A Comparative Study of Car Image Generation Quality Using DCGAN and VSGAN

Mirza Ahmad Shayer[1], Nafisha Anjum[2], Sushana Islam Mim[3], Md. Abu Sajid Chowdhury[4], Noshin Nanjiba Islam Preoshi[5], Moin Mostakim[6]
[1,2,3,4,5,6] Department of Computer Science and Engineering, BRAC University, 66 Mohakhali, Dhaka 1212, Bangladesh
Email: [1]mirza.ahmad.shayer@g.bracu.ac.bd, [2]nafisha.anjum@g.bracu.ac.bd, [3]sushana.islam.mim@g.bracu.ac.bd, [4]md.abu.sajid.chowdhury@g.bracu.ac.bd, [5]noshin.nanjiba.islam.preoshi@g.bracu.ac.bd, [6]mostakim@bracu.ac.bd

***Abstract*—In today's modern society, image generation (synthesis) has a great number of uses in various tasks. Image generation is used in crime forensics, improving image quality and generating better images. In 2014, a scientific breakthrough occurred in the machine learning community when Ian Goodfellow and his colleagues introduced the GAN (Generative Adversarial Network). Ever since then, GANs have become a more popular concept in the scientific community. This thesis is a comparative study of two GANs used for generating images of cars- DCGAN (Deep Convolution) and VS-GAN (Vehicle Synthesis). The study will determine which of the two is better suited to generate high quality images of cars. We will train both GANs using the same dataset. The dataset consists of about 16185 Google images of random cars, 8144 for training and another 8041 for testing. The dataset is already preprocessed and split. We will compare the GANs training times, losses, accuracies and pictures generated, showing how well they perform. We will run all the GANs for 40 epochs in both training and testing. We will compare the CGAN, DCGAN, VSGAN, WGAN and WGAN-GP, to see which performs the best. We have used K-Nearest Neighbors, Regression and Random Forest Classifier to calculate the accuracies of all the GANs.We have displayed the results in tabular and graphical formats. We believe this will improve GAN research by providing an excellent comparison between the GANs and determine which is better suited for the given task.**

***Index Terms*—image generation, GAN, CGAN, DCGAN, VS-GAN, WGAN, WGAN-GP, epochs, training, testing, K Nearest Neighbors, Regression, Random Forest Classifier**

I. Introduction

BEFORE creating the generative adversarial Network, people used different means of generating images, such as, using traditional neural networks for classification. The images were either distorted or generated poorly. In 2014, when Ian Goodfellow and his colleagues made the GAN, things were substituted. Here the generator generates the images based on calculations while at the same time the discriminator attempts to detect the fake images through calculations as well. Therefore, the generator generates more accurate images to trick the discriminator while it tries to find the fakes better to tackle the generator. Accuracy increases, and the images generated are far sharper and closer to the original. Many different variants have been made for several purposes.

A. Other Uses of GANs

Generative adversarial networks are composed of two neural networks - generator and discriminator. The generator learns to make new samples, whereas the discriminator learns to detect the generated data from the true ones. In the industry, GANs have a wide range of applications like gaming, cyber security, etc. It is often used in military surveillance, traffic control and economic or social studies [1]. Also used in automatic vehicle re-identification [2]. Can be used for image cross-domain classification in aerial vehicles [3] [4]. Often used for vehicle image synthesis as well [5]. Unsupervised neural networks and generative adversarial networks train themselves by examining data from a dataset to generate new picture examples. GANs are mainly used for improving cyber-security, healthcare, generating animations, translating images, and editing photos [6]. Can create 3D models automatically for cartoons, video games and animated movies. Also used for image enhancement. Are used to remove rain and snow from photographs. Can also be utilized for remaking face images and identify changes such as gender, hair color and expressions [7].

B. Research Methodology

We have decided to take a sample size of 6000 images, for both training and testing. We did not use the full sets of 8144 - training and 8041 – testing, as some issues were present in the images past 6000 for both sets. We have trained and tested the given architectures – CGAN, DCGAN, VSGAN, WGAN and WGAN-GP. We have fed the images to all the GANs. We trained and tested them for 40 epochs each. We have recorded the training and testing times, losses of both generators and discriminators and have given a side by side comparison of some of the generated images of all models.

II. Literature Review and Related Work

A. Literature Review

A generative adversarial Network (GAN) is a robust neural network utilized in unsupervised machine learning. GAN can catch and copy variations within datasets. This technique generates new data with similar stats as the training-set given

the trained data set. At first, we input the images, then we systematically pre-process the images. The given input noise generates fake images and sends this to the discriminator, where the discriminator compares these images with the real images. When the discriminator can detect fake images easily, back-propagation happens in discriminator and generator until the discriminator cannot detect the fake images quality becomes close to the real ones. The comparisons are done computationally where the generated images' accuracy, sharpness, and quality are assessed. GANs solve the problem of generating data when we do not have enough time and require no human supervision. Labelling data is a time-consuming manual operation. They do not need labelled data to be trained. It can learn the internal representations even if the data is not marked. They can make graphics, text, audio, and video that look just like the genuine data.

B. Related Works

GANs helps us to build good predictive models and now we can generate realistic fake data using GANs. The generator continuously attempts to make false images and the discriminator tries to find out the fakes from the real ones. DCGAN: The main purpose of the DCGAN is to produce convolution based images. In the discriminator we produced 64x64 images. Used conv2d to make a 32x32 image then LeakyRelu activation in the discriminator for all layers. Used conv2d to process 16x16 images, then Batchnorm2d in both the generator and the discriminator to extract the features. Used conv2d again to produce an 8x8 image and again LeakyRelu activated in the discriminator. Following this pattern we generated 4x4 images, then 2x2 and finally a 1x1 image. We also added some noise to every intermediate output [8] [9]. CGAN: We use the Conditional-GAN for balancing image dataset. If we train a CGAN, we can generate novel images for the class that needs balancing. Since DCGAN doesn't let us to control the appearance of the samples, that's why we need to condition the GAN output. We sampled the noise from a normal distribution and we accounted the class labels. We also added the number of classes as well as the discriminator. Then we distributed it uniformly with the label identities [10]. WGAN: The Wasserstein GAN replaces the generator model and omits the use of a discriminator to organize the probability of generated images. WGAN is a more stable architecture. There are some changes in this GAN compared to DCGAN or CGAN. We did not use the 'Sigmoid' function, instead linear activation-function. It updates the generator model more times in each iteration [11]. WGAN-GP: To train this model we had to train the discriminator for more iterations than the generator. The gradient penalty follows the idea that functions are 1-Lipschitz. The squared difference from Norm-1 is utilized as the gradient penalty. For the original WGAN to achieve 1-Lipschitz weight clipping is used, which leads to capacity underuse. WGAN-GP is more stable to train than WGAN [12]. VSGAN: Vehicle Synthesis Generative Adversarial Network generates interpreted vehicles from remote sensing images. Uses two discriminators and one generator. It is an extension of the DCGAN. A patch-GAN is used in both discriminators to detect sharper images. The generator has a basic residual block structure is added . A series of a convolution layers are used to increase depth and reduce size. A background discriminator is utilized to learn the context information of the background. The vehicle discriminator distinguishes the positive vehicles from the negative vehicles [1]. The VSGAN can only process a very small number of samples. Too many results in a higher chance of mode collapse. The six residual blocks causes this limitation. Removing one or two residual blocks, will allow an increase in sample size input, albeit at the cost of accuracy and quality as less features will be extracted due to a shallower network.

C. Using GANs for Vehicle Image Synthesis

Convolution Neural Networks have been utilized to make more promising results. An end-to-end detection model is a combination of shallow and deep feature maps, following a convolution that is deforming and Region-of-Interest pooling. There is the issue of heavy dependence on the quality and quantity of observations in the training-data [1] [13] [14] [15] [16]. Two identical vehicles may be mistaken for the same model or car type. Different viewpoints can also get in the way of properly recognizing or generating images due to their 3D structure. Two similar yet different vehicles viewed from the same viewpoint look more alike than seen from two different perspectives [2] [17] [18]. The DCGAN architecture is improved using the Wasserstein loss to reduce mode collapse [19] [20] [21].

