CARTOONIFYING IMAGES USING GENERATIVE ADVERSARIAL NETWORKS

Project Report Submitted in partial fulfillment of

the requirement for the degree of

Special Project

By

U. Naveen Subhash - 19STUCHH010273

A. Vamshidhar Reddy – 19STUCHH010205

G. Manideepak – 19STUCHH010231

Under the Guidance of

Dr. P. Pavan Kumar

Faculty Associate

ICFAI University, Hyderabad

Faculty of Science and Technology

ICFAI University, Hyderabad-501203, Telangana, India.

Declaration

We U. NAVEEN SUBASH, A. VAMSHIDHAR REDDY, G.MANIDEEPAK the team of 3 herebycertify that the work presented in this thesis entitled **"CARTOONIFYING IMAGES USING GENERATIVE ADVERSARIAL NETWORKS"** in fulfillment of the requirements for the special project under the guidance of Dr P. Pavan Kumar. This work has not been submitted to any other university/institution for the award of anydegree/diploma.

Acknowledgement

It gives us great pleasure to express our gratitude for the assistance and direction provided to the team for a number of people to whom I owe agreat deal for the successful completion of this project.

We would like to express our gratitude to **Dr P. Pavan Kumar** sir forall the timely support and valuable suggestions and constant support throughout the project work.

Abstract

In the present generation the non-verbal cues such as Emojis, Toons, Stickers andAvatars have been widely used in the internet conversations, product reviews and other activities. With this, the amount of research in Data Science domain has been increased into emoji-driven storytelling. So, we came up with an idea of creating our own Emojis, Stickers etc. With the help of advancements in Computer Vision and Deep Learning, It is now possible to implement the above idea to detect human emotions and create required emojis and toons. Here in this project, we are using Generative Adversarial Networks to create the model we wanted. In this Special Project we will work with Python, TensorFlow, GAN's, Computer Vision, Deep Learning and Data Science as well.

So basically, we wanted to create a model which gives us the animated output of the respective image we have given as input. Here we used the data set of 7000+ images consisting of selfies of the humans and the testing dataset part consists of the animated version of the given selfies. Itis simply known as selfie2anim

TABLE OF CONTENTS

1. INTRODUCTION

Generative adversarial networks (GANs) are an exciting recent innovation in machine learning. GANs are generative models: they createnew data instances that resemble your training data. For example, GANs can create images that look like photographs of human faces, even thoughthe faces don't belong to any real person. GANs achieve this level of realism by pairing a generator, which learns to produce the target output, with a discriminator, which learns to distinguish true data from the output of the generator. The generator tries to fool the discriminator, and the discriminator tries to keep from being fooled.

Generative – To learn a generative model, which describes how data is generated in terms of a probabilistic model. In simple words, it explains how data is generated visually.

Adversarial – The training of the model is done in an adversarial setting.

Networks – use deep neural networks for training purposes.

Cartoons are a popular style of art that we see every day. Their applications span from publication in printed media to narrative for children's education, in addition to aesthetic interests. Many iconic cartoon images, like other kinds of art, were based on real-life scenes.

However, carefully replicating real-world scenes in cartoon styles is time-consuming and requires significant artistic talent. Artists must draw every line and shade every colour region of target scenes in order to produce high-quality cartoons. Meanwhile, ordinary image editing software/algorithms are incapable of producing decent cartoonization outcomes.

As a result, specially created systems that can automatically transform real-world photos into high-quality cartoon style images are quite useful, and artists can save a significant amount of time so they can focus on more creative work. Photo editing applications like Instagram and Photoshop benefit from the addition of such tools.

Fig 1: An image generated by a StyleGAN that looks depectively like a photograph of a real person.

To begin with, cartoon images are very simplified and abstracted from real-world photography, rather than adding textures such as brush strokes as in many other genres.Second, despite the diversity of techniques among artists, cartoon images have a distinct aesthetic — crisp edges, smooth colour shading, and basic textures — that sets them apart from other types of art.Despite the fact that learning-based stylization has had great success, current methods fail to produce cartoonized images of acceptable quality. There are two factors at play.

To make images with flat shading that resemble cartoon styles, a variety of methods have been devised. Image filtering or formulations in optimization problems are used in these strategies. However, capturing complex aesthetic styles with simple mathematical methods is difficult.Applying uniform filtering or optimization to the entire image, for example, does not achieve the high-level abstraction that an artist would achieve, such as making object borders evident.

Alternative methods rely on image/video segmentation to increase outcomes, however this comes at the cost of some user interaction. For portraits, dedicated methods have been developed, in which semantic segmentation can be produced automatically by detecting facial components.

A GAN framework consists of two CNNs. One is the generator G which is trained to produce output that fools the discriminator. The other is the discriminator D which classifies whether the image is from the real target manifold or synthetic. We design the generator and discriminator networks to suit the particularity of cartoon images

1.1 HISTORY

The most direct inspiration for GANs was noise-contrastive estimation,which uses the same loss function as GANs. Others had comparable ideas but did not develop them in the same way. Olli Niemitalo released a blog post in 2010 about a hypothesis utilising adversarial networks. This concept was never realised, and because the generator lacked stochasticity, it was not a generative model. It's currently known as a cGAN (conditional GAN). Li, Gauci, and Gross employed a concept similar to GANs to model animal behaviour in 2013.

Additional than generative modelling, adversarial machine learning has other applications and can be applied to models other than neural networks. In control theory, adversarial learning with neural networks was used to train resilient controllers in a game theoretic sense in 2006, by alternating iterations between a minimizer policy, the controller, and a maximizer policy, the disturbance.

In 2017, a GAN was used to improve image quality at high magnification by focusing on realistic textures rather than pixel-accuracy. The first faces were created in 2017. These were on display at the Grand Palais in February 2018. StyleGAN's 2019 faces have drawn similarities to deepfakes.

Beginning in 2017, GAN technology made its presence felt in the fine arts arena with the introduction of a newly developed implementation that was said to have crossed the threshold of being able to generate unique and appealing abstract paintings, and was thus dubbed a "CAN," or "creative adversarial network." The picture Edmond de Belamy, which sold for US$432,500 in 2018, was created using a GAN algorithm. Members of the original CAN team highlighted future work with that system, as well as the overall prospects for AI-enabled art, in an early 2019 piece.

In May 2019, Samsung researchers revealed a GAN-based system that can make movies of a person speaking using just a single shot.

A huge dataset of 12,197 MIDI songs with linked lyrics and melody alignment was developed in August 2019 for neural melody creation from lyrics using conditional GAN-LSTM (refer to sources at GitHub AI Melody Generation from Lyrics).

Nvidia researchers taught an AI system (dubbed "GameGAN") to reproduce Pac-Man by simply watching it being played in May 2020.

1.2 APPLICATIONS

Fashion, art and advertising

GANs can be used to make art; in March 2019, The Verge noted, "The visuals made by GANs have become the defining look of current AI art." GANs can also be used to inpaint images or generate shots of fictitious fashion models, eliminating the need to employ a model, photographer, or makeup artist, as well as the costs of a studio and transportation. GANs have also been used to create virtual shadows.

Science

For dark matter research, GANs can improve astronomical images and replicate gravitational lensing. They were successfully utilised in 2019 to estimate the distribution of dark matter in a certain direction in space and predict gravitational lensing.

Through calorimeters of high-energy physics experiments, GANs have been presented as a rapid and accurate technique of modelling high-energy jet production and simulating showers. In computationally intensive simulations of particle physics investigations, GANs have also been trained to accurately approximate bottlenecks. The potential of these technologies for accelerating simulation and/or improving simulation fidelity has been established in the context of current and proposed CERN investigations.

Video games

In 2018, GANs made their way into the video game modding community as a way of reproducing low-resolution 2D textures at classic video games in 4k or higher resolutions using picture training, and then down-sampling them to fit the game's native resolution (with results resembling the super sampling method of anti-aliasing).

GANs create a cleaner and sharper 2D texture image magnitudes greater in quality than the original with full retention of the original's degree of details, colours, and so on with adequate training. Final Fantasy VIII, Final Fantasy IX, Resident Evil REmake HD Remaster, and Max Payne are all known to use GAN extensively.

Audio Synthesis

Generative audio is the process of synthesising audio files from large databases of audio recordings, such as words and sentences that were never spoken. The system is different from AI voices like Apple's Siri or Amazon's Alexa, which use a collection of pieces stitched together on demand.

"Generative audio works differently, using neural networks to learn the statistical properties of the audio source in question, then reproducing those properties directly in any context, modelling how speech changes not just second-by-second, but millisecond-by-millisecond."

Transfer Learning

GANs are used in state-of-the-art transfer learning research, such as deep reinforcement learning, to enforce the alignment of the latent feature space.This works by giving the source and target task embeddings to a discriminator that attempts to guess the context. The encoder then backpropagates the resulting loss (inversely).

Miscellaneous applications

- GAN can be used to detect glaucomatous pictures, assisting in early diagnosis, which is critical to avert visual loss.

- Interior design, industrial design, shoes, bags, and clothing items, as well as elements for computer game scenarios, can all benefit from GANs that produce photorealistic images. Facebook is said to use these types of networks.
- GANs can rebuild 3D models of objects from photographs, produce unique objects in the form of 3D point clouds, and model motion patterns in video.
- GANs can be used to age images of people's faces to highlight how their appearance changes as they get older.
- By reducing the random light reflection on the dynamic weld pool surface, GANs can denoise welding images.
- GANs can be used to enhance data, such as to improve the DNN classifier.
- GANs can also be used to fill in gaps in maps,transfer map styles in cartography], and supplement street view data.
- A GAN model called Speech2Face can reconstruct an image of a person's face after listening to their voice.
- Similarly, A GAN model we used is selfie2anime which can produce an animated image of a person's selfie.
- Generate realistic pictures of people that have never existed.
- Gans is not limited to Images, It can generate text, articles, songs, poems, etc.
- Text to Image Generation (Object GAN and Object Driven GAN)
- Creation of anime characters in Game Development and animation production.
- Image to Image Translation – We can translate one Image to another without changing the background of the source image. For example, Gans can replace a dog with a cat.
- Low resolution to High resolution – If you pass a low-resolution Image or video, GAN can produce a high-resolution Image version of the same.
- Prediction of Next Frame in the video – By training a neural network on small frames of video, GANs are capable to generate or predict a small next frame of video. For example, you can have a look at below GIF.

