UFCFSN-15-3 – Artificial Intelligence Documentation

Generative Task

Patryk Ostrowski 17015998

1. A brief description of an idea and aims

The aim of this project was to use the Generative Adversarial Network (GAN) to generate images of passenger cars. To do it a few hundred images of cars were collected. The idea was to use these images in some sort of multiplayer quiz-style mobile game.

2. Research into other related work

Research into other related work was conducted in order to find out how other people used DCGAN and what methods they used to get good results. It also gave an opportunity to explore different applications for DCGAN other than the creative field.

A really interesting application of Deep Convolutional Generative Adversarial Network (DCGAN) was presented by (Wei, Feihong, Victor and Yewen, 2018). In this paper, it was used as a method to improve CNN-Based image recognition of radar profiles used in the meteorological field to detect different weather conditions. The image recognition performance is mainly dependant on the number of images and quality of them. As DCGAN is used to generate images, sounds or videos it was used to create new image samples that were difficult to collect in this case. Therefore, using a DCGAN allowed training CNN on newly generated images. This allowed to enhanced classification model and improve the accuracy of image recognition.

Another usage of the DCGAN was presented in this paper (Qiufeng, Yiping and Jun, 2020). Similarly, to the previous example, it was used to generate images that could be later used in an image recognition system. The purpose of this project was to detect tomato leaf diseases which are something really important in the agriculture field. Standard data augmentation methods, for instance, rotation or flipping were fairly limited and could not achieve good generalization results. That is why the DCGAN was used, to generate augmented images which helped to achieve over 94% average identification accuracy. Generated images not only increased the data set but also provided more diversity which led to the model with a good generalization effect. They were also of better quality than real images and helped to reduce the cost of data collection.

Lastly, implementation of DCGAN was found in (Lu, Qi and Yu, 2021). This time it was used to personalize fashion designs and choose the best design from presented fashion sketches. The customer's interest in generated images is calculated by tracking eye movement and facial expressions. The most liked images are then feed back to the training data for iterative training. It aims to improve customer satisfaction using a few iterations with newly generated images. Tested results showed that the satisfaction rating of newly generated images went up.

3. Methods that were used

Firstly, the data set had to be collected. To do it a Google Chrome extension was used, called Image Downloader – Imageye. It helped with selecting appropriate images and downloading them. All of them were found on Google Images. For this project, only standard passenger cars were used in data set creation.

For image generation, Deep Convolutional Generative Adversarial Network (DSGAN) was used. It is an extension of GAN architecture for using deep convolutional neural networks for image data generation. It uses a generator that learns to create images and a discriminator which learns to distinguish real images from generated ones.

```python
def make_generator_model():
    model = tf.keras.Sequential()

    n = IMAGE_SIZE // 4

    model.add(layers.Dense(n*n*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((n, n, 256)))

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(IMAGE_CHANNELS, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))

    return model
```

Figure 1: Final Generator

4. Technical description of implementation

The generator (see Figure 1) uses a sequential model which is made out of a few layers. The first one is a dense layer that takes the seed as an input. Another important layer is called Conv2DTranspose. There is a few of them in total and they are responsible for producing an image from the seed. This type of layer is used when there is a need of changing the direction of normal convolution. In this case, a few parameters are passed. First one is a filter which is the dimensionality of the output space. Its value decreases in the lower transpose layers. Second one is Kernal size which in this implementation is 5 by 5 size. The Kernal is a square that is applied to the original image pixels. Its values and pixels values are multiplied and sum together to create a new pixel of a convoluted image.

The discriminator also consists of a few types of layers including Conv2D, which applies a specified kernel to the image or LeakyReLU which is an activation function that allows a small gradient when the value is below zero or returns the value if its greater than zero. It can be described as follows:

```
f(x) = alpha * x if x < 0
f(x) = x if x >= 0
```

Lastly, there is a dropout layer that sets random input units to 0.

In order to train the generator for an extended period of time using Google Colaboratory Notebook, a separate function for saving a checkpoint was implemented. It takes the number of epochs that need to pass to save the checkpoint. Firstly, it deletes all files from the given directory (where checkpoints are saved) and then saves the latest one (see code below).

```python
def saveLatestCheckpoint(howOften, epoch, checkpoint):

    if (epoch + 1) % howOften == 0:

        deleteLastCheckpoint()

        checkpoint.save(file_prefix = checkpoint_prefix)
```

To be able to generate a gif showing the process of training, images are saved on Google Drive. As before, the frequency of saving can be specified. To make all images more manageable the epoch number and current time are saved in their name (see code below).

```python
def saveImages(plt, epoch):

    now = datetime.now()

    current_time = now.strftime("%H:%M:%S")


    plt.savefig('gdrive/MyDrive/ai_saved_images/image_at_epoch_{:04d}_{:s}.png'.format(epoch + 1, current_time))
```

5. Results with testing dataset and discussion of findings

In this project, a few different image sizes were tested to see which ones would be the best suitable in terms of time to complete and final results.