III. Research Methodology

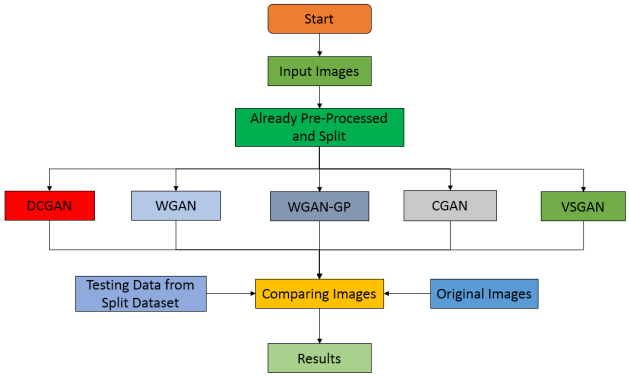


Fig. 1. Research Method

Our methodology is shown above. Here, we input the images on all the GAN's keeping the hyper-parameters constant. The sample size and epochs are kept similar for all models. We note the training and testing times. We used a sample size of 6000 images, for both training and testing and used 40 epochs. This is done for all the models, except the VSGAN. For the VSGAN we chose a random sample of 40 images from both sets of 6000 images. The VSGAN only works with a small numbers of samples. We did not use the full sets because, there are some small issues with the last images in both. We put the training and testing times, losses and accuracy of the models

and the generated fake images along with the real images of each model using the best visual representations.

A. GAN

GANs are used for a multitude of purposes like voice, image and video generation. The generator generates the fakes, and the discriminator catches the fakes. This forces the generator to make better images and the detector to detect them better. After running for many epochs, they both start to converge. We can get perfect fakes that look close to the original. For our research, we have used DCGAN, WGAN, WGAN-GP, CGAN and VSGAN. The VSGAN is relatively new, and much research has not been done. We have compared the results to see how well they perform on the given set of conditions.

B. DCGAN

Deep Convolution Generative Adversarial Network utilizes deep convolution neural networks for both the generator and discriminator models. The training parameters and model results in consistent training of the generator. The difference is how it performs its functions. Here, the discriminator in this scenario is mainly composed of strides of convolution-layers, batch-normalization layers, and LeakyRelu activation function. It receives a 3x64x64 image as input, uses convolution-transpose layers, batch-normalization layers, and ReLU-activations to compensate the generator. The outcome will be a 3x64x64 RGB picture. The DCGAN is significant, as it proposed the model restrictions needed to create high-quality generator models effectively. For instance, this architecture served as the foundation for quickly creating many GAN extensions [8].

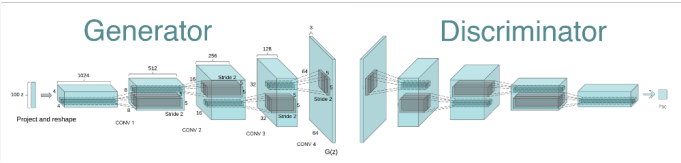


Fig. 2. Architecture of DCGAN

C. WGAN

Wasserstein GAN improves image quality and stabilizes training. WGAN produces realistic images in addition to more excellent stability. In GAN, there is a problem in the Jensen-Shannon (JS) divergence. If we can replace JS with WS, we can minimize the Earth-Mover's distance. The problem of mode collapse plagues GANs, preventing the generation of ideal images. WGAN provides us stable curves and better stability. Unfortunately, it takes longer to train and sometimes fails to converge. The gradient penalty in WGANs does not show unwanted behavior such as weight clipping. WGAN is designed to reduce issues with training traditional GANs while avoiding the problem of mode collapse. The WGAN's value function is created using Kantorovich-Rubinstein duality producing $minG$, $maxDD$. The WGAN value function produces a critic function whose gradient is better behaved with its input making generator optimization easier [11] [22] [23].

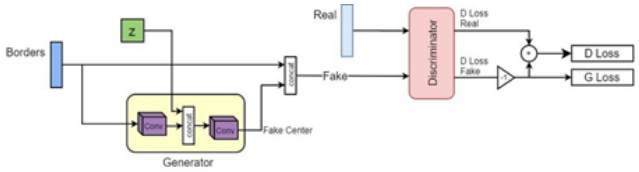


Fig. 3. Architecture of WGAN

1) WGAN-GP: WGAN-GP was introduced to overcome the weight clipping problem. Sometimes WGAN fails to converge for both the generator and discriminator, which WGAN-GP solves. In WGAN-GP, we use Gradient Penalty for better conversion. It achieves Lipschitz continuity by combination of Wasserstein loss formulation and gradient-norm penalty. Weight clipping is used in the original WGAN to perform 1-Lipschitz functions, but without careful adjustment of the weight clipping parameter, it can lead to unwanted behavior such as surfacing of pathological value, gradient explosion or vanishing due to capacity under-use. WGAN-GP is much more stable to train then WGAN. Gradient Penalty is a variant of the Lipschitz constraint which is softer, this arises as functions are Lipschitz-1, if their gradients have a maximum norm of 1 [12].

D. CGAN

Conditional-GANs can generate images with specific conditions or attributes. CGAN is where, generator and discriminator are prepared to utilize the extra samples. This model creates samples with a similar structure as the training sample observations corresponding to the same label, when given a label and a random array as input. The discriminator receives batches of labeled data comprising statements from both the training and the produced data. This network attempts to identify the observations as "actual" or "fabricated." CGANs can produce high-resolution photorealistic symbolism without utilizing hand-crafted misfortunes or pre-trained systems. Some datasets contain more information like class labels [10] [24].

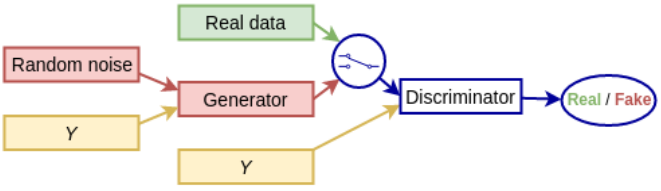


Fig. 4. Architecture of CGAN

E. VSGAN

The Vehicle Synthesis-GAN is identical to the DCGAN, but it has a distinct design. One generator and two discriminators-one for the cars and one for the backdrop were used. Random noise surrounds the automobile on the ground. The picture and noise are sent into the generator. The generator aims in studying the mapping of function $F : x \rightarrow y$, where

y- ground truth image and x- picture with input noise. The residual block is used to avoid gradient loss. The background discriminator (D_b) is used to learn the context information of the backdrop, and is utilized to integrate the generated picture onto the background elegantly. The vehicle discriminator (D_v) is used to differentiate between positive and negative vehicles. Positive pictures are genuine with ground truth images, while the generator makes the negative ones. Loss functions are computed mathematically [1] [25] [26]. The residual blocks causes a limitation in the number of samples that can be taken as input. Removing one or two blocks will allow for more input of samples, although the images generated will be lower in quality and accuracy. The network will become shallower and less features of the images will be extracted. The models are all listed below:

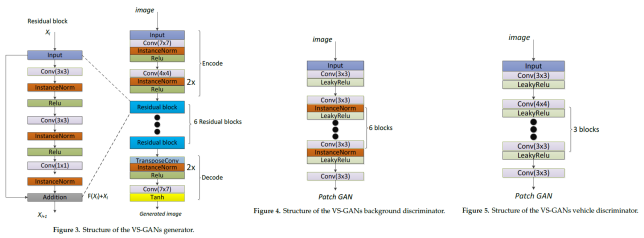


Figure 3. Structure of the VS-GANs generator.

Figure 4. Structure of the VS-GANs background discriminator.

Figure 5. Structure of the VS-GANs vehicle discriminator.

Fig. 5. Architecture of VSGAN

F. Generator

The generator of a GAN learns to generate fake data through integration of discriminator feedback. It studies to manipulate the discriminator to categorize output as true [27] [28] [29]. The generator is trained as follows: Input at random, random input is converted into a data instance by the generator network, the Discriminator, which organizes the discriminator output supplied by discriminator, generator is punished by Generator loss for failing to deceive the discriminator. For acquiring gradients, back propagation via both networks occur. Generator weights are adjusted using gradients. This is one generator training iteration.