1.3 TOOLS AND DATASET USED

Fig 2: Tools used

Dataset used was imported from Kaggle website which contains

3400 women faces in Train A

3400 women faces in Train B

100 women faces in Test A

100 women faces in Test B

Dataset link:

(https://www.kaggle.com/datasets/arnaud58/selfie2anime)

2. LITERATURE REVIEW

The project's purpose is to create a model that can detect the emotion of the person on the basis of the text that he/she has written. Basically, the project is made using NLP that will detect the type of the sentence, we are going to predict 6 types of emotions

2.1 Why GANs?

By adding some noise to data, machine learning algorithms and neural networks can be readily misled into miss classifying things. The possibilities of miss classifying the photos grow after adding some noise. As a result of the slight increase, is it possible to create something that allows neural networks to begin visualizing new patterns, such as sample train data? As a result, GANs were created to produce fresh fake results that are identical to the original.

2.2 Different Types of GANs

STYLE GAN: The Style Generative Adversarial Network, or StyleGAN for short, is a GAN extension that proposes major changes to the generator model, such as the use of a mapping network to map points in latent space to an intermediate latent space, the use of the intermediate latent space to control style at each point in the generator model, and the introduction of noise as a source of variation at each point in the generator model.

STAR GAN: StarGAN is an unique and scalable method for doing image-to-image translations across various domains with using one model. StarGAN's unified model architecture allows it to train many datasets with various domains in the same network at the same time.

DC GAN: It's a GAN with deep convolutions. It is one of the most widely used, powerful, and successful GAN architecture kinds. ConvNets are used instead of a Multi-layered perceptron to implement it. ConvNets are formed with a convolutional stride and no max pooling, and the layers in this network are not entirely connected.

Conditional GAN and Unconditional GAN (CGAN): A conditional GAN is a deep learning neural network with a few more parameters. Labels are also added to Discriminator inputs to assist the discriminator in appropriately classifying the input and preventing the generator from filling it.

Least Square GAN(LSGAN): It's a GAN with a discriminator that uses the least-square loss function. The Pearson divergence is minimised when the LSGAN objective function is minimised.

Auxilary Classifier GAN(ACGAN): It's the same as CGAN, but with more features. It specifies that the Discriminator should not only categorize the image as real or fake, but also offer the input image's source or class label.

Dual Video Discriminator GAN: DVD-GAN is a video generating generative adversarial network based on the BigGAN architecture. A Spatial Discriminator and a Temporal Discriminator are employed by DVD-GAN.

SRGAN: Its primary purpose is Domain Transformation, which converts low resolution to high resolution.

Cycle GAN: It was released in 2017 and is used for Image Translation. Assume we've trained it on a dataset of horse photographs and can now translate it to zebra images.

2.3 How do GANs work?

A generative adversarial network (GAN) has two parts:

- The **generator** learns to generate plausible data.The generatedinstances become negative training examples for the discriminator.
- he **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake.As training progresses, the generator gets closer to producing output that canfool the discriminator. Finally, if generator training goes well, the discriminator gets worse at tellingthe difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

2.4 Steps to implement a basic GAN

1. Importing all libraries
2. Getting the Dataset
3. Data Preparation – It includes various steps to accomplish like preprocessing data, scaling, flattening, and reshaping the data.
4. Define the function Generator and Discriminator.
5. Create a Random Noise and then create an Image with Random Noise.
6. Setting Parameters like defining epoch, batch size, and Sample size.
7. Define the function of generating Sample Images.
8. Train Discriminator then trains Generator and it will create Images.
9. Will see what clarity of Images is created by Generator.

2.5 GAN Architecture

GANs (generative adversarial networks) are computational structures that pit two neural networks against each other (thus the name "adversarial") to generate fresh, synthetic examples of data that can pass for real data. They're commonly employed in picture, video, and voice generation..

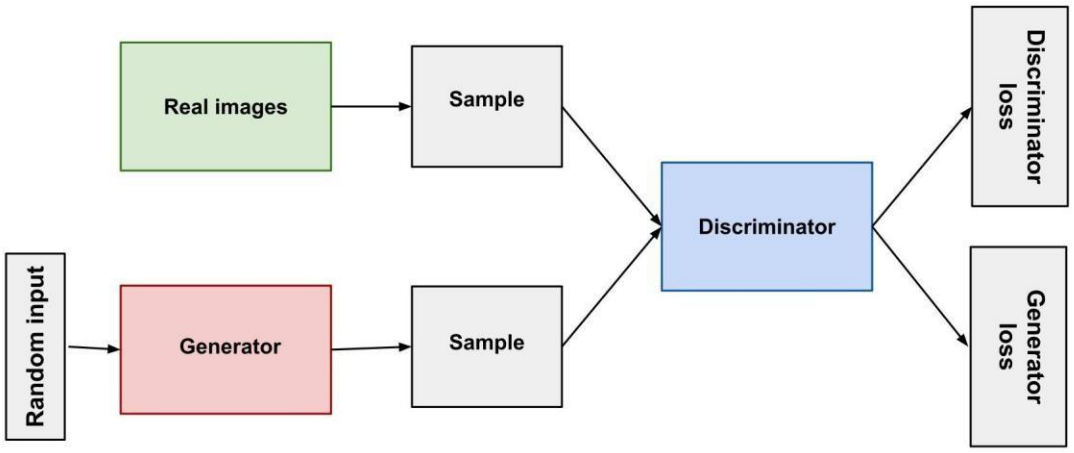

Fig 3: GAN Architecture

Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides asignal that the generator uses to update its weights.

2.5.1 Network Architectures

Although deep neural networks have the potential to increase the capacity to represent complex functions, many studies reveal that they are difficult to train due to the infamous vanishing gradient problem. The concept of residual blocks, which was recently developed, is a significant tool for simplifying the training process. It creates a "identity shortcut link" that solves the problem of vanishing gradients during training. In generative networks, models based on residual blocks have performed admirably.

Batch normalization, which is aimed to counteract the internal covariate shift and lessen oscillations when reaching the minimum point, is another typical strategy to make deep CNN training easier. LReLU (Leaky ReLu) is a widely used activation function in deep CNNs for efficient gradient propagation, which improves network performance by permitting a tiny, non-zero gradient when the unit is not active. These techniques are incorporated into our deep cartoonization architecture.

2.5.2 Discriminator

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying. A Discriminator is a police officer whose job it is to spot anomalies in Generator's samples and label them as Fake or Real.

Because it is a supervised approach, It's a straightforward classifier that determines if data is bogus or authentic. It learns from real-world data and gives feedback to a generator.

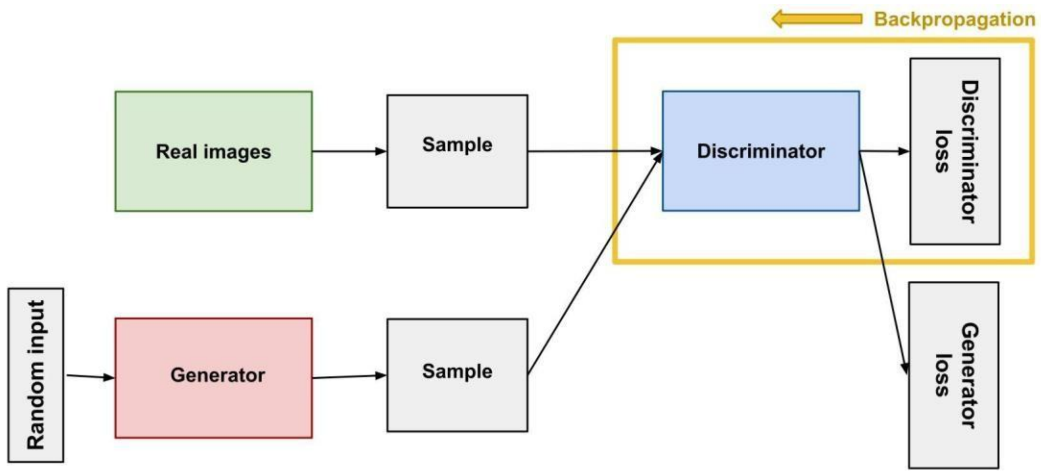

Fig 4: Discriminator Training

Discriminator Training Data

The discriminator's training data comes from two sources:

Real data instances, such as real pictures of people. The discriminatoruses these instances as positive examples during training.

Fake data instances created by the generator. The discriminator uses these instances as negative examples during training.

During discriminator training:

1. The discriminator classifies both real data and fake data from thegenerator.
2. The discriminator loss penalizes the discriminator for miss classifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

2.5.3 Generator

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real.The generator's duty is similar to that of a burglar, creating phoney samples based on the original sample and fooling the discriminator into mistaking Fake for Real.

It is a method of unsupervised learning. It will generate phoney data based on the original(real) data. It's also a neural network with hidden layers, activation, and loss. Its goal is to use feedback to build a fake image and deceive the discriminator into thinking it can't predict a fake image. When the generator fools the discriminator, the training terminates, and we may say that a generalised GAN model is produced.

The portion of the GAN that trains the generator includes:

- random input
- generator network, which transforms the random input into a data instance
- discriminator network, which classifies the generated data
- discriminator output
- generator loss, which penalizes the generator for failing to fool the discriminator.