Firstly, following a tutorial, the image size was set to 32 by 32 pixels. Time per one epoch was quite small, averaging about 3-4 seconds. Due to the aim of the project, which is generating car images, this size turned out to be too small. Original images were in different sizes and presented cars in much greater

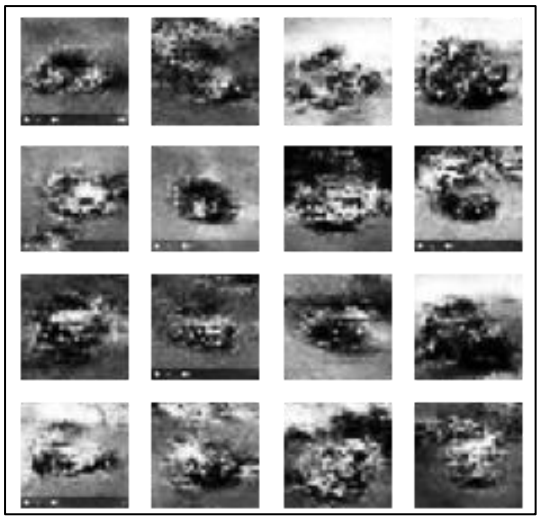

Figure 2: 32 by 32 pixels, around 4500 epochs

resolution with a lot of characteristic features, such as the shape of the vehicle. Despite the fact that images in Figure 2 were produced for about 4500 thousand epochs, results were not great. Some sort of objects could be seen in the centre of images but without any noticeable similarities to vehicles.

After the initial results, the image size was changed to 128 by 128 pixels. That meant the generated images were going to be four times bigger than previous ones. At this stage of the project saving checkpoints was not implemented and there was a limit to how long a single Google Colaboratory session could last. Therefore, changing the size, resulted in the much greater time needed to complete one epoch (about 65 seconds). In the end, it was only possible to complete 700 epochs and generated image is shown in Figure 3 showed results that were even worse than before.

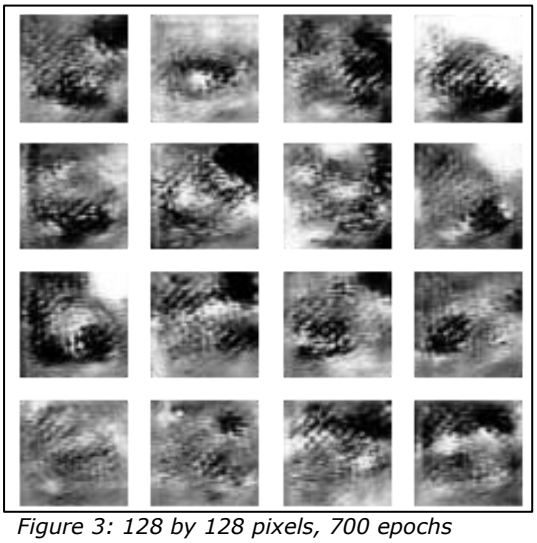

Figure 3: 128 by 128 pixels, 700 epochs

Taking into consideration previous results, this time image size was set to 64 by 64 pixels. That significantly reduced time needed for a single epoch to about 20 seconds and also left enough room to include all key information about generated vehicles. In the first training run, about 2100 epochs were completed giving quite good results (see Figure 4).

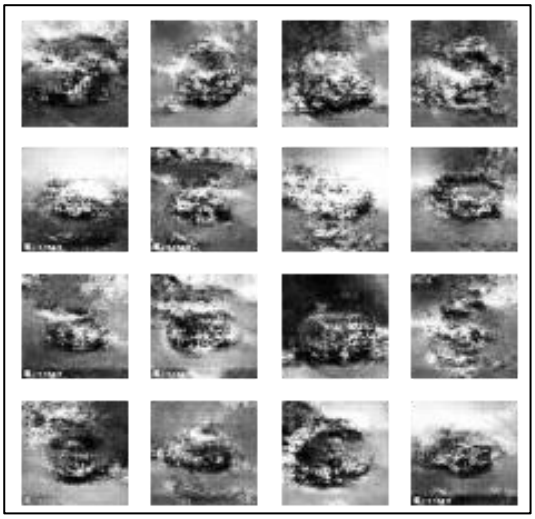

Figure 4: 64 by 64 pixels, around 2100 epochs

Lastly, generated images were set to be in colour and with implemented checkpoint system program was run for a few days, reaching about 8800 epochs.