G. Discriminator

It attempts to discern between actual data and generated data. It might utilize any network model suitable for the type of data it's categorizing [27]. While training, the discriminator ponders the scenarios as positives. During training, it utilizes the situations as negatives. During discriminator training, generator training halts, the weights stay similar, when generating samples for the discriminator to learn from. The discriminator distinguishes between real and fake data throughout discriminator training. The discriminator loss punishes the discriminator for incorrectly categorizing a true image being fake or a fake image being true. Back propagation from discriminator's loss via the discriminator network is used to update the discriminator's weights.

H. Layers

In any Neural Network or Generative Adversarial Network, the layers are used to make up the whole network. In the architectures, many types of layers along with their details, features and conditions are given to construct the model. All the GAN architectures use the Convolution-layer and Transpose-layer. As such they are discussed below in detail.

1) Convolution Layer: A CNN's main building block is a convolution layer. Throughout training the parameters must be learned [30]. The filters sizes are smaller than the image size. They manage spatial redundancy through weight sharing. This mapping function no longer requires sharing weight as complete feature vector is required to build an informed conclusion. Convolution feature extractors learned features are usually transformed into a vector that can be utilized as an image descriptor.

2) Transpose Layer: Transposed convolutions do not reverse the standard convolution by values, but by dimensions only. The transposed convolution layer performs the same functions as a regular convolution layer but on different input feature-map. It is used for up-sampling or producing a yield map of features with a spatial area higher than input feature map. The stride and padding define the transposed convolution layer in the same way [31]. However, there is a checkerboard problem that can be resolved using up-sampling.

I. ReLU

ReLU, an activation function of non-linearity. ReLU stands for "Rectified Linear Unit". With input x, such as a matrix from a convoluted image, ReLU is, $maxfunction(x, 0)$. ReLU sets all negative values in matrix x to 0, and the other standards remain constant. It's a piecewise linear function, if it is positive, it will output the input directly, else 0. It is the default activation for many neural networks.

1) Leaky ReLU: Another type of activation function, based on ReLU. To solve the "dying ReLU" problem, zero-slope portions are absent. It has been found that "mean activation" would be close to 0, which helps to make the training faster. The main advantage of leaky ReLU compared to ReLU is, in this method we cannot have any vanishing gradient. In the Leaky ReLU, the output slope for negative inputs is a learnable parameter. It's referred to as a hyper-parameter, this is the only difference between them.

J. Norms

Norms stand for normalization. Normalization is the process of converting all the data in range of 0 or 1 or perhaps also any other range. Some algorithms benefit from this process-especially when Euclidean distance is involved. In all the architectures above we have used batch-norm or instance-norm, as such these two are given below.

1) Batch-Norm: Batch-Normalization,has the ability to improve model performance while simultaneously cutting down on training time. This is mostly dependent on its ability to smooth the optimal landscape by increasing the Lipchitz of the loss gradient. Batch-Norm is a method for improving neural

network training by stabilizing layer input distributions. This is accomplished by adding extra network layers to manage its first two moments of these distributions (mean and variance) [32].

2) Instance-Norm: Instance normalization is a special case of group normalization because it normalizes all the characteristics of the channel. The group size corresponds to the channel size. Empirically, when the learning rate is adjusted in proportion to the batch size, its accuracy is more stable than the batch standard in a wide range of small batch sizes. Substituting batch normalization with instance normalization improves the execution.

K. Sigmoid

The sigmoid function is a function that has a sigmoid shape to adjust the dynamic range of an image, this is a point process that is conducted directly on each pixel of that image, irrespective of all other pixels in that image. The sigmoid function is a nonlinear activation function that is continuous.

L. Tanh

The hyperbolic tangent function can be utilized as an alternative to the 'Sigmoid' function. For calculating the error effects on weights the tanh derivative is utilized. The derivative of the hyperbolic tangent function, like the sigmoid function, has a simple form. This explains why neural networks frequently use hyperbolic tangents [33] [34].$\tanh(a) = (ea - e - a)/(ea + e - a)$. The range of tanh is [-1 to 1] and the range of the derivative of the function is 0 to 1.

M. Gradient Penalty

While the original WGAN improves the stability for training, there are situations where the GAN does not converge. WGAN-GP enforces the Lipschitz continuity, with a gradient norm constant on the critic, replacing weight clipping. This increases training stability, more than the WGAN and it does not require the hyper-parameters to change too much. The issues with weight clipping is that it learns simple functions and does not capture higher moments, gradient penalty fixes this issue. Weight clipping may result in vanishing or exploding gradients as it pushes the weights of the critic to extreme clipping values. Batch Normalization is not used for the critic anymore as batch norm maps the input batches to output batches [35]. This is used in the WGAN-GP and CGAN. Though in CGAN we also pass the labels to the gradient penalty.

N. Spatial Pyramid Pooling

Existing neural networks need input of a fixed size image . The artificial requirement can lessen the accuracy of recognition for images of erratic sizes or scales. For fixing the issue, Spatial Pyramid Pooling (SPP), used to generate any representation of fixed length regardless of scales or sizes. It can calculate the feature map of the full picture just one time, then features are pooled in erratic regions, making a representation of fixed length in training. The convolution layers accept any size, but to make outputs of various sizes, either classifiers $(SVM/softmax)$ or layers that fully connect may be of any size. It permits various scales or ratios. The image can be resized to any scale and the same neural network applied [36]. This is used in the VSGAN.

O. Weight Clipping

Gradient clipping deals with explosive gradients. Explosive gradients, during training the gradients get too large and makes the training unstable. Likewise, when gradients get too small– it is called vanishing gradients. This obstructs the network from changing their weight values, which makes the network unable to learn the data. Gradient clipping disrupts the exploding gradients. If a gradient gets too large, it is re-scaled to keep its value small. If $||g|| \geq c$ then $g \leftarrow c.g/||g||$ where, c- hyper-parameter, g- gradient and $|g|$ is the norm of g. After re-scaling c is the norm g will have. It makes sure that g has a norm of at-most c [37]. This is utilized in WGAN.

P. Optimizers

When we train the models we need to ensure modification of the weight of each epoch and minimize the loss function. An optimizer is either a function or an algorithm that modifies an attribute of a neural network, like learning rate or weights. Thus, this increases accuracy and reduces the loss. All models use Adam optimizer with the exception of WGAN that uses RMSProp optimizer. The two optimizers are discussed below.

1) Adam: It can be utilized to update iterative based network weights in the training data. Adam is a combination of advantages of two other additions of Stochastic Gradient-Descent. Adaptive Gradient Algorithm– improves the execution on issues that have 'sparse' gradients by keeping the learning rate that is per parameter and Root Mean Square Propagation– keeps learning rates per perimeter which are adapted, fixed on mean of new volume of the weight gradients. Adam adopts the parameter learning rates on mean of first and second moments of the gradients. The decay rates of dynamic means are controlled by Beta1, Beta2. By computing the 'biased estimates' first then computing 'bias-corrected estimates', bias gets defeated. Adam is the best as it achieves good results quickly. Adam is easier for configuration, and the given parameters work well on most issues [38]. All the GAN architectures uses Adam optimizer except for WGAN.

2) RMSProp: Root Mean Squared Propagation, is the extension of both its AdaGrad version and gradient descent, which uses the decaying mean of partial gradients in adaptation of size of step per parameter. Using a moving decaying mean ensures the algorithm forgets its previous gradients and focuses on new ones, thus overcoming AdaGrad's limitations. It was invented to make the process of optimization faster. This causes vibrations to smooth out in problems with bigger curvatures. The issue with AdaGrad, it slows down the searching, which results in learning rates that are very small for every parameter. For root mean square (RMS) – $custStepSize = stepSize/(1e-8+RMS(s(t+1)))$. Then update the perimeter as given $x(t+1) = x(t) - custStepSize(t+1) * f'(x(t))$ which is looped per input variable till a new point is created for evaluation is the search space [39]. RMSProp is only used in WGAN.