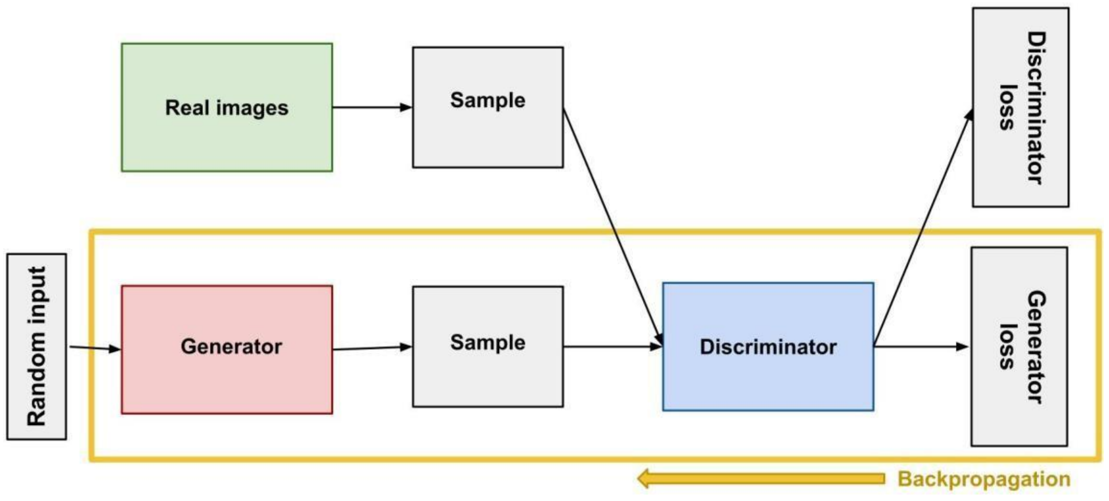

Fig 5: Generator Training

So we train the generator with the following procedure:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification for generator output.
4. Calculate loss from discriminator classification.
5. Backpropagate through both the discriminator and generator to obtaingradients.
6. Use gradients to change only the generator weights.

2.6 GAN Training

Because a GAN contains two separately trained networks, its training algorithm must address two complications:

- GANs must juggle two different kinds of training (generator and discriminator).
- GAN convergence is hard to identify.

2.6.1 Alternating Training

The generator and the discriminator have different training processes. So how do we train the GAN as a whole?

GAN training proceeds in alternating periods:

1. The discriminator trains for one or more epochs.

2.The generator trains for one or more epochs.

3. Repeat the steps 1 and 2 to continue to train.

2.6.2 Convergence

As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction.

This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.

For a GAN, convergence is often a fleeting, rather than stable, state.

2.7 Loss Functions

GANs try to replicate a probability distribution. They should therefore use loss functions that reflect the distance between the distribution of the data generated by the GAN and the distribution of the real data.

We'll address two common GAN loss functions here:

- **minimax loss**: The loss function used in the paper that introduced GANs.
- **Wasserstein loss**: The default loss function for TF-GAN Estimators. First described in a 2017 paper.

TF-GAN implements many other loss functions as well.

2.7.1 Minimax Loss

In the paper that introduced GANs, the generator tries to minimize the following function while the discriminator tries to maximize it:

$$\mathbf{Ex[log(D(x))]+Ez[log(1-D(G(z)))]}$$

In this function:

- D(x) is the discriminator's estimate of the probability that real data instance x is real.
- E_x is the expected value over all real data instances.
- G(z) is the generator's output when given noise z.
- D(G(z)) is the discriminator's estimate of the probability that a fake instance is real.
- E_z is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z)).
- The formula derives from the cross-entropy between the real and generated distributions.

2.7.2 Wasserstein Loss

This loss function depends on a modification of the GAN scheme (called "Wasserstein GAN" or "WGAN") in which the discriminator does not actually classify instances. For each instance it outputs a number. This number does not have to be less than one or greater than 0, so we can't use 0.5 as a threshold to decide whether an instance is real or fake. Discriminator training just tries to make the output bigger for real instances than for fake instances.

Because it can't really discriminate between real and fake, the WGAN discriminator is actually called a "critic" instead of a "discriminator". This distinction has theoretical importance, but for practical purposes we can treat it as an acknowledgement that the inputs to the loss functions don't have to be probabilities.

The loss functions themselves are deceptively simple:

Critic Loss: D(x) - D(G(z))

The discriminator tries to maximize this function. In other words, it tries to maximize the difference between its output on real instances and its output on fake instances.

Generator Loss: D(G(z))

The generator tries to maximize this function. In other words, It tries to maximize the discriminator's output for its fake instances.

In these functions:

- D(x) is the critic's output for a real instance.
- G(z) is the generator's output when given noise z.
- D(G(z)) is the critic's output for a fake instance.
- The output of critic D does *not* have to be between 1 and 0.
- The formulas derive from the earth mover distance between the real and generated distributions.

2.7.3 L1 Loss Function

L1 Loss Function is used to minimize the error which is the sum of the all the **absolute** differences between the true value and the predicted value.

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

2.7.4 Mean Squared Error (MSE)

The Mean Squared Error (MSE) is perhaps the simplest and most common loss function, often taught in introductory Machine Learning courses. To calculate the MSE, you take the difference between your model's predictions and the ground truth, square it, and average it out across the whole dataset.

The MSE will never be negative, since we are always squaring the errors. The MSE is formally defined by the following equation:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

2.7.5 Binary Cross Entropy Loss (BCE)

The binary cross entropy loss function calculates the loss of an example by computing the following average:

$$\text{Loss} = -\frac{1}{\substack{\text{output}\\\text{size}}}\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

where hat{y} is the i*i*-th scalar value in the model output, *yi* is the corresponding target value, and output size is the number of scalar values in the model output. And the optimizer used for both generator and discriminator is ADAM.

3. METHODOLOGY

3.1 Importing Files

We are importing the selfie2anime dataset containing data for training, testing. We have acquired data from kaggle and divided the data into 2 parts. The data contains about 7000 images of selfie images and animated images.

Preprocessing functions used in code

3.1.1 Utils.py

load_test_data(image_path, size=256):

A function used to get all the test data to perform activities for modelling. It reads the image and resizes it into 256x256 image size and thenpreprocess it using preprocessing function. It uses 2 arguments image path and image size.

preprocessing(x):

Here it changes the values of x's into the range of -1 to 1. Takes only 1 argument x.

save_images(images, size, image_path):

Save multiple images into one single image. Number of images should beequal or less than size[0] * size[1]. Arguments follows are images, imagesize and path.

inverse_transform(images):

This converts the images into high dimensional representation and changes the values into (images+1.)/2

imsave(images, size, path):

It saves the images after inverse transforming the images using inversetransform function. Takes 3 arguments similar to save_image function.

merge(images, size):

Merges the images one by one based on the direction given, like up and down or side by side. Takes 2 arguments images and image size.

check_folder(log_dir):

checks if log_dir is in the path and if no create a new directory namedlog_dir. Returns a boolean status.

str2bool(x):

String to Boolean conversion. A simple function to convert a string toBoolean i.e. True,false, by guessing the meaning of input string.

cam(x, size = 256):

Used to change the selected size and apply color mapping to the images,changes the inputs into (x-min)/max(x).

imagenet_norm(x): and denorm(x):

These norm functions are used for normalization which used to train themodel. Stored as constants inside the library named imagenet. denorm changes the mean to 0.5 and std to 0.5 to the normalized distribution.

tensor2numpy(x):

Allows to convert a 2-dimensional tensor to a NumPy array and then backto tensor.

RGB2BGR(x):

Changes the format of coloring from RGB to BGR using open cv.

3.1.2 dataset.py

has_file_allowed_extension(filename, extensions):

Checks if filename file belongs to the permitted extensions or not. Takes 2 arguments filename and extensions.

find_classes(dir):

Finds the class folder in a dataset. This method can be overridden to only consider a subset of classes.

make_dataset(dir, extensions):

If the file in the directory is allowed (i.e., belongs to the image formatextensions then add these files into the dataset. Generates a list of samples. Takes 2 arguments directory and extensions.

class DatasetFolder(data.Dataset):

(child class of torch.utils.data class) makes the dataset using make_dataset function by passing the current folder path

__get_item_(self,index):

By passing the index it gets the desired image from the dataset.

__len_(self):

Gives the length of the samples in the dataset.

__repr__(self):

Used to compute the official string representation of an object.

pil_loader(path):

It is a helper function used to load an image using PIL library, and returnit as a RGB image.

default_loader(path):

calls the pil loader.

class ImageFolder(DatasetFolder):

A generic data loader where the images are arranged in default way.This class inherits from class:`~torchvision.datasets.DatasetFolder` so the same methods can be overridden to customize the dataset.

3.1.3 Networks.py

ResnetGenerator:

It is a class used to create a generator which was proposed in a paper called Generative Adversarial Network based on Resnet for Conditional Image Restoration. It contains some down sampling, class activation layers and some linear layers. We then restore the dimensions using up-sampling layers. Forward block, resnet block, resnetadaILN block, adaILN block and ILN block are used in creating generator for this GAN. Each of these blocks contains sub networks for the main generator block containing series of padding, conv and reflection padding layers.

Discriminator:

It is a class for creating discriminator of the GAN. It is a classifier that classifies the real image(real selfie image in this case) and fake images those generated by resnet generator and gives a loss to improve the performance of both generator and discriminator.

It contains series of reflection ads, conv2 layers. And generates class activation maps(CAMs), lets us see which regions in the image were relevant to this class for example: eyes, nose, lips, etc.

