Comparing CDCGAN with a new PCDCGAN architecture

Eran Shmuel (ID. 301796561), Moshe Binieli (ID. 311800668)

Final project report for Computer Vision, BIU 2021.

1 Introduction

In this project, we compared the traditional Conditional Deep Convolutional Generative Adversarial Network (CDCGAN) and a new framework Predictive Deep Convolutional Generative Adversarial Network (PCDCGAN). The goal of our project is to come up with a new architecture for CGAN, which might give ideas for other GAN frameworks. We tested the new method on the "Real-world Affective Faces Database" dataset[7] (RAF-DB). The dataset contains 15,000 face images with different types of emotions: surprise, fear, disgust, happiness, sadness, anger, and neutral. The goal of both frameworks is to create new images which look like real images and describe the specific emotion that is given to the network. From our experiences, the PCDCGAN framework has multiple advantages over the CDCGAN framework, and it performs better according to Fréchet Inception Distance and Inception Score evaluation methods. Because we used the same generator for both frameworks and a very similar discriminator, those results are very reliable.

1.1 Related Works

A Generative Adversarial Network (GAN) is a framework in the deep learning field. The framework contains two neural networks: the generator and the discriminator[1]. The goal of the generator is to create images that fool the discriminator, and the goal of the discriminator is to distinguish between real images and the fake images that were created by the generator.

Conditional Generative Adversarial Network (CGAN) is very similar to the GAN framework except that both the generator and the discriminator receive a label in addition to their input[2]. The discriminator aims to maximize the function Log(D(x/y))+Log(1–D(G(z/y))), while the generator tries to maximize the function Log(D(G(z/y))).

Deep Convolutional Adversarial Network (DCGAN), is an implementation of the GAN framework which uses only Deep Convolutional layers in the generator and discriminator[3]. This architecture was also implemented in a CGAN framework and proved to be effective[4].

2 Solution

2.1 General approach

In this project, we implemented Conditional Deep Convolutional Generative adversarial network (CDCGAN) and a new framework Predictive Deep Convolutional Generative Adversarial Network (PCDCGAN) which is very similar to the CDCGAN architecture, with three exceptions:

1. The discriminator doesn't get y as his input, instead he outputs a predictions vector in addition to its regular output.
2. The generator tries to minimize the loss function:

$$-Log(D(G(z/y))) + alpha * crossEntropy(D(G(z/y)).preds, y)$$

when D(G(z/y)).preds is the second output from the discriminator.

3. The discriminator tries to minimize the function:

$$-Log(D(x)) - log(1–D(G(z/y))) + beta * crossEntropy(D(x).preds, y)$$

We also tried to use the MSE loss function for the prediction as part of the loss function, but it provided bad results and the framework wasn't stable enough.

2.2 Design

We used the same generator for both frameworks (figure 1). The generator gets a latent vector z and a label y and outputs an image with (3, 100, 100) dimensions. It uses 6 layers of transposed convolution, with batch normalization on every layer besides the output layer. It also uses ReLU activation function on every layer, besides the output layer which uses Tanh activation function.

Layer	Input shape	stride	padding	Kernel size	Batch norm	activation	Output shape
TConv1	[107,1,1]	1	0	4	✓	Relu	[12*scale,4,4]
TConv2	[12*scale,4,4]	1	0	3	✓	Relu	[8*scale,6,6]
TConv3	[8*scale,6,6]	2	1	4	✓	Relu	[4*scale,12,12]
TConv4	[4*scale,12,12]	2	1	4	✓	Relu	[2*scale,24,24]
TConv5	[2*scale,24,24]	2	0	4	✓	Relu	[1*scale,50,50]
TConv6	[3,100,100]	2	1	4	×	tanh	[3,100,100]

Figure 1: Generator model for both architectures.

The discriminator is very similar in both frameworks (figure 2), the only differences between them are in the input and the output layers. In the CDCGAN framework, we used an embedding layer and a linear layer which maps the label input to another image channel. In the LCDCGAN framework, we used a linear layer before the last convolution layer which outputs the network label predictions. The other layers are similar to the generator in reverse order, except that we used convolution layers instead of transposed convolution, and leaky ReLU activation function for all the layers besides the output layer which uses Sigmoid activation function.

2.3 Generator's label probabilities

One very important thing to notice when choosing class labels (y) for the generator is the probability for each class. Because the dataset's class distribution is unbalanced, choosing y with the uniform distribution means that the discriminator can cheat, and predict that when y has high probability in the dataset it is a real image, and when it has low probability it is a generated image. This leads to a bad loss function for the generator. Officially, it's important that for every $y \in labels$, $p(real/y) = p(fake/y)$. We solved the problem by sampling y with the dataset's class distribution. This method is more important for the CDCGAN framework, because the discriminator gets the exact value of y, but it also helped the PCDCGAN framework.