Q. Criterions

Criterions are used in GANs to measure- how close the generated images are to the original images. They act as a stopping criteria for when to stop converging. In case of WGAN, WGAN-GP and CGAN the generator loss can be used as a stopping criteria. The DCGAN uses only BCE-Loss while VSGAN uses BCE-Loss, MSE-Loss and L1-Loss. All the loss function are given below.

1) BCE-Loss: Binary Cross Entropy (BCE) is a loss function that is often utilized in binary organization tasks. Binary duties only respond questions with either a yes-[1] or no-[0]. It can give answers to several independent questions at the same time, like in multi-label classification.

$$\begin{aligned} &\ell(x,y) = L = \{l_1,...,l_N\}^T, \\ &l_n = -w_n[y_n.logx_n + (1-y_n).log(1-x_n)] \end{aligned} \tag{1}$$

Here, x-input, y-target, N-batch-size. The target y should be numbers between 0 and 1. Utilized to measure the loss of a reconstruction. If x_n is either 0 or 1 then automatically either one log term would become undefined. The solution is to make BCE-Loss clamp its log function outputs to be $\geq$ to -100. It is equal to the average result of the Categorical Cross Entropy loss function applied to a lot of independent classification problems, where each problem has only two classes with targets being y_n and $(1-y_n)$. The only activation function that is compatible with this loss function is the 'Sigmoid' [40].

2) MSE-Loss: Mean Squared Error is often utilized in calculating losses for regression. The loss is the mean squared supervised data squared deviation between the real and anticipated values. The average of the square difference between the true result and anticipated result are measured by the MSE-Loss process.

$$\ell(x,y) = L = \{l_1,...,l_N\}^T, l_n = (x_n - y_n)^2 \tag{2}$$

Here, x-input, y-target, N-batch-size. Here x and y are erratic shaped tensors which has n elements total each. The sum operates above all elements and divides by n. To avoid n division [41].

3) L1-Loss: Least Absolute Deviations is utilized in machine learning to reduce the mistakes. It diminishes error – which is the addition of all absolute deviations between predicted and true results. Outliers do not affect L1.

$$\ell(x,y) = L = \{l_1,...,l_N\}^T, l_n = |x_n - y_n| \tag{3}$$

Here, x-input, y-target, N-batch size. Here x and y are erratic shaped tensors with total of n elements each. The sum operation works on all elements and divides by n. This supports both real-valued and complex- valued inputs [42].

R. Hyper Parameters

A model hyper-parameter is a setting that is external to the model, where the values cannot be predicted from data. Often used in processes to estimate the parameters of the model.

1) Epochs: A hyper-parameter which tells how many times the algorithm will go through the whole dataset. An epoch consists of a single or many batches. An epoch of a single batch is coined as batch gradient descent learning. It is like a 'for' loop over epoch numbers proceed through the whole training-set. Another nested 'for' loop within this loop, goes through each of the batches of samples. One batch has a 'batch-size' assigned. The batch-size is the number of samples. To graphically display this data where $y - axis$ shows error or skill of the model and the $x - axis$ shows the time. They are called learning curves [43]. We have used 40 epochs for all the GANs.

2) Batch Size: It defines, how many samples it has to go through, before the internal model parameters are updated. It is a 'for' loop that iterates above many or just one sample, makes predictions. At the end of the batch, the expected output variables and the predictions are compared. The error is computed. Using the mistakes, the updating algorithm is utilized to make the model better, going downward along the error gradient. Sometimes batches do not divide evenly, removing some samples is a good solution here [43]. We have used a batch size of 15 for all the GANs.

S. GAN Loss Functions

GANs copy distribution of a probability. To reflect the distance of the distribution of generated and real datas, loss function is used. GAN loss function is a very broad experimentation and various ideas have been put forward. Some GANs have more than one loss function.

1) DCGAN Loss Function: The conventional GAN loss function, often known as the $Min - Max$ loss. The fundamental problem is that the model necessitates the training of two networks at the same time: generator and discriminator. Although, non-saturating loss function is commonly used, the GAN network is defined by the $Minimax$ GAN loss. The $least-squares$ and $Wasserstein$ loss functions are common alternates utilized in current GANs.
For Discriminator:

$$maxlog(D(x)) + log(1 - D(G(z))) \tag{4}$$

where $D(x)$- discriminator with real image and $D(G(z))$ - discriminator with generated image.
For Generator:

$$minlog(1 - D(G(z))) \leftrightarrow maxlog(D(G(z))) \tag{5}$$

where $D(G(z))$- discriminator with the generated image [8].

2) VSGAN Loss Function: The VSGAN consists of two procedures of adversarial learning- $G \leftrightarrow Db$ and $G \leftrightarrow Dv$. Procedure between G, D_b:

$$\begin{aligned} L(G,D_b) &= E_{y\sim p_{gt.image(y)}}[(D_b(y)-1)^2] \\ &+ E_{x,z\sim p_{noise.image(x,z)}}[(D_b(G(x,z)))^2] \end{aligned} \tag{6}$$

Here, y- the ground truth image and x- the image with noise box. Least-squares loss of LSGAN was used here instead of the original GAN loss. To generate realistic images of vehicles

using G in the noise box z of input image x, the second adversarial training process is executed between G, D_v.

$$\begin{aligned} L(G, D_v) &= -E_{y_v \sim p_{vehicle(y_v)}}[D_v(y_v)] \\ &+ E_{z \sim p_{noise(z)}}[D_v(G(z))] \\ &+ \lambda E_{z \sim p_{noise(z)}}[(|V_z D_v(z)|_2 - 1)^2] \end{aligned} \tag{7}$$

Here Y_v- the cropped vehicle from the ground truth image y, z- the noise box in image x. To balance the training procedure the strategy of gradient penalty of WGAN is used. L1 loss is used for balancing the difference between the generated and ground truth images.

$$L_{\ell_1}(G) = E_{x,z \sim p_{noise.image(x,z)}, y \sim p_{gt.image(y)}}[|y - G(x,z)|_1] \tag{8}$$

By summing all the previous defined losses given above, the final loss is calculated. Here, to control L1 loss function λ is the hyper-parameter. The final equation is given below:

$$L(G, D_b, D_v) = L(G, D_b) + L(G, D_v) + \lambda L_{\ell_1}(G) \tag{9}$$

All equations are given above as per the VSGAN-paper [1].

3) WGAN Loss Function: It necessitates a movement in thinking from a discriminator that forecasts the likelihood of a fake image being true to a critic model which rates the trueness of the image. To increase the difference between fake and real picture rankings, the Wasserstein loss function has been utilized.

Critic Loss = [average critic score on real images] – [average critic score on fake images]

Generator Loss = - [average critic score on fake images]

Computations are easy to comprehend as Stochastic Gradient Descent aims to minimize loss.

4) WGAN-GP Loss Function: WGAN-GP loss improves signal and noise transmission characteristics of a CNN based on VGG loss. It improves the attenuation execution of the CNN based on VGG loss, which is consistent with the qualitative assessment. With gradient penalty and Wasserstein distance, the WGAN-GP model modifies regular GANs [44] [45].

5) CGAN Loss Function: Combining the GAN loss function of the generator with traditional loss functions, like the architecture of the pixel-to-pixel (Pix2Pix). Where the loss function is upgraded by adding L1 loss function.

$$\begin{aligned} LcGAN(G, D) &= E_{x,y}[log(D(x, y))] \\ &+ E_{x,z}[log(1 - D(x, G(x, z)))] \end{aligned} \tag{10}$$

Where z- random noise vector, y- ground truth image, x- input picture, D- discriminator and G- generator. The CGAN loss function varies, the discriminator in CGAN observes the inputs.

$$L_{GAN}(G, D) = E_y[log(D(y))] + E_{x,z}[log(1 - D(G(x, z)))] \tag{11}$$

Where $log(D(x))$- the probability of generator accurately identifying actual images, maximizing $log(1 - D(G(z)))$ [46] [47].

IV. Dataset

We are using the Stanford Cars Dataset [48]. It contains 16185 car images of 196 different classes. This data is already preprocessed and split into 8,041 testing images and 8,144 training images. Each class has a rough 50-50 split.