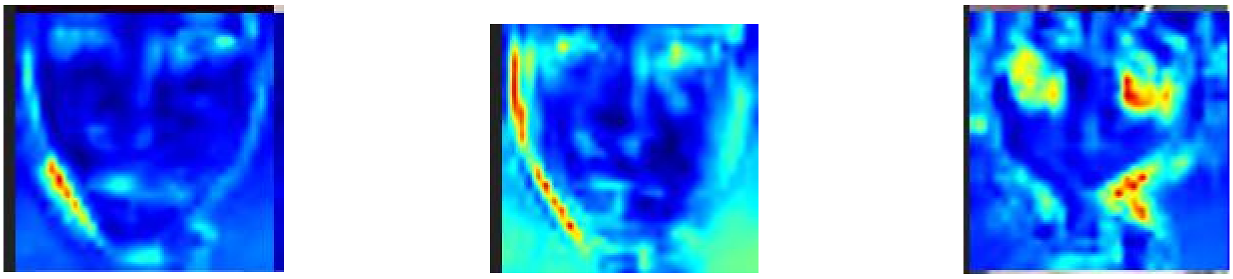

Fig 6: Different generated layers of images

3.1.4 UGATIT.py

It is used to integrate generator and discriminator into one single model and to create some functions to train, test, save the model, load the model.

Build_model function:

Integrates the generator and the discriminator which were already defined in networks.py file. It first loads the train set and test set which we have from kaggle. Train set contains around 7000 images and test set contains around 200 images. In this model, we use L1 loss and Mean Square Error(MSE) loss and Binary Cross Entropy loss(BCE).

Train function:

It first loads the already saved model and then continue the iterations using the losses defined earlier and then updating the weights of generator and discriminator. Then saves the model for the specific iterations and saves the sample images and their animated images to a folder named “results”.

The images saved contains the series of images with their heat maps (class activation map) attached at the bottom of this image. And finally the generated animated image of the given selfie image at the bottom.

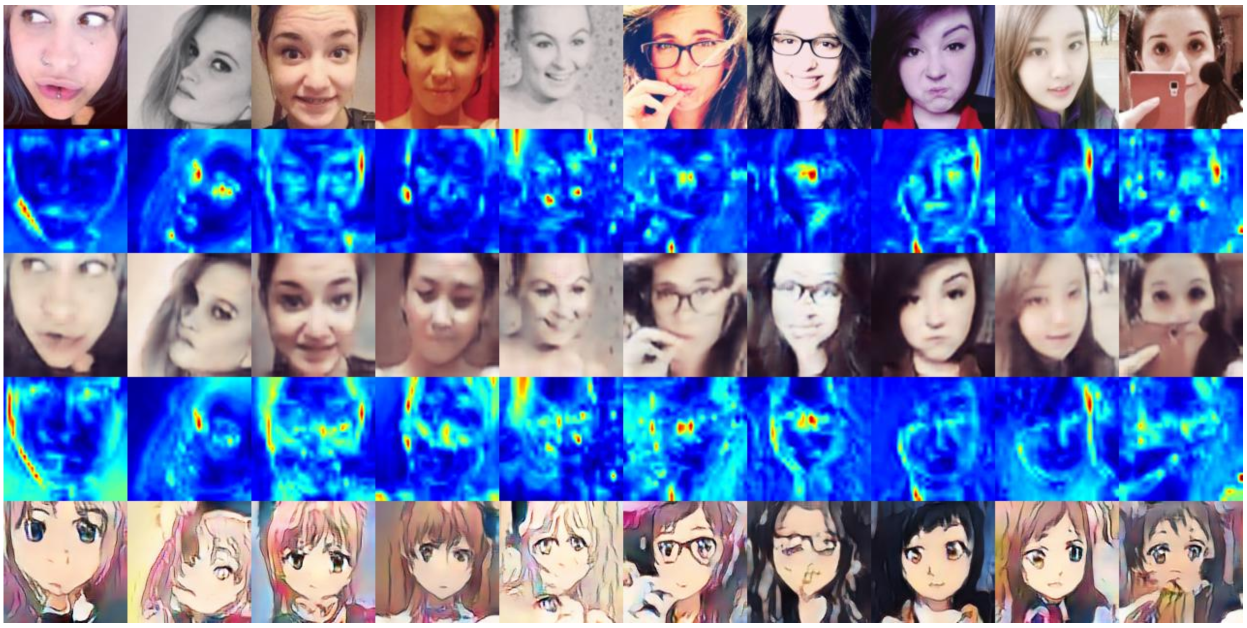

Fig 7: Sample output of Train function

Save function:

It saves the parameters(weights) of the layers in the generator and discriminator to the specified location.

Load function:

Loads the parameters(weights) of the generator and discriminator from the given path of the folder.

Test function:

First, loads the latest model which was saved during training part. Then by taking the images from the test folder. The model will generate the animated images to the images in the test folder. And saves the newly generated image in a sub folder called “test” in the results folder.

3.1.5 Main.py:

It gives the user the option to run in cpu or gpu. And whether to train or test or resume the training. And gives the user to change the batch size, image size, image channel, and many other parameters which makes the user to control the training process. And can run train the model with lesser computation power. But, remember that quality of images generated will depend on these parameters.

4. RESULTS

Results folder will be generated in the process of training the model. What we have observed in the training process is that the similarity between the animated image and real image would decrease as the training iterations increase.

By increasing the number of iterations the quality of the image also get increased, which can be observed in the below image samples. The model randomly pics the images and gives the output of few samples. We expected a good quality image after 100000 iterations and obtained as well.