A few images from different points in time can be seen in Figure 5. The best results were achieved at around 6500 epochs with slightly worse at the end of training.

6. Demonstrated application to a creative task

This project was created with the aim to use generated images in a mobile video game. It would be a quiz-style game where players are shown an image and have to choose if it is real or fake. The single game would consist of a few rounds, each of them having a certain amount of time to select the right answer. Time would decrease further into the rounds. Due to the fact that the game would be made for mobile devices, it would consist only of multiplayer mode. Therefore, people could play with friends or random people. It would also include global and local (friends) statistic of the player's score. The current score could be improved not only by wining with other people but also the way the game was won. For instance, taken time, number of correct

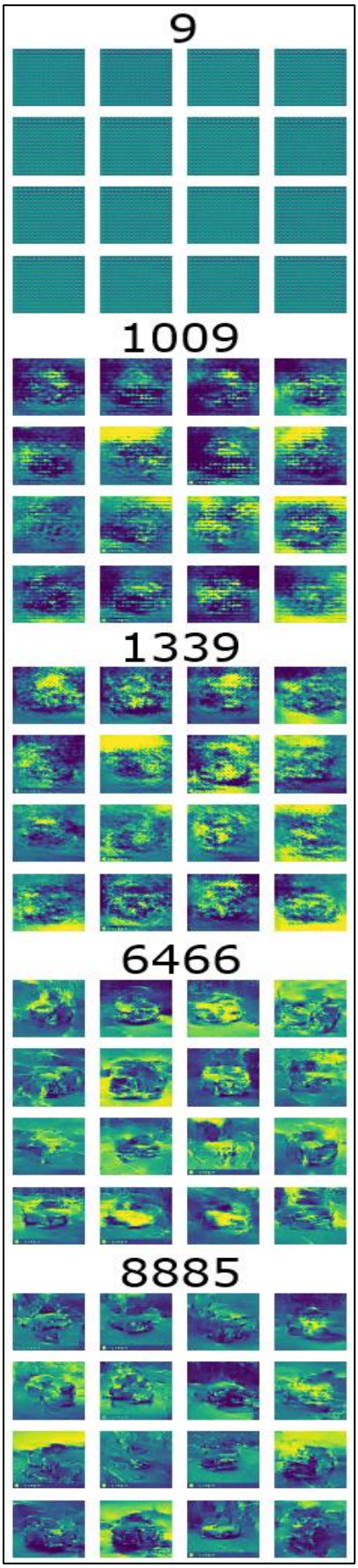

Figure 5: Images at different epoch

answers or opponent's main score. The game would allow creating an account and adding friends to see their scores and mentioned earlier statistics.

7. Conclusion and suggestion for further work

In conclusion this Deep Convolutional Generative Adversarial Network was able to generate images which started to show some sort of vehicles with similar features that real cars have. To make the results better it would be a good idea to collect more images of cars and train the model for a longer period of time.

For the future work, it would be interesting to expand data set by adding more types of vehicles, such as SUVs or Vans. It would be also beneficial to implement transfer learning which could greatly increase performance of the DCGAN.

8. Bibliography

Wei, F., Feihong, Z., Victor, S. and Yewen, D., 2018. A method for improving CNN-based image recognition using DCGAN. *Computers, Materials and Continua*, [online] 57(1), pp.167-178. Available at: <https://doi.org/10.32604/cmc.2018.02356> [Accessed 9 May 2021].

Qiufeng, W., Yiping, C. and Jun, M., 2020. DCGAN-Based Data Augmentation for Tomato Leaf Disease Identification. *IEEE Access*, [online] 8, pp.98716-98728. Available at: <https://ieeexplore.ieee.org/abstract/document/9099295/citations#citations> [Accessed 13 May 2021].

Lu, C., Qi, F. and Yu, C., 2021. Intelligent Clothing Interaction Design and Evaluation System Based on DCGAN Image Generation Module and Kansei Engineering. *Journal of Physics: Conference Series*, [online] 1790(1), p.012025. Available at: <https://iopscience.iop.org/article/10.1088/1742-6596/1790/1/012025/pdf> [Accessed 11 May 2021].