Fig. 6. Dataset Images

From this dataset we have decided to use 6000 images for both training and testing. There are some issues with some of the pictures past 6000 images.

A. Training Set

We use the training set, so our models can learn the behaviors of the data. We keep the hyper-parameters constant for all architectures. We train the models, so they can understand the data and the various behaviors of the data. If we do not train our GANs, they cannot learn the labels or features and thus will not be able to generate good images.

B. Testing Set

To ensure the data we got from the GANs are valid. We use the testing set to solidify our results. We know, both the testing and training data are slightly different, if the results are close or follow the same patterns, the architectures were trained accurately and the results obtained are valid. This will give us better confidence in the models capabilities.

C. Architecture Training

To train and test the architectures, we have used two methods. For the CGAN, DCGAN, WGAN and WGAN-GP, we have used Google Colaboratory. The run-time type was set to GPU. For the VSGAN, we had to utilize our laptop. The VSGAN was different and took inputs using a different method. It was too difficult to run it in Google Colaboratory. The specifications of the laptop used CPU: Intel Core i5 – 8265U, RAM: 8 GB, GPU: Intel ® UHD Graphics 620. We used these two methods to save time and ensure the research was moving forward without delay.

V. Training and Testing of All Models

We have simulated the models and noted their training and testing times, generator and discriminator losses, accuracy via various methods, and finally a comparison of the training and testing generated images of all the models.

A. Comparison of Training and Testing Times

The parameters of all models are - 40 epochs and 15 sample size. 6000 images for both in all GANs with the exception of the VSGAN were used.

TABLE I
TRAINING AND TESTING TIMES

GAN-Model	Training	Testing
CGAN	2 hours 10 minutes	2 hours 6 minutes
DCGAN	42 minutes 4 seconds	41 minutes 24 seconds
VSGAN	1 hour 55 minutes	1 hour 52 minutes
WGAN	2 hours 16 minutes	2 hours 14 minutes
WGAN-GP	2 hours 7 minutes	2 hours 2 minutes

DCGAN processed the information the fastest as it is not too deep. The VSGAN, despite being a deeper network is fast though not as fast as the DCGAN. The WGAN-GP is slower due to the time required to calculate the gradient penalty. The CGAN is also slow due to the classes, which increases processing time. WGAN is the slowest as the Wasserstein-Loss function of the WGAN slows the process down and trades the faster processing for more stable GAN training. Finally, regarding the speed and time required to train and test the models: DCGAN, VSGAN, WGAN-GP, CGAN, and WGAN.

B. Comparison of Generator and Discriminator Losses

1) CGAN: The CGAN training and testing losses are discussed in detail.

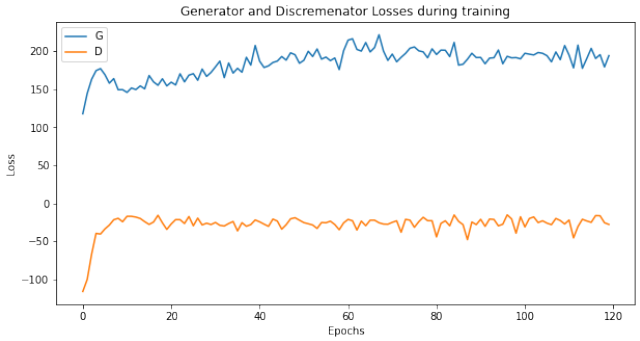


Fig. 7. CGAN Training Loss

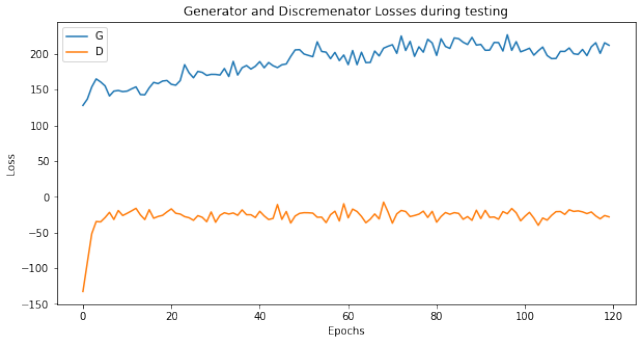


Fig. 8. CGAN Testing Loss

As the number of epochs kept increasing, there are signs of diverging instead. Near the end of the last 10-15 epochs, it started to diverge. The CGAN took a considerable amount of time to train and test. If we ran the CGAN for more number of epochs there may be a chance of convergence or divergence. The CGAN can extract surface features, unlike the VSGAN, cannot extract deep features. The CGAN can produce images of decent quality, though not very sharp.

2) DCGAN: The DCGAN training and testing losses are discussed in detail.

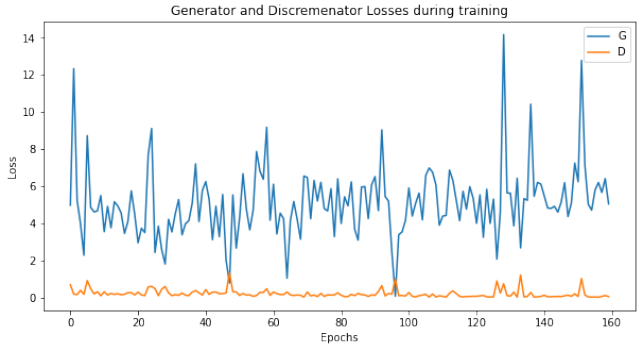


Fig. 9. DCGAN Training Loss

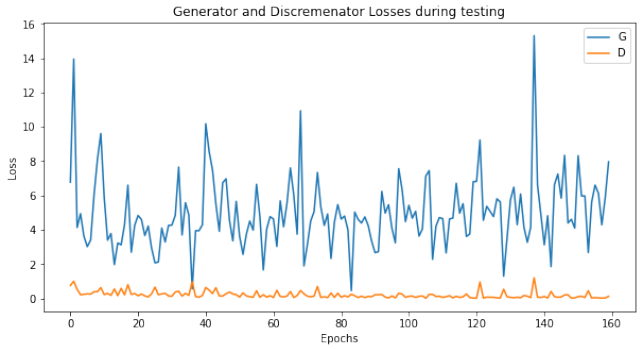


Fig. 10. DCGAN Testing Loss

There is a high chance of convergence. The images produced from the DCGAN were also of high quality. This shows how powerful the DCGAN is compared to the others, though not much compared to the VSGAN. There are various points of convergence and divergence. If we keep increasing the epochs of the DCGAN we can get higher quality images.

3) VSGAN: The VSGAN training and testing losses are discussed in detail.

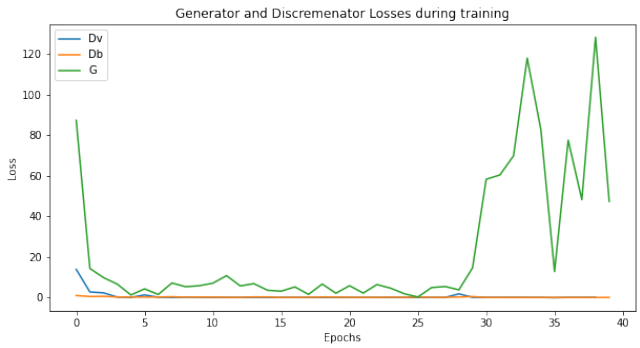


Fig. 11. VSGAN Training Loss

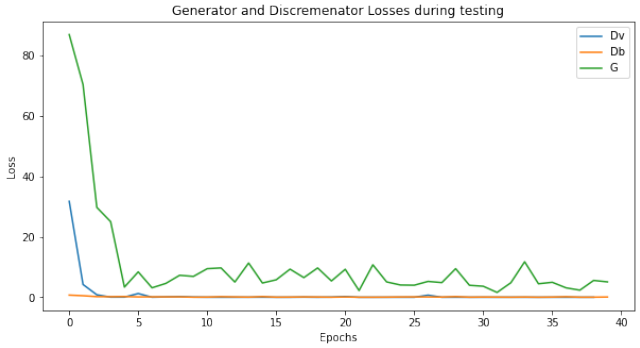


Fig. 12. VSGAN Testing Loss

We have ignored D_b- discriminator background loss as it stays constant. In the training loss, from 0 to 27 epochs, it was converging well, then suddenly it diverged quickly from epoch-28. This was not seen in the testing phase, which shows clear convergence from 0 to 40 epochs. A resolution of 160 and cropping values of 140 were used. Sharper and larger images produce better results but takes longer time. The VSGAN works best only for a low number of samples. 40 random samples from our 6000 train and test datasets were used. When the sample number increases there is a high chance of mode collapse. The VSGAN's impressive ability to extract deep features from a small number of samples allows such high quality images to be generated.