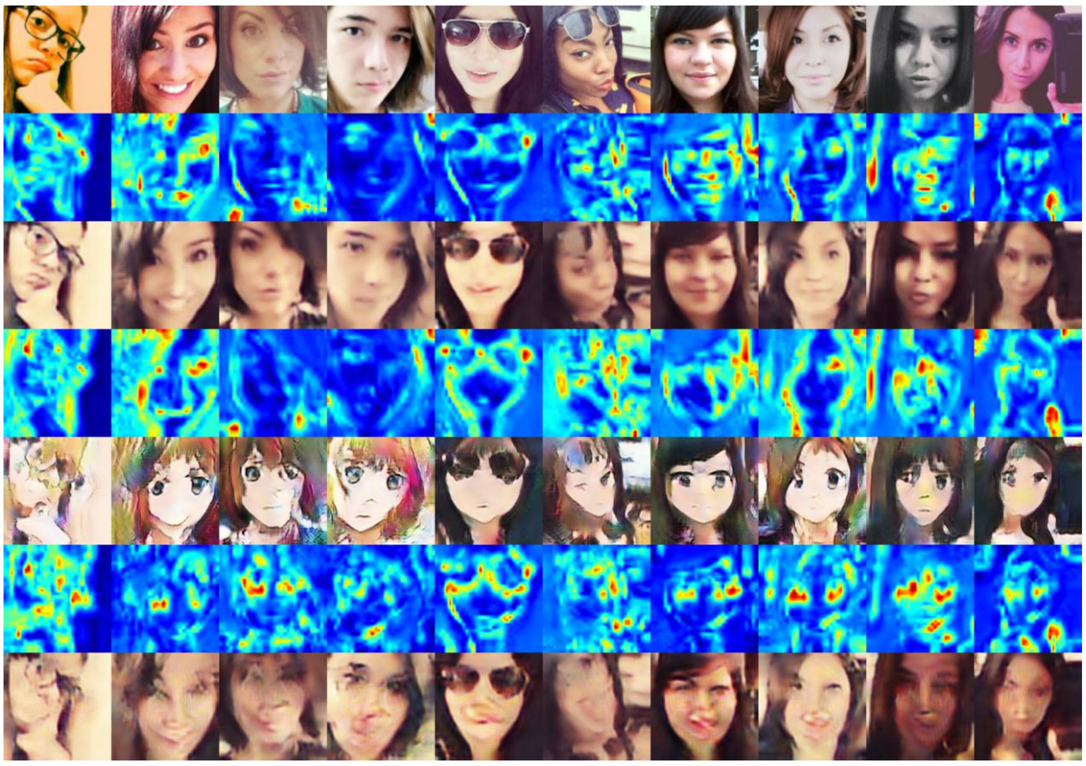

Fig 8: Generated image of some samples after 10000 iterations

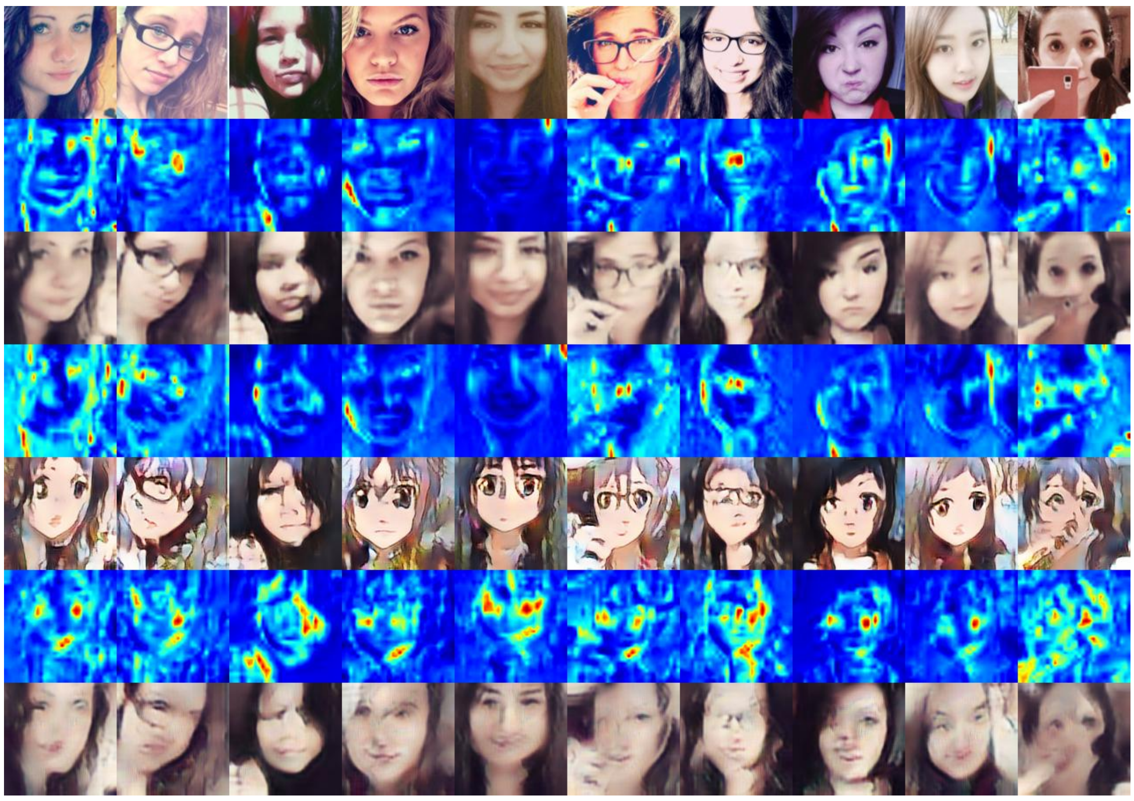

Fig 9: Generated image of some samples after 27000 iterations

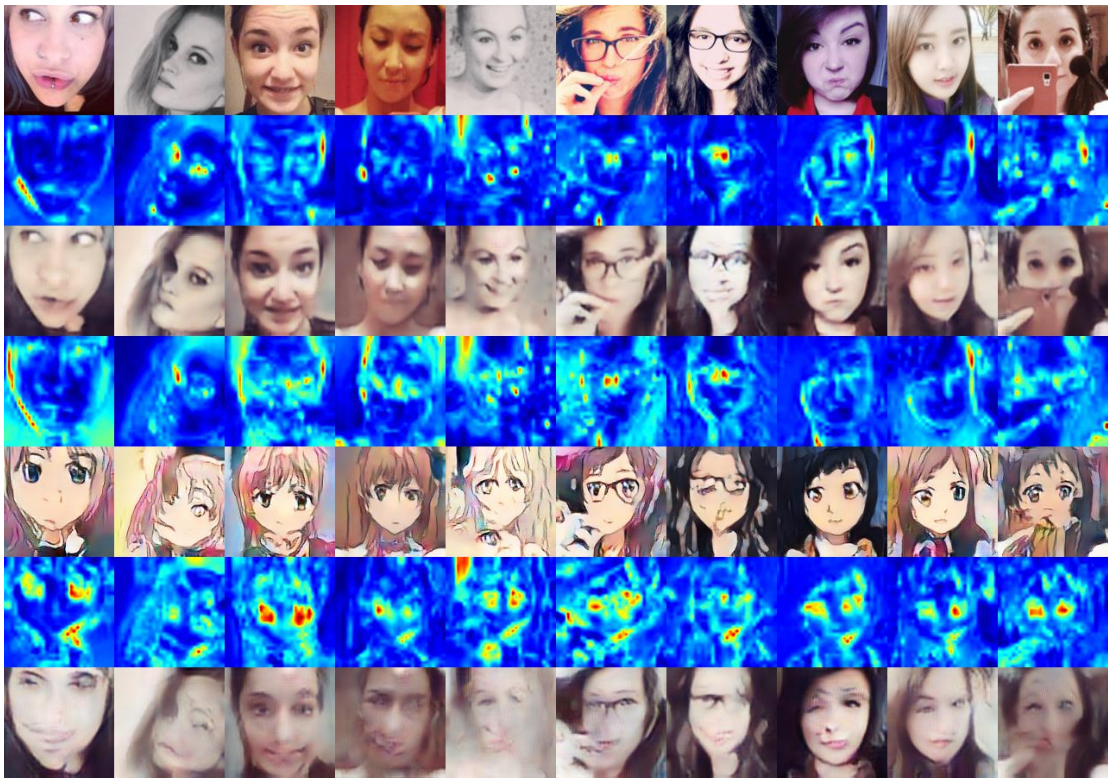

Fig 10: Generated image of some samples after 600000 iterations

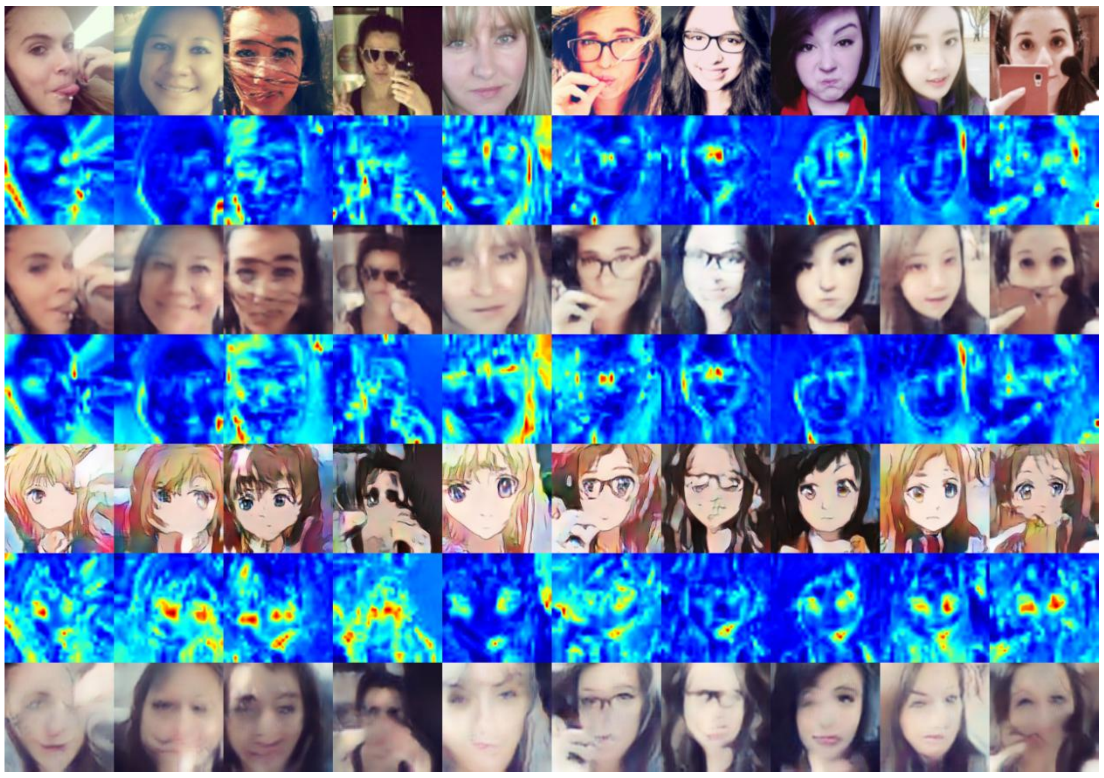

Fig 11: Generated image of some samples after 100000 iterations

As we can observe the animated image generated gets more closer to the real image which was given to the generator. And the training process a lot of time to run through these iterations which makes it a bit harder to train.

5. CODE

5.1 Utils.py

```python
from scipy import misc
import os, cv2, torch
import numpy as np

def load_test_data(image_path, size=256):
    img = misc.imread(image_path, mode='RGB')
    img = misc.imresize(img, [size, size])
    img = np.expand_dims(img, axis=0)
    img = preprocessing(img)

    return img

def preprocessing(x):
    x = x/127.5 - 1 # -1 ~ 1
    return x

def save_images(images, size, image_path):
    return imsave(inverse_transform(images), size, image_path)

def inverse_transform(images):
    return (images+1.) / 2

def imsave(images, size, path):
    return misc.imsave(path, merge(images, size))

def merge(images, size):
    h, w = images.shape[1], images.shape[2]
    img = np.zeros((h * size[0], w * size[1], 3))
    for idx, image in enumerate(images):
        i = idx % size[1]
        j = idx // size[1]
        img[h*j:h*(j+1), w*i:w*(i+1), :] = image

    return img

def check_folder(log_dir):
    if not os.path.exists(log_dir):
        os.makedirs(log_dir)
    return log_dir

def str2bool(x):
    return x.lower() in ('true')

def cam(x, size = 256):
    x = x - np.min(x)
    cam_img = x / np.max(x)
    cam_img = np.uint8(255 * cam_img)
    cam_img = cv2.resize(cam_img, (size, size))
    cam_img = cv2.applyColorMap(cam_img, cv2.COLORMAP_JET)
    return cam_img / 255.0

def imagenet_norm(x):
    mean = [0.485, 0.456, 0.406]
    std = [0.299, 0.224, 0.225]
    mean = torch.FloatTensor(mean).unsqueeze(0).unsqueeze(2).unsqueeze(3).
to(x.device)
    std = torch.FloatTensor(std).unsqueeze(0).unsqueeze(2).unsqueeze(3).t
o(x.device)
    return (x - mean) / std

def denorm(x):
    return x * 0.5 + 0.5

def tensor2numpy(x):
    return x.detach().cpu().numpy().transpose(1,2,0)

def RGB2BGR(x):
    return cv2.cvtColor(x, cv2.COLOR_RGB2BGR)
```

