as followed:

1. **Mode collapse**: Defined as the tendency of the generator to create the same output to deceive the discriminator. It tends to depend upon 3 - 4 classes out of the whole dataset to diminish loss as early as possible. During training, the generator assumes a single fixed point to be the global optimum and thus producing the same output regardless of the variation in noise. Mode collapse is revisited in various ways [3, 37], and occasionally occurs, in most of the cases.

2. **Variety in data**: Due to previous two reasons our model is based on only 30 different images. With increased variety the generator gets stick into local minima and is unable to train further.

8 Conclusion and Future Works

One of the most interesting applications of the this work can be for enhanced training of facial recognition technologies. The discriminatory model can be re-configured to detect and prevent adversarial attacks on the same face recognition technology. Our team is working to cure the aforementioned limitations [7] of the work with interesting approaches involving the WGANs (Wasserstein GAN) [33] which applies a novel loss function along with certain architectural changes. Another interesting application this research work is serving as the basis for is detection of counterfiet currency and subsequent training of such detection models for banking systems worldwide.

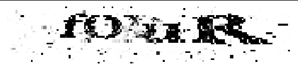
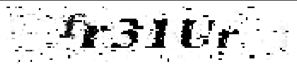
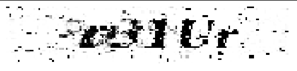
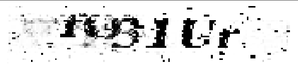
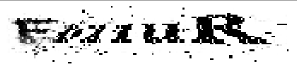
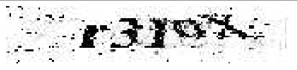
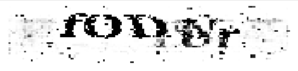
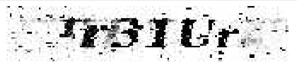
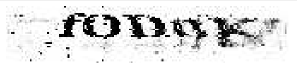
 foUR	 fr31Ur	 o31Ur
 fo31Ur	 F011uR	 r31oX
 foDUr	 rr31Ur	 fODoK

Table 3: Unseen Captchas Generated

References

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

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

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

[4] Ledig, Christian, Lucas Theis, Ferenc Huszár, Alejandro Acosta, Jose Caballero, Andrew Cunningham, Andrew Aitken et al. "Photo-realistic single image super-resolution using a generative adversarial network." *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681-4690. 2017.

[5] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends® in Machine Learning 2*, no. 1 (2009): 1-127.

[6] Jordan, Michael Irwin, ed. Learning in graphical models. *Vol. 89. Springer Science and Business Media*, 1998.

[7] Hinton, Geoffrey, Li Deng, Dong Yu, Abdel-rahman Mohamed, George Dahl, Navdeep Jaitly, Andrew Senior et al. "Deep neural networks for acoustic modeling in speech recognition." *IEEE Signal processing magazine 29* (2012).

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

[9] Jarrett, Kevin, Yann LeCun, and Koray Kavukcuoglu. "What is the best multi-stage architecture for object recognition?." *In 2009 IEEE 12th international conference on computer vision*, pp. 2146-2153. IEEE, 2009.

[10] Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Rob Fergus, and Ian Goodfellow. "Intriguing properties of neural networks." *arXiv preprint* arXiv:1312.6199 (2013).

[11] Coates, Andrew Y. Ng , and Adam. "Learning feature representations with k-means." *In Neural networks: Tricks of the trade*, pp. 561-580. Springer, Berlin, Heidelberg, 2012.

[12] Vincent, Pascal, Hugo Larochelle, Isabelle Lajoie, Pierre-Antoine Manzagol, and Yoshua Bengio. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research 11*, no. Dec (2010): 3371-3408.

[13] Lee, Honglak, Roger Grosse, Andrew Y. Ng, and Rajesh Ranganath. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." *In Proceedings of the 26th annual international conference on machine learning*, pp. 609-616. ACM, 2009.

[14] Gregor, Karol, Alex Graves, Daan Wierstra, Ivo Danihelka, and Danilo Jimenez Rezende. "Draw: A recurrent neural network for image generation." 'arXiv preprint arXiv:1502.04623 (2015).

[15] Denton, Emily L., Rob Fergus, and Soumith Chintala . "Deep generative image models using a laplacian pyramid of adversarial networks." *In Advances in neural information processing systems*, pp. 1486-1494. 2015.

[16] Springenberg, Jost Tobias, Thomas Brox, Alexey Dosovitskiy, and Martin Riedmiller. ”Striving for simplicity: The all convolutional net.” *arXiv preprint* arXiv:1412.6806 (2014).

[17] Mordvintsev, Christopher Olah, Alexander, and Mike Tyka. ”Inceptionism: Going deeper into neural networks.” (2015).

[18] Ioffe, Sergey, and Christian Szegedy. ”Batch normalization: Accelerating deep network training by reducing internal covariate shift.” *arXiv preprint* arXiv:1502.03167 (2015).

[19] Nair, Geoffrey E. Hinton, and Vinod. ”Rectified linear units improve restricted boltzmann machines.” *In Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807-814. 2010.

[20] Rowel Atienza. ”GAN by Example using Keras on Tensorflow Backend”. Towards Data Science (Blog) (2017) Cross Ref.

[21] ”About Keras models” Keras Documentation Cross Ref.

[22] Srivastava, Nitish, Geoffrey Hinton, Ilya Sutskever, Alex Krizhevsky, and Ruslan Salakhutdinov. ”Dropout: a simple way to prevent neural networks from overfitting.” *The journal of machine learning research 15*, no. 1 (2014): 1929-1958.

[23] Finney, David John. Probit analysis: a statistical treatment of the sigmoid response curve. *Cambridge university press, Cambridge*, 1952.

[24] Kingma, Jimmy Ba, and Diederik P. . ”Adam: A method for stochastic optimization.” *arXiv preprint* arXiv:1412.6980 (2014).

[25] Krogh, John A. Hertz, and Anders. ”A simple weight decay can improve generalization.” *In Advances in neural information processing systems*, pp. 950-957. 1992.

[26] Tieleman, Geoffrey, Tijmen and Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* (2012)

[27] Aditya Mishra. ”Metrics to Evaluate your Machine Learning Algorithm” (2018) Cross Ref.

[28] Noh, Seunghoon Hong, Hyeonwoo, and Bohyung Han. ”Learning deconvolution network for semantic segmentation.” *In Proceedings of the IEEE international conference on computer vision*, pp. 1520-1528. 2015.

[29] ”Login” Tourism Development Council Manali (H.P.)-GreenTax Cross Ref.

[30] ”Image Thresholding” OpenCV Documentation Cross Ref.

[31] ”Image Filtering” OpenCV Documentation Cross Ref.

[32] ”Sequential” Keras Documentation Cross Ref.

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

[34] Von Ahn, Luis, Nicholas J. Hopper, Manuel Blum, and John Langford. ”CAPTCHA: Using hard AI problems for security.” *In International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 294-311. Springer, Berlin, Heidelberg, 2003.

[35] Luke Metz, David Pfau, Ben Poole,Jascha Sohl-Dickstein. ”Unrolled Generative Adversarial Networks” *arXiv preprint* arXiv:1611.02163 2017

[36] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. ”Explaining and harnessing adversarial examples.” *arXiv preprint* arXiv:1412.6572 (2014).

[37] Wang, Jun-Yan Zhu, Ting-Chun, Ming-Yu Liu, Jan Kautz, Andrew Tao, and Bryan Catanzaro. ”High-resolution image synthesis and semantic manipulation with conditional gans.” *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8798-8807. 2018.