4) WGAN: The WGAN training and testing losses are discussed in detail.

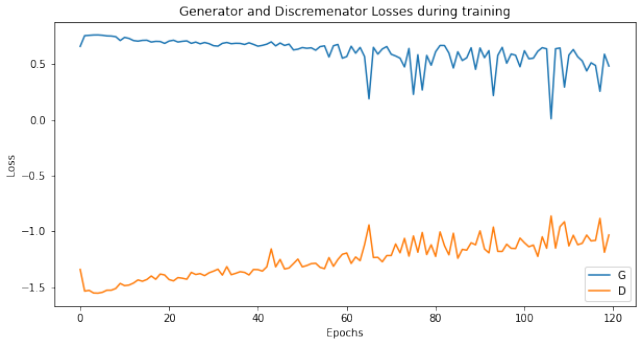


Fig. 13. WGAN Training Loss

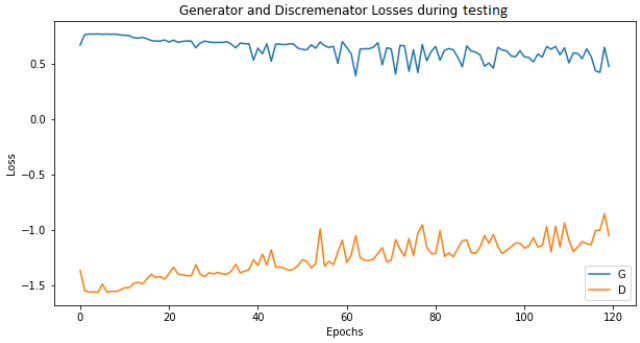


Fig. 14. WGAN Testing Loss

As WGAN goes through the epochs it starts to converge. However, to get close to convergence it needs to run for a large number of epochs, which will take a lot of time. WGAN is the slowest of all the models. The WGAN can generate relatively good quality images, although they are not sharp. The WGAN is very stable and can be trained for a long time, but at the cost of processing speed.

5) WGAN-GP: The WGAN-GP training and testing losses are discussed in detail.

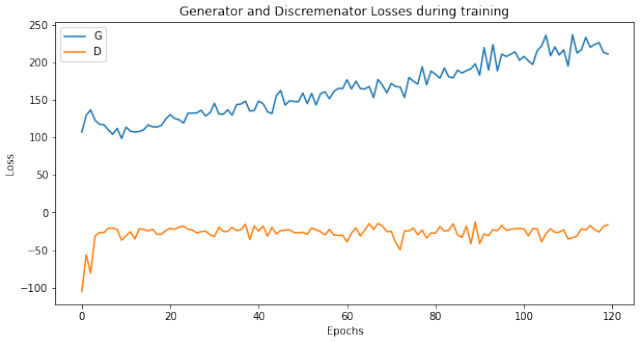


Fig. 15. WGAN-GP Training Loss

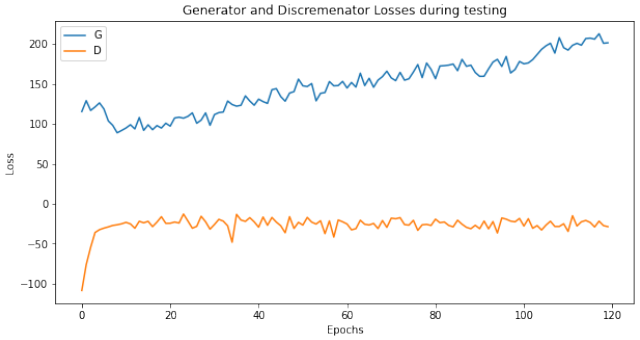


Fig. 16. WGAN-GP Testing Loss

It starts to converge after going through some epochs, but halfway, it begins to diverge. It will take a considerable amount of time and epochs for convergence. There is also an equal chance divergence. The WGAN-GP can produce good, sharp images and the quality is relatively good. The gradient potential helps in greater feature extraction, however- it takes a longer time to process information.

C. Comparison of Accuracies

In GAN literature, accuracy is not often measured much. There is no universal metric that will work in calculating the accuracy of the generated images of various GANs. Most authors use object detection algorithms - YOLOv3, R-CNN, Inception v3 or SSD. In our research, the real and fake scores of the discriminators were used. Before using any algorithm, the data of all models were reshaped. All data was given to the algorithms for accuracy calculation. We have utilized three different accuracy algorithms which are given below.

1) K Nearest Neighbors-KNN: K-Nearest Neighbors is utilized in classification and regression, in supervised Machine-Learning. It depends on labeled inputs to map the functions, making an output, when unlabeled data is given. KNN works through finding distances between all samples in the data and a group of data, selects the given number of samples- k closest to the group, then chooses the highest repeated label-(classification) or averages the label-(regression) [49].

TABLE II
KNN ACCURACIES OF ALL ARCHITECTURES

GAN-Model	Train Accuracy(%)	Test Accuracy(%)
CGAN	83.3	83.3
DCGAN	62.5	56.2
VSGAN	100	100
WGAN	58.3	50
WGAN-GP	75	83.3

KNN is used on both the train and test data of each model. VSGAN has the highest accuracy, although it shows 100% in both training and testing, the actual accuracy is somewhere between 98-99%. We see that in training and testing the CGAN and VSGAN remains the same. The CGAN has an accuracy of 83.3% in both cases. The DCGAN, WGAN and WGAN-GP shows fluctuation in training and testing. The WGAN-GP has a 75% accuracy in training and 83.3% in testing signifying an improvement, whereas in the DCGAN and WGAN accuracy reduces during testing. The accuracy reduction may be a result of not extracting enough features during training or the testing samples were too random.

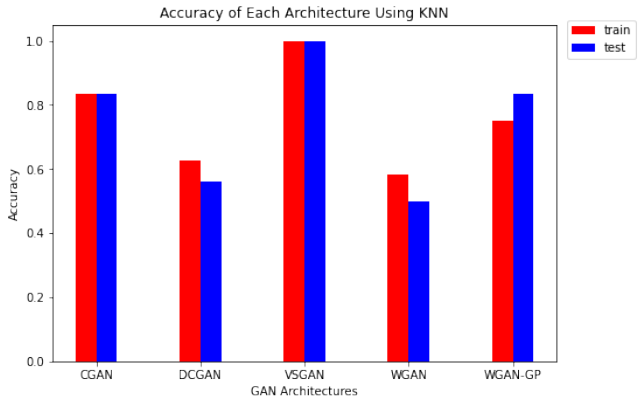


Fig. 17. KNN Accuracy of All Models

In terms of accuracy overall – VSGAN, CGAN, WGAN-GP, DCGAN and WGAN. This is a very good measure and is very accurate. As per the authors, our accuracy results using KNN do match some of the implications. Although, KNN is a good measure, it should be noted that inconsistencies can be seen in some cases.

2) Regression: Linear regression is used for making predictions. It utilizes previous data and predicts an output variable. There are two variables- the input or predictor variable that will assist in predicting the output. The output variable – the value we wish to predict. The equation, $Y_e = \alpha + \beta x$ where Y_e - the predicted value of Y, our objective, to find α and β values. To find the values we use ordinary least squares method [50].