5.2 Dataset.py

```
import torch.utils.data as data

from PIL import Image

import os
import os.path


def has_file_allowed_extension(filename, extensions):
    """Checks if a file is an allowed extension.

    Args:
        filename (string): path to a file

    Returns:
        bool: True if the filename ends with a known image extension
    """
    filename_lower = filename.lower()
    return any(filename_lower.endswith(ext) for ext in extensions)


def find_classes(dir):
    classes = [d for d in os.listdir(dir) if os.path.isdir(os.path.join(d
ir, d))]
    classes.sort()
    class_to_idx = {classes[i]: i for i in range(len(classes))}
    return classes, class_to_idx


def make_dataset(dir, extensions):
    images = []
    for root, _, fnames in sorted(os.walk(dir)):
        for fname in sorted(fnames):
            if has_file_allowed_extension(fname, extensions):
                path = os.path.join(root, fname)
                item = (path, 0)
                images.append(item)

    return images


class DatasetFolder(data.Dataset):
    def __init__(self, root, loader, extensions, transform=None, target_t
ransform=None):
        # classes, class_to_idx = find_classes(root)
        samples = make_dataset(root, extensions)
        if len(samples) == 0:
            raise(RuntimeError("Found 0 files in subfolders of: " + root
+ "\n"
                               "Supported extensions are: " + ",".join(ex
tensions)))

        self.root = root
        self.loader = loader
        self.extensions = extensions
        self.samples = samples

        self.transform = transform
        self.target_transform = target_transform

    def __getitem__(self, index):
        """
        Args:
            index (int): Index

        Returns:
            tuple: (sample, target) where target is class_index of the ta
rget class.
        """
        path, target = self.samples[index]
        sample = self.loader(path)
        if self.transform is not None:
            sample = self.transform(sample)
        if self.target_transform is not None:
            target = self.target_transform(target)

        return sample, target

    def __len__(self):
        return len(self.samples)

    def __repr__(self):
        fmt_str = 'Dataset ' + self.__class__.__name__ + '\n'
        fmt_str += '    Number of datapoints: {}\n'.format(self.__len__())
        fmt_str += '    Root Location: {}\n'.format(self.root)
        tmp = '    Transforms (if any): '
        fmt_str += '{0}{1}\n'.format(tmp, self.transform.__repr__().repla
ce('\n', '\n' + ' ' * len(tmp)))
        tmp = '    Target Transforms (if any): '
        fmt_str += '{0}{1}'.format(tmp, self.target_transform.__repr__().
replace('\n', '\n' + ' ' * len(tmp)))
        return fmt_str


IMG_EXTENSIONS = ['.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif']


def pil_loader(path):
    # open path as file to avoid ResourceWarning (https://github.com/pyth
on-pillow/Pillow/issues/835)
    with open(path, 'rb') as f:
        img = Image.open(f)
        return img.convert('RGB')


def default_loader(path):
    return pil_loader(path)


class ImageFolder(DatasetFolder):
    def __init__(self, root, transform=None, target_transform=None,
                 loader=default_loader):
        super(ImageFolder, self).__init__(root, loader, IMG_EXTENSIONS,
                                          transform=transform,
                                          target_transform=target_transfo
rm)
        self.imgs = self.samples
```

5.3 Main.py

```
from UGATIT import UGATIT
import argparse
from utils import *

"""parsing and configuration"""
torch.cuda.empty_cache()
def parse_args():
    desc = "Pytorch implementation of U-GAT-IT"
    parser = argparse.ArgumentParser(description=desc)
    parser.add_argument('--
phase', type=str, default='train', help='[train / test]') # have changed it from train to load
    parser.add_argument('--light', type=str2bool, default=True, help='[U-GAT-
IT full version / U-GAT-IT light version]')
    parser.add_argument('--dataset', type=str, default='YOUR_DATASET_NAME', help='dataset_name')

    parser.add_argument('--
iteration', type=int, default=85000, help='The number of training iterations') #1000000
    parser.add_argument('--
batch_size', type=int, default=1, help='The size of batch size') # changed it from 32
    parser.add_argument('--
print_freq', type=int, default=100, help='The number of image print freq') #1000
    parser.add_argument('--
save_freq', type=int, default=100, help='The number of model save freq')  #1000
    parser.add_argument('--decay_flag', type=str2bool, default=True, help='The decay_flag')

    parser.add_argument('--lr', type=float, default=0.0001, help='The learning rate')
    parser.add_argument('--weight_decay', type=float, default=0.0001, help='The weight decay')
    parser.add_argument('--adv_weight', type=int, default=1, help='Weight for GAN')
    parser.add_argument('--cycle_weight', type=int, default=10, help='Weight for Cycle')
    parser.add_argument('--identity_weight', type=int, default=10, help='Weight for Identity')
    parser.add_argument('--cam_weight', type=int, default=1000, help='Weight for CAM')

    parser.add_argument('--ch', type=int, default=32, help='base channel number per layer')
    parser.add_argument('--n_res', type=int, default=4, help='The number of resblock')
    parser.add_argument('--n_dis', type=int, default=6, help='The number of discriminator layer')

    parser.add_argument('--
img_size', type=int, default=128, help='The size of image') # have changed it from 256 to 128
    parser.add_argument('--img_ch', type=int, default=3, help='The size of image channel')

    parser.add_argument('--
result_dir', type=str, default='results', help='Directory name to save the results')
    parser.add_argument('--
device', type=str, default='cuda', choices=['cpu', 'cuda'], help='Set gpu mode; [cpu, cuda]')
    parser.add_argument('--benchmark_flag', type=str2bool, default=False)
    parser.add_argument('--resume', type=str2bool, default=False)
    return check_args(parser.parse_args())
```

```
"""checking arguments"""
def check_args(args):
    # --result_dir
    check_folder(os.path.join(args.result_dir, args.dataset, 'model'))
    check_folder(os.path.join(args.result_dir, args.dataset, 'img'))
    check_folder(os.path.join(args.result_dir, args.dataset, 'test'))

    # --epoch
    try:
        assert args.epoch >= 1
    except:
        print('number of epochs must be larger than or equal to one')

    # --batch_size
    try:
        assert args.batch_size >= 1
    except:
        print('batch size must be larger than or equal to one')
    return args

"""main"""
def main():
    # parse arguments
    args = parse_args()
    if args is None:
      exit()

    # open session
    gan = UGATIT(args)

    # build graph
    gan.build_model()

    if args.phase == 'train' :
        gan.train()
        print(" [*] Training finished!")

    if args.phase == 'load' :
        gan.load("/content/drive/MyDrive/UGATIT-pytorch-master/Results", step = 200)
        print(" [*] Loaded the saved model from step = 2000")

    if args.phase == 'test' :
        gan.test()
        print(" [*] Test finished!")

if __name__ == '__main__':
    mai
```

6. CONCLUSION

We suggested CartoonGAN, a Generative Adversarial Network, in this study to convert real-world pictures into high-quality cartoon style graphics. Attempting to recreate We present (1) a method for capturing the faithful properties of cartoon images.for clear edges, a unique edge-promoting adversarial loss, and
(2) high-level feature maps with a l1 sparse regularisationfor content loss in the VGG network, which provides enough flexibility for replicating smooth shading We also present a straightforward but effective initialization phase to assist enhance convergence.

The results reveal that CartoonGAN can learn a model that converts photos of real-world scenes into cartoon-style images with great quality and efficiency, exceeding state-of-the-art stylization approaches by a wide margin.

7. REFERENCES

1. *Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). Generative Adversarial Nets* (PDF). *Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.*
2. ^ *Salimans, Tim; Goodfellow, Ian; Zaremba, Wojciech; Cheung, Vicki; Radford, Alec; Chen, Xi (2016). "Improved Techniques for Training GANs".* arXiv:1606.03498 *[*cs.LG*].*
3. ^ *Isola, Phillip; Zhu, Jun-Yan; Zhou, Tinghui; Efros, Alexei (2017).* "Image-to-Image Translation with Conditional Adversarial Nets". Computer Vision and Pattern Recognition.
4. ^ *Ho, Jonathon; Ermon, Stefano (2016).* "Generative Adversarial Imitation Learning". Advances in Neural Information Processing Systems. *29: 4565–4573.* arXiv:1606.03476. Bibcode:2016arXiv160603476H.
5. ^ "Vanilla GAN (GANs in computer vision: Introduction to generative learning)". theaisummer.com. *AI Summer. April 10, 2020.* Archived *from the original on June 3, 2020.* Retrieved September 20, 2020.
6. ^ *Luc, Pauline; Couprie, Camille; Chintala, Soumith; Verbeek, Jakob (November 25, 2016). "Semantic Segmentation using Adversarial Networks".* NIPS Workshop on Adversarial Training, Dec, Barcelona, Spain. *2016.* arXiv:1611.08408. Bibcode:2016arXiv161108408L.
7. ^ Andrej Karpathy*;* Pieter Abbeel*; Greg Brockman; Peter Chen; Vicki Cheung; Rocky Duan; Ian Goodfellow; Durk Kingma; Jonathan Ho; Rein Houthooft; Tim Salimans; John Schulman; Ilya Sutskever; Wojciech Zaremba, Generative Models*, OpenAI, retrieved April 7, 2016
8. ^ *Lin, Zinan; et al. (December 2018).* "PacGAN: the power of two samples in generative adversarial networks". *NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems: 1505–1514.* arXiv:1712.04086. (also available arXiv:1712.04086)
9. ^ *Mescheder, Lars; Geiger, Andreas; Nowozin, Sebastian (July 31, 2018). "Which Training Methods for GANs do actually Converge?".* arXiv:1801.04406 *[*cs.LG*].*
10. ^ *Mohamed, Shakir; Lakshminarayanan, Balaji (2016). "Learning in Implicit Generative Models".* arXiv:1610.03483 *[*stat.ML*].*
11. ^ *Caesar, Holger (March 1, 2019), A list of papers on Generative Adversarial (Neural) Networks: nightrome/really-awesome-gan*, retrieved March 2, 2019
12. ^ *Robertson, Adi (February 21, 2022).* "The US Copyright Office says an AI can't copyright its art". The Verge. Retrieved February 24, 2022.
13. ^ *Vincent, James (March 5, 2019).* "A never-ending stream of AI art goes up for auction". The Verge. Retrieved June 13, 2020.

14. ^ Yu, Jiahui, et al. "Generative image inpainting with contextual attention." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
15. ^ *Wong, Ceecee.* "The Rise of AI Supermodels". CDO Trends.
16. ^ *Taif, K.; Ugail, H.; Mehmood, I. (2020).* "Cast Shadow Generation Using Generative Adversarial Networks". Computational Science – Iccs 2020. *Lecture Notes in Computer Science. 12141: 481–495.* doi:10.1007/978-3-030-50426-7_36. ISBN 978-3-030-50425-0. PMC 7302543.
17. ^ *Schawinski, Kevin; Zhang, Ce; Zhang, Hantian; Fowler, Lucas; Santhanam, Gokula Krishnan (February 1, 2017). "Generative Adversarial Networks recover features in astrophysical images of galaxies beyond the deconvolution limit".* Monthly Notices of the Royal Astronomical Society: Letters. *467 (1): L110–*
L114. arXiv:1702.00403. Bibcode:2017MNRAS.467L.110S. doi:10.1093/mnrasl/slx008. S2CID 7213940.
18. ^ *Kincade, Kathy.* "Researchers Train a Neural Network to Study Dark Matter". *R&D Magazine.*
19. ^ *Kincade, Kathy (May 16, 2019).* "CosmoGAN: Training a neural network to study dark matter". Phys.org.
20. ^ "Training a neural network to study dark matter". Science Daily. *May 16, 2019.*
21. ^ *at 06:13, Katyanna Quach 20 May 2019.* "Cosmoboffins use neural networks to build dark matter maps the easy way". www.theregister.co.uk. Retrieved May 20, 2019.
22. ^ *Mustafa, Mustafa; Bard, Deborah; Bhimji, Wahid; Lukić, Zarija; Al-Rfou, Rami; Kratochvil, Jan M. (May 6, 2019). "CosmoGAN: creating high-fidelity weak lensing convergence maps using Generative Adversarial Networks".* Computational Astrophysics and Cosmology. *6 (1): 1.* arXiv:1706.02390. Bibcode:2019ComAC...6....1M. doi:10.1186/s40668-019-0029-9. ISSN 2197-7909. S2CID 126034204.
23. ^ *Paganini, Michela; de Oliveira, Luke; Nachman, Benjamin (2017). "Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis".* Computing and Software for Big Science. *1: 4.* arXiv:1701.05927. Bibcode:2017arXiv170105927D. doi:10.1007/s41781-017-0004-6. S2CID 88514467.
24. ^ *Paganini, Michela; de Oliveira, Luke; Nachman, Benjamin (2018). "Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multi-Layer Calorimeters".* Physical Review Letters. *120 (4): 042003.* arXiv:1705.02355. Bibcode:2018PhRvL.120d2003P. doi:10.1103/PhysRevLett.120.042003. PMID 29437460. S2CID 3330974.
25. ^ *Paganini, Michela; de Oliveira, Luke; Nachman, Benjamin (2018). "CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks".* Phys. Rev. D. *97 (1):*

014021. arXiv:1712.10321. Bibcode:2018PhRvD..97a4021P. doi:10.1103/PhysRevD.97.014021. S2CID 41265836.

26. ^ *Erdmann, Martin; Glombitza, Jonas; Quast, Thorben (2019). "Precise Simulation of Electromagnetic Calorimeter Showers Using a Wasserstein Generative Adversarial Network".* Computing and Software for Big Science. *3: 4*. arXiv:1807.01954. doi:10.1007/s41781-018-0019-7. S2CID 54216502.
27. ^ *Musella, Pasquale; Pandolfi, Francesco (2018). "Fast and Accurate Simulation of Particle Detectors Using Generative Adversarial Networks".* Computing and Software for Big Science. *2: 8*. arXiv:1805.00850. Bibcode:2018arXiv180500850M. doi:10.1007/s41781-018-0015-y. S2CID 119474793.
28. ^ *ATLAS, Collaboration (2018).* "Deep generative models for fast shower simulation in ATLAS".
29. ^ *SHiP, Collaboration (2019). "Fast simulation of muons produced at the SHiP experiment using Generative Adversarial Networks".* Journal of Instrumentation. *14 (11): P11028*. arXiv:1909.04451. Bibcode:2019JInst..14P1028A. doi:10.1088/1748-0221/14/11/P11028. S2CID 202542604.
30. ^ *Tang, Xiaoou; Qiao, Yu; Loy, Chen Change; Dong, Chao; Liu, Yihao; Gu, Jinjin; Wu, Shixiang; Yu, Ke; Wang, Xintao (September 1, 2018). "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks".* arXiv:1809.00219. Bibcode:2018arXiv180900219W.
31. ^ "Fake news: you ain't seen nothing yet". The Economist. *July 2017*. Retrieved July 1, 2017.
32. ^ *msmash (February 14, 2019).* "'This Person Does Not Exist' Website Uses AI To Create Realistic Yet Horrifying Faces". Slashdot. Retrieved February 16, 2019.
33. ^ *Doyle, Michael (May 16, 2019).* "John Beasley lives on Saddlehorse Drive in Evansville. Or does he?". *Courier and Press.*
34. ^ *Targett, Ed (May 16, 2019). "California moves closer to making deepfake pornography illegal". Computer Business Review.*
35. ^ *Mihalcik, Carrie (October 4, 2019).* "California laws seek to crack down on deepfakes in politics and porn". *cnet.com*. CNET. Retrieved October 13, 2019.
36. ^ *Knight, Will (August 7, 2018).* "The Defense Department has produced the first tools for catching deepfakes". MIT Technology Review.
37. ^ *Li, Bonnie; François-Lavet, Vincent; Doan, Thang; Pineau, Joelle (February 14, 2021). "Domain Adversarial Reinforcement Learning".* arXiv:2102.07097 *[*cs.LG*]*.
38. ^ *Bisneto, Tomaz Ribeiro Viana; de Carvalho Filho, Antonio Oseas; Magalhães, Deborah Maria Vieira (February 2020). "Generative adversarial network and texture features applied to automatic glaucoma detection".* Applied Soft Computing. *90: 106165*. doi:10.1016/j.asoc.2020.106165. S2CID 214571484.

39. ^ *Wei, Jerry (July 3, 2019).* "Generating Shoe Designs with Machine Learning". Medium. Retrieved November 6, 2019.
40. ^ *Greenemeier, Larry (June 20, 2016).* "When Will Computers Have Common Sense? Ask Facebook". Scientific American. Retrieved July 31, 2016.
41. ^ "3D Generative Adversarial Network". 3dgan.csail.mit.edu.
42. ^ *Achlioptas, Panos; Diamanti, Olga; Mitliagkas, Ioannis; Guibas, Leonidas (2018). "Learning Representations and Generative Models for 3D Point Clouds".* arXiv:1707.02392 *[*cs.CV*].*
43. ^ *Vondrick, Carl; Pirsiavash, Hamed; Torralba, Antonio (2016).* "Generating Videos with Scene Dynamics". carlvondrick.com. arXiv:1609.02612. Bibcode:2016arXiv160902612V.
44. ^ *Antipov, Grigory; Baccouche, Moez; Dugelay, Jean-Luc (2017). "Face Aging With Conditional Generative Adversarial Networks".* arXiv:1702.01983 *[*cs.CV*].*
45. ^ *Feng, Yunhe; Chen, Zongyao; Wang, Dali; Chen, Jian; Feng, Zhili (January 2020).* "DeepWelding: A Deep Learning Enhanced Approach to GTAW Using Multisource Sensing Images". IEEE Transactions on Industrial Informatics. *16 (1): 465–474.* doi:10.1109/TII.2019.2937563. ISSN 1551-3203. S2CID 202783302.
46. ^ *Feltus, Christophe (December 2021). "LogicGAN–based Data Augmentation Approach to Improve Adversarial Attack DNN Classifiers".* Proceedings of the 2021 International Conference on Computational Science and Computational Intelligence (CSCI).
47. ^ *Wu, Abraham Noah; Biljecki, Filip (2022). "GANmapper: geographical data translation".* International Journal of Geographical Information Science*: 1–29.* arXiv:2108.04232. doi:10.1080/13658816.2022.2041643. S2CID 247012122.
48. ^ *Kang, Yuhao; Gao, Song; Roth, Rob (2019).* "Transferring Multiscale Map Styles Using Generative Adversarial Networks". International Journal of Cartography. *5 (2–3): 115–141.* arXiv:1905.02200. Bibcode:2019arXiv190502200K. doi:10.1080/23729333.2019.1615729. S2CID 146808465.
49. ^ *Wijnands, Jasper; Nice, Kerry; Thompson, Jason; Zhao, Haifeng; Stevenson, Mark (2019). "Streetscape augmentation using generative adversarial networks: Insights related to health and wellbeing".* Sustainable Cities and Society. *49: 101602.* arXiv:1905.06464. Bibcode:2019arXiv190506464W. doi:10.1016/j.scs.2019.101602. S2CID 155100183.
50. ^ *Ukkonen, Antti; Joona, Pyry; Ruotsalo, Tuukka (2020).* "Generating Images Instead of Retrieving Them: Relevance Feedback on Generative Adversarial Networks". Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*: 1329–1338.* doi:10.1145/3397271.3401129. hdl:10138/328471. ISBN 9781450380164. S2CID 220730163.
51. ^ *Padhi, Radhakant; Unnikrishnan, Nishant (2006). "A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear*

systems". Neural Networks. *19 (10): 1648–1660*. doi:10.1016/j.neunet.2006.08.010. PMID 17045458.