TABLE III
REGRESSION ACCURACIES OF ALL ARCHITECTURES

GAN Models	Accuracy(%)
CGAN	92.5
DCGAN	95
VSGAN	97.5
WGAN	90.8
WGAN-GP	93.3

The algorithm calculates total accuracy. VSGAN shows the highest accuracy. There is also much difference in accuracy for the others. DCGAN shows an accuracy of 95% which is much higher. Same for CGAN, WGAN and WGAN-GP as well. WGAN shows an accuracy of 90.8% more than the previous computation. These readings are close to both the authors' implications and our hypothesis. Usually CGAN, WGAN and WGAN-GP – with its higher classification, Wasserstein loss function and feature extraction capabilities respectively, should have a higher accuracy than DCGAN.

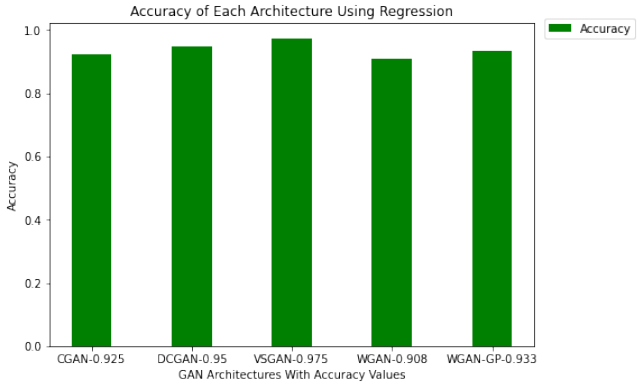


Fig. 18. Regression Accuracy of All Models

In terms of accuracy – VSGAN, DCGAN, WGAN-GP, CGAN and WGAN. Even though the accuracies match our hypothesis, there are some conflicts. There may be some inconsistencies though, but overall, this is also a good measure.

3) Random Forest-RFC: Random Forest Classifier is a supervised learning algorithm and is utilized in classification and regression. On randomly selected data, random forest makes decision trees and gets a forecast from each tree. It selects the best answer through casting a vote. It is highly accurate and can handle missing values. Though it is slow to generate predictions as all trees must provide a prediction which takes time [51].

TABLE IV
RFC ACCURACIES OF ALL ARCHITECTURES

GAN-Model	Train Accuracy(%)	Test Accuracy(%)
CGAN	100	100
DCGAN	81.2	75
VSGAN	100	100
WGAN	91.7	83.3
WGAN-GP	91.7	100

Both the CGAN and VSGAN show 100% accuracy, the actual accuracy is close to 97-99% for both cases. Both the WGAN and WGAN-GP shows an accuracy of 91.7% during training, however, its difference for testing, in WGAN it decreased to 83.3% while for WGAN-GP it increased to 100%, though it may be 97-99%. This indicates that WGAN-GP can increase the accuracy through gradient penalty. For the DCGAN it was at 81.2% and decreased to 75%, most likely due to too many random samples, or few good predictions. Random Forest shows that DCGAN has less accuracy compared to others.

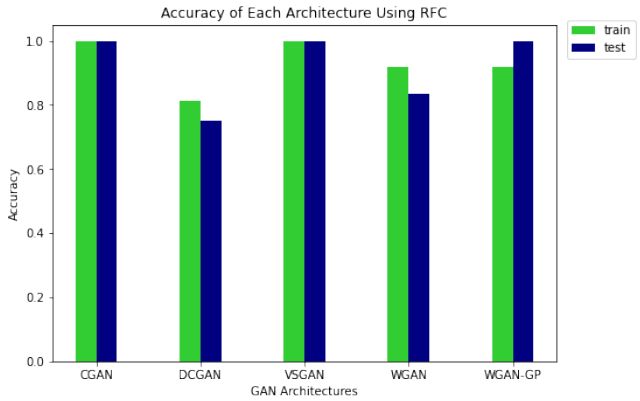


Fig. 19. RFC Accuracy of All Models

In terms of overall accuracy – VSGAN, CGAN, WGAN-GP, WGAN and DCGAN. This is a good measure, though difficult to interpret. It is stated that VSGAN is the most accurate, followed by CGAN, WGAN-GP, WGAN and DCGAN. DCGAN is less accurate as it is mostly random and unsupervised. It relatively matches both ours and the authors' implications.

D. Comparison of Generated Images

All models generated fake images of various qualities. We have put all the fake training and testing images along with the real training and testing images, per model, to compare each models generative capabilities. All the data is given below.

1) CGAN: The data table of CGAN.

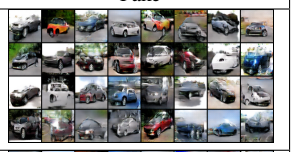
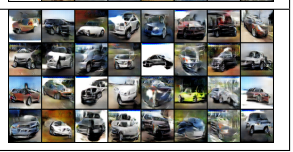
TABLE V
CGAN IMAGES

Data	Real	Fake
Train		
Test		

The fake images are clear, but distorted. We can distinguish the cars in the pictures, but some are blurry. The CGAN took considerable amount of time to generate images. Although unlike WGAN the images are distinguishable, but unlike VSGAN the images are not very sharp. The images are accurate. If we ran for a large number of epochs, we believe the image quality will improve but it will take a considerable amount of time. The number of fake and real images are based on batch size and we gave a batch size of 15.

2) DCGAN: The data table of DCGAN.

TABLE VI
DCGAN IMAGES

Data	Real	Fake
Train		
Test		

The fake images have high quality, but they are mixed and jumbled. We can see the vehicles but not as clearly as we can see them in the CGAN, VSGAN and WGAN-GP. If we ran the DCGAN for more epochs we can generate even sharper images. The images obtained are of decent quality and we can say the DCGAN can generate good images in a short amount of time although they are not very accurate as we compare them with the real samples. Here fake images are generated as per grid value - 32 and real images based on batch-size value 15.

3) VSGAN: The data table of VSGAN.

TABLE VII
VSGAN IMAGES

Data	Real	Fake
Train		
Test		

The fake images have the highest quality and accuracy. If we ran it for more epochs we may have gotten slightly sharper images or in the worst case scenario - mode collapse. The VSGAN can generate high quality images in a short amount of time, though not as short as the DCGAN. The images have much higher quality and accuracy as compared to the DCGAN, WGAN-GP, and CGAN. We have given number of rows-8 and dataset loader value-16, thus 16 images in 8 rows are shown for both real and fake samples.

4) WGAN: The data table of WGAN.

TABLE VIII
WGAN IMAGES

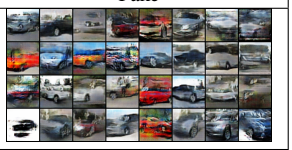
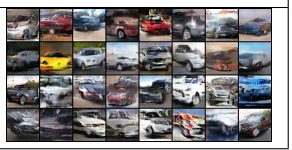
Data	Real	Fake
Train		
Test		

WGAN generates images of decent quality, but it is difficult to distinguish them. We can see some resemblances of cars,

but some of the images are blurry and distorted. We may have to run the WGAN for a lot more epochs to get similar quality images of CGAN and a much larger number of epochs to get the quality close to the VSGAN. The images are clear and shows some accuracy. Despite WGAN being stable, it takes a considerable amount of epochs and time to get the model to generate distinguishable images. Images written on both grids are based on batch-size value 15.

5) WGAN-GP: The data table of WGAN-GP.

TABLE IX
WGAN-GP IMAGES

Data	Real	Fake
Train		
Test		

WGAN-GP generates good quality images, even though they are distorted and blurry. We can see the images quite clearly and they are accurate. The images are close to the generated images of the CGAN. It is similar to WGAN, but more accurate, thanks to the gradient penalty. Though not as accurate as the CGAN. If we trained and tested for more epochs, we may have gotten more accurate images, although it would take a lot of time and as the epochs increases the processing time of the gradient penalty also goes up. The fake images are written based on grid value - 32 while the real ones based on batch size value - 15.

VI. Conclusion And Future Work

A. Conclusion

Today, vehicle image generation is a fundamental research. It has many applications of which searching, identification and security are most common. We need to use the right tool at the correct time and purpose. We hope to increase the information available on the VSGAN and for everyone to become more aware of it. We have compared the training and testing times of all the GANs. With a sample size of 6000 training images and another 6000 testing images. The other hyper-parameters were kept constant for all models. We have also compared the losses of both the generators and discriminators of all the models to see how quickly they converge and by how much in both training and testing. We have also compared the accuracy of all models in both cases as well. We entrust our results will enlighten the scientific community with our findings. We believe increasing the information on the VSGAN will increase its use. Finally, this research aims to provide comparative data on the models used, increase interest in GANs and GAN-research all together.

B. Future Work

In our future-work, we hope to utilize these models and test them in different environments, use different datasets, and change more variables, hyper-parameters and image resolutions. We also wish to change functions within the architectures and see what results can be procured such as, adding or removing Sigmoid, Tanh, ReLU, LeakyReLU etc. We may also want to improve the comparisons by including more datasets with much larger sample sizes. We may also wish to generate different images like-birds, planes, buildings and trees etc. We can also try to see if orientation, object-distance, classification, image-labels, light-levels and color palettes, affect our architectures and by how much. We also wish to research the idea of removing one or two residual layers of the VSGAN, to increase the sample size and assess the quality, accuracy of the generated images. Future research has a lot of potential. We hope to research further and increase information on the given models. Lastly, we hope to pave a new era of intensive GAN research in the scientific community.

References

[1] K. Zheng, M. Wei, G. Sun, and et al, "Using vehicle synthesis generative adversarial networks to improve vehicle detection in remote sensing images," *[J].International Journal of Geo-Information*, vol. 8, no. 9, p. 390, 2019.

[2] Y. Zhao and L. Shao, "Cross-view gan based vehicle generation for re-identification," *The British Machine Vision Association and Society for Pattern Recognition*, pp. 186.1–186.12, 2017.

[3] L. Bashmal, Y. Bazi, H. Alhichri, M. Alrahhal, N. Ammour, and N. Alajlan, "Siamese- gan: Learning invariant representations for aerial vehicle image categorization," *Multidisciplinary Digital Publishing Institute*, vol. 10, no. 2, p. 351, 2018.

[4] R. Dinakaran, L. Zhang, and R. Jiang, "In-vehicle object detection in the wild for driverless vehicles," *arXiv*, 2020.

[5] K. Lv, H. Sheng, Z. Xiong, W. Li, and L. Zheng, "Pose-based view synthesis for vehicles: A perspective aware method," *IEEE Transactions on Image Processing*, vol. 29, pp. 5163–5174, 2020.

[6] N. Joshi, "5 applications of generative adversarial networks," 2020. [Online]. Available: https://www.allerin.com/blog/5-applications-of-generative-adversarial-networks

[7] J. Brownlee, "18 impressive applications of generative adversarial networks (gans)," 2019. [Online]. Available: https://machinelearningmastery.com/impressive-applications-of-generative-adversarialnetworks

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv*, 2016.

[9] S. Chintala, "Dcgan.torch. train your own image generator," 2016. [Online]. Available: https://github.com/soumith/dcgan.torch

[10] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv*, 2014.

[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv*, 2017.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv*, 2017.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation.,"," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016.

[14] R. Girshick, "Fast r-cnn," *arXiv*, pp. 1440–1448, 2016.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[16] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *arXiv*, 2017.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, Y. Fu, and et al, "Ssd: Single shot multibox detector," *Computer Vision European Conference on Computer Vision*, vol. 9905, pp. 21–37, 2016.

[18] D. Shan, I. Mahmoud, and et al, "Automatic license plate recognition (alpr): A state-of-the-art review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 2, p. 311–325, 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6213519

[19] H. Kim, D, "Deep convolutional gans for car image generation," *arXiv*, 2020.

[20] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deep driving: Learning affordance for direct perception in autonomous driving," *IEEE International Conference*, pp. 2722–2730, 2015.

[21] J. Shen, N. Liu, H. Sun, and H. Zhou, "Vehicle detection in aerial images based on lightweight deep convolution network and generative adversarial network," *IEEE*, vol. 7, pp. 148 119–148 130, 2019.

[22] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. Efros, A, "Context encoders: Feature learning by inpainting," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[23] P. Kingma, D and W. Max, "Auto-encoding variational bayes," *arXiv*, 2014.

[24] I. Goodfellow, J. Pouget-Abadie, and et al, "Generative adversarial nets. in advances in neural information processing systems," *NeurIPS Proceedings*, p. 2672–2680, 2014.

[25] H. Lyu, "Automatic vehicle detection and identification using visual features," *Electronic Theses and Dissertations*, p. 7378. [Online]. Available: https://scholar.uwindsor.ca/etd/7378

[26] R. Krajewski, T. Moors, and L. Eckstein, "Vegan: Using gans for augmentation in latent space to improve the semantic segmentation of vehicles in images from an aerial perspective," *IEEE Winter Conference on Applications of Computer Vision*, 2019.

[27] Google-Developers, "Generative adversarial networks - gans - the generator." [Online]. Available: https://developers.google.com/machine-learning/gan/generator

[28] J. Rocca, "Understanding generative adversarial networks (gans)," 2019. [Online]. Available: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

[29] J. Brownlee, "A gentle introduction to generative adversarial networks (gans)," 2019. [Online]. Available: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans

[30] S. Mostafa and X. Wu, F, "Diagnosis of autism spectrum disorder with convolutional autoencoder and structural mri images," *Neural Engineering Techniques for Autism Spectrum Disorder*, 2021.

[31] A. Aqeel, "What is transposed convolutional layer?" 2020. [Online]. Available: https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11

[32] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018. [Online]. Available: https://papers.nips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html

[33] I. Serengil, S, "Hyperbolic tangent as neural network activation function," 2017. [Online]. Available: https://sefiks.com/2017/01/29/hyperbolic-tangent-as-neural-network-activation-function

[34] A. Matthew, "The hyperbolic tangent function - tanh - where is tanh used as an activation function for neural networks to determine the output in the interval of each neuron its based on."

[35] A. Sankar, "Demystified: Wasserstein gan with gradient penalty(wgan-gp)," 2021. [Online]. Available: https://towardsdatascience.com/demystified-wasserstein-gan-with-gradient-penaltyba5e9b905ead

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Computer Vision - European Conference on Computer Vision*, vol. 8691, pp. 346–361, 2014.

[37] W. Wong, "What is gradient clipping?" 2020. [Online]. Available: https://towardsdatascience.com/what-isgradient-clipping-b8e815cdfb48

[38] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," 2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning

[39] ——, "Gradient descent with rmsprop from scratch," 2021. [Online]. Available: https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch

[40] PyTorch-Organisation(a), "Bce-loss." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

[41] PyTorch-Organisation(b), "Mse-loss." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html

[42] A. Shekhar, "What are l1 and l2 loss functions?" 2019. [Online]. Available: https://afteracademy.com/blog/what-are-l1-and-l2-loss-functions

[43] J. Brownlee, "Difference between a batch and an epoch in a neural network," 2019. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch

[44] C. Yang and Z. Wang, "An ensemble wasserstein generative adversarial network method for road extraction from high resolution remote sensing images in rural areas," *IEEE Access*, vol. 8, 2020.

[45] B. Kim, M. Han, H. Shim, and J. Baek, "A performance comparison of convolutional neural network-based image denoising methods: The effect of loss functions on low-dose ct images," *Medical Physics, American Association of Physicists in Medicine*, 2019.

[46] A. Srhan, M. Abushariah, and O. Kadi, "The effect of loss function on conditional generative adversarial networks," *Journal of King Saud University - Computer and Information Sciences*, 2022.

[47] H. Dwivedi, "Understanding gan loss functions," 2022. [Online]. Available: https://neptune.ai/blog/gan-lossfunctions

[48] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," *IEEE International Conference on Computer Vision Workshops*, 2013. [Online]. Available: http://ai.stanford.edu/~jkrause/cars/car_dataset.html

[49] O. Harrison, "Machine learning basics with the k-nearest neighbors algorithm," 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighborsalgorithm-6a6e71d01761

[50] L. Li, "Introduction to linear regression in python," 2018. [Online]. Available: https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0

[51] A. Navlani, "Understanding random forests classifiers in python tutorial," 2018. [Online]. Available: https://www.datacamp.com/community/tutorials/random-forests-classifier-python