52. ^ "AI can show us the ravages of climate change". MIT Technology Review. *May 16, 2019*.
53. ^ *Christian, Jon (May 28, 2019)*. "ASTOUNDING AI GUESSES WHAT YOU LOOK LIKE BASED ON YOUR VOICE". *Futurism*.
54. ^ *Zhavoronkov, Alex (2019). "Deep learning enables rapid identification of potent DDR1 kinase inhibitors"*. Nature Biotechnology. *37 (9): 1038–1040*. doi:10.1038/s41587-019-0224-x. PMID 31477924. S2CID 201716327.
55. ^ *Gregory, Barber*. "A Molecule Designed By AI Exhibits 'Druglike' Qualities". Wired.
56. ^ *Mohammad Navid Fekri; Ananda Mohon Ghosh; Katarina Grolinger (2020)*. "Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks". Energies. *13 (1): 130*. doi:10.3390/en13010130.
57. ^ *Gutmann, Michael; Hyvärinen, Aapo*. "Noise-Contrastive Estimation" (PDF). International Conference on AI and Statistics.
58. ^ *Niemitalo, Olli (February 24, 2010)*. "A method for training artificial neural networks to generate missing data within a variable context". Internet Archive (Wayback Machine). Archived *from the original on March 12, 2012*. Retrieved February 22, 2019.
59. ^ "GANs were invented in 2010?". reddit r/MachineLearning. *2019*. Retrieved May 28, 2019.
60. ^ *Li, Wei; Gauci, Melvin; Gross, Roderich (July 6, 2013). "Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13"*. Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO 2013). *Amsterdam, The Netherlands: ACM. pp. 223–230*. doi:10.1145/2463372.2465801. ISBN 9781450319638.
61. ^ *Abu-Khalaf, Murad; Lewis, Frank L.; Huang, Jie (July 1, 2008). "Neurodynamic Programming and Zero-Sum Games for Constrained Control Systems"*. IEEE Transactions on Neural Networks. *19 (7): 1243–1252*. doi:10.1109/TNN.2008.2000204. S2CID 15680448.
62. ^ *Abu-Khalaf, Murad; Lewis, Frank L.; Huang, Jie (December 1, 2006). "Policy Iterations on the Hamilton–Jacobi–Isaacs Equation for* H_∞ *State Feedback Control With Input Saturation"*. IEEE Transactions on Automatic Control. doi:10.1109/TAC.2006.884959. S2CID 1338976.
63. ^ *Sajjadi, Mehdi S. M.; Schölkopf, Bernhard; Hirsch, Michael (December 23, 2016). "EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis"*. arXiv:1612.07919 *[*cs.CV*]*.
64. ^ "This Person Does Not Exist: Neither Will Anything Eventually with AI". *March 20, 2019*.
65. ^ "ARTificial Intelligence enters the History of Art". *December 28, 2018*.
66. ^ *Tom Février (February 17, 2019)*. "Le scandale de l'intelligence ARTificielle".
67. ^ "StyleGAN: Official TensorFlow Implementation". *March 2, 2019 – via GitHub*.
68. ^ *Paez, Danny (February 13, 2019)*. "This Person Does Not Exist Is the Best One-Off Website of 2019". Retrieved February 16, 2019.

69. ^ *BESCHIZZA, ROB (February 15, 2019).* "This Person Does Not Exist". Boing-Boing. Retrieved February 16, 2019.
70. ^ *Horev, Rani (December 26, 2018).* "Style-based GANs – Generating and Tuning Realistic Artificial Faces". Lyrn.AI. *Archived from* the original *on November 5, 2020.* Retrieved February 16, 2019.
71. ^ *Elgammal, Ahmed; Liu, Bingchen; Elhoseiny, Mohamed; Mazzone, Marian (2017). "CAN: Creative Adversarial Networks, Generating "Art" by Learning About Styles and Deviating from Style Norms".* arXiv:1706.07068 *[*cs.AI*].*
72. ^ *Cohn, Gabe (October 25, 2018).* "AI Art at Christie's Sells for $432,500". The New York Times.
73. ^ *Mazzone, Marian; Ahmed Elgammal (February 21, 2019).* "Art, Creativity, and the Potential of Artificial Intelligence". Arts. *8: 26.* doi:10.3390/arts8010026.
74. ^ *Kulp, Patrick (May 23, 2019).* "Samsung's AI Lab Can Create Fake Video Footage From a Single Headshot". AdWeek.
75. ^ *Yu, Yi; Canales, Simon (2021). "Conditional LSTM-GAN for Melody Generation from Lyrics".* ACM Transactions on Multimedia Computing, Communications, and Applications. *17: 1–20.* arXiv:1908.05551. doi:10.1145/3424116. ISSN 1551-6857. S2CID 199668828.
76. ^ "Nvidia's AI recreates Pac-Man from scratch just by watching it being played". The Verge. *May 22, 2020.*
77. ^ *Seung Wook Kim; Zhou, Yuhao; Philion, Jonah; Torralba, Antonio; Fidler, Sanja (2020). "Learning to Simulate Dynamic Environments with GameGAN".* arXiv:2005.12126 *[*cs.CV*].*
78. ^ *Donahue, Jeff; Krähenbühl, Philipp;* Darrell, Trevor *(2016). "Adversarial Feature Learning".* arXiv:1605.09782 *[*cs.LG*].*
79. ^ *Makhzani, Alireza; Shlens, Jonathon; Jaitly, Navdeep;* Goodfellow, Ian*;* Frey, Brendan *(2016). "Adversarial Autoencoders".* arXiv:1511.05644 *[*cs.LG*].*
80. ^ *Dumoulin, Vincent; Belghazi, Ishmael; Poole, Ben; Mastropietro, Olivier; Arjovsky, Alex; Courville, Aaron (2016). "Adversarially Learned Inference".* arXiv:1606.00704 *[*stat.ML*].*
81. ^ *Xi Chen; Yan Duan; Rein Houthooft; John Schulman;* Ilya Sutskever*;* Pieter Abeel *(2016). "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets".* arXiv:1606.03657 *[*cs.LG*].*
82. ^ *Zhirui Zhang; Shujie Liu; Mu Li; Ming Zhou; Enhong Chen (October 2018).* "Bidirectional Generative Adversarial Networks for Neural Machine Translation"
83. J. Bruna, P. Sprechmann, and Y. LeCun. Super-resolution with deep convolutional sufficient statistics. In International Conference on Learning Representations (ICLR), 2016.
84. J. Canny. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, (6):679–698, 1986.
85. Y. Chen, Y.-K. Lai, and Y.-J. Liu. Transforming photos to comics using convolutional neural networks. In International Conference on Image Processing, 2017.

86. R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A MATLAB-like environment for machine learning. In NIPS Workshop on BigLearn, 2011.
87. V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville. Adversarially learned inference. In International Conference on Learning Representations (ICLR), 2017.
88. L. Gatys, A. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2414–2423, 2016.
89. L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. arXiv preprint arXiv:1505.07376, 12, 2015.
90. L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman. Controlling perceptual factors in neural style transfer. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
91. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems 27, pages 2672–2680. 2014.
92. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770– 778, 2016.
93. A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In ACM SIGGRAPH, pages 327– 340, 1998.
94. S.-S. Huang, G.-X. Zhang, Y.-K. Lai, J. Kopf, D. Cohen-Or, and S.-M. Hu. Parametric meta-filter modeling from a single example pair. The Visual Computer, 30(6-8):673–684.
95. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning, pages 448– 456, 2015.
96. P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-toimage translation with conditional adversarial networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
97. J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In European Conference on Computer Vision, pages 694–711, 2016.
98. L. Karacan, Z. Akata, A. Erdem, and E. Erdem. Learning to generate images of outdoor scenes from attributes and semantic layouts. arXiv preprint arXiv:1612.00215, 2016.
99. A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105, 2012.
100. S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. IEEE Transactions on Neural Networks, 8(1):98–113, 1997. C. Ledig, L. Theis, F. Huszar, J. Caballero, A.

Cunningham, ′ A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

101. C. Li and M. Wand. Combining Markov random fields and convolutional neural networks for image synthesis. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2479–2486, 2016.

102. J. Liao, Y. Yao, L. Yuan, G. Hua, and S. B. Kang. Visual attribute transfer through deep image analogy. ACM Transactions on Graphics, 36(4):120, 2017.

103. R. R. Luque. The cel shading technique. Technical report, 2012.

104. A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In International Conference on Machine Learning, volume 30, 2013.

105. S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In International Conference on Machine Learning, 2016.

106. P. L. Rosin and J. Collomosse. Image and Video-Based Artistic Stylisation. Springer, 2013.

107. P. L. Rosin and Y.-K. Lai. Non-photorealistic rendering of portraits. In Workshop on Computational Aesthetics, pages 159–170, 2015.

108. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3):211–252, 2015.

109. T. Saito and T. Takahashi. Comprehensible rendering of 3- D shapes. In ACM SIGGRAPH, volume 24, pages 197–206, 1990.

110. E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa. Learning to simplify: Fully convolutional neural networks for rough sketch cleanup. ACM Transactions on Graphics, 35(4):121, 2016.

111. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

112. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1– 9, 2015.

113. J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. Video tooning. ACM Transactions on Graphics, 23(3):574–583, 2004.

114. H. Winnemoller, S. C. Olsen, and B. Gooch. Real-time video ¨ abstraction. ACM Transactions on Graphics, 25(3):1221– 1226, 2006. 9473

115. J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In Advances in Neural Information Processing Systems, pages 82–90, 2016.

116. L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via L0 gradient minimization. ACM Transactions on Graphics, 30(6):174, 2011.
117. M. Yang, S. Lin, P. Luo, L. Lin, and H. Chao. Semanticsdriven portrait cartoon stylization. In International Conference on Image Processing, 2010.
118. R. Yeh, C. Chen, T. Y. Lim, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with perceptual and contextual losses. arXiv preprint arXiv:1607.07539, 2016.
119. J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired imageto-image translation using cycle-consistent adversarial networks. In International Conference on Computer Vision, 2017.