3 Experimental results

We compared the results using the Fréchet Inception Distance and Inception Score (figure 3). PCDCGAN got better results with both evaluation methods. We used the Inception V3 model for computing both scores. In order to fit the model to our problem, we took a pre-trained model, replaced the last linear layer and trained it with our data. For the Fréchet Inception Distance computation, we used both the trained and pre-trained

Layer	Input shape	stride	padding	Kernel size	Batch norm	activation	Output shape
Conv1	[3,100,100]	2	1	4	✓	Leaky Relu	[scale,50,50]
Conv2	[scale,50,50]	2	0	4	✓	Leaky Relu	[2*scale,24,24]
Conv3	[2*scale,24,24]	2	1	4	✓	Leaky Relu	[4*scale,12,12]
Conv4	[4*scale,12,12]	2	1	4	✓	Leaky Relu	[8*scale,6,6]
Conv5	[8*scale,6,6]	1	0	3	✓	Leaky Relu	[12*scale,4,4]
Conv6	[12*scale,4,4]	1	0	4	×	sigmoid	Output [1]
Linear	[12*scale*4*4]	-	-	-	×	×	Preds [class_size=7]

Figure 2: LCDCGAN discriminator

models and compared the images with 20 batches of 3000 images. For the Inception Score, we tested the images with 10 batches of 5000 images.

3.1 The importance of alpha

For the PCDCGAN framework, we used different values for alpha, which had a big influence on the results. When using alpha=0.01 we received the best Fréchet Inception Distance score in the pre-trained classifiers, but alpha=0.02 gave the best inception score and the best Fréchet Inception Distance in the trained classifier. All PCDCGAN models gave better results than the CDCGAN model with the three evaluation methods (figure 3).

Framework	Alfa	Beta	Inception Score	Fréchet Inception Distance	Fréchet Inception Distance pretrained
CDCGAN	-	-	1.95	234.45	71.91
PCDCGAN	0.01	0.3	2.63	118.58	64
PCDCGAN	0.02	0.3	2.69	116.58	66.14
PCDCGAN	0.05	0.3	2.44	129.94	70.74

Figure 3: Evaluation of both frameworks

3.2 Images creation

When creating images, we wanted to show how the generator created images with the same latent vector but different labels. The results show that for most of the time it looks like it's the same person, with different types of emotions (figures 4, 5, 6).

4 Discussion

From the 2 evaluation methods and our own eyes, we concluded that PCDCGAN is an improved version of CDCGAN in the RAF datasets. Further research should be made using this framework with other, perhaps

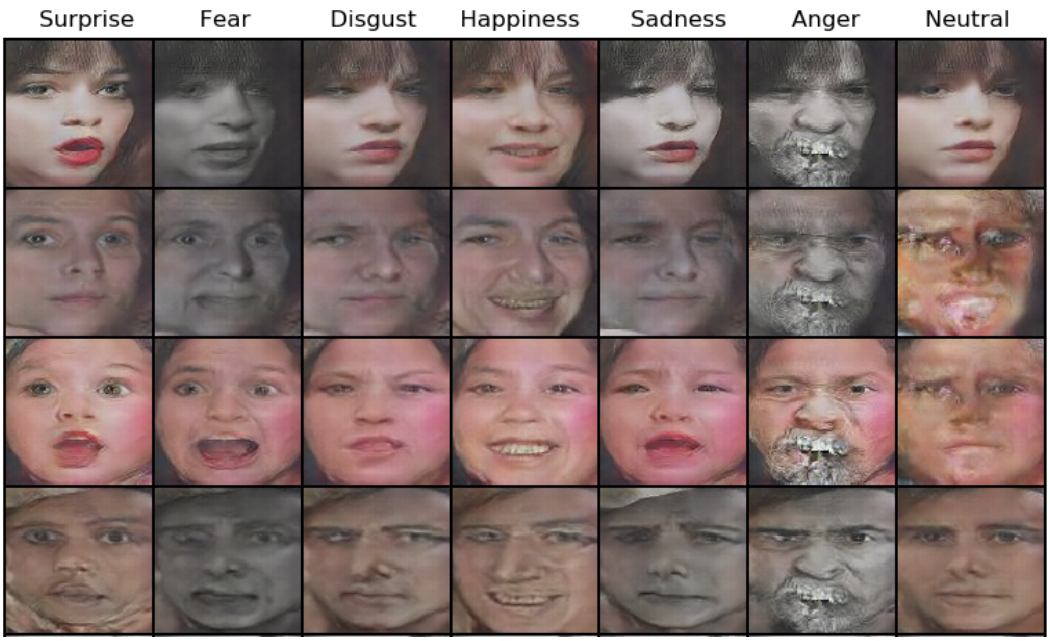

Figure 4: "PCDCGAN images with alpha=0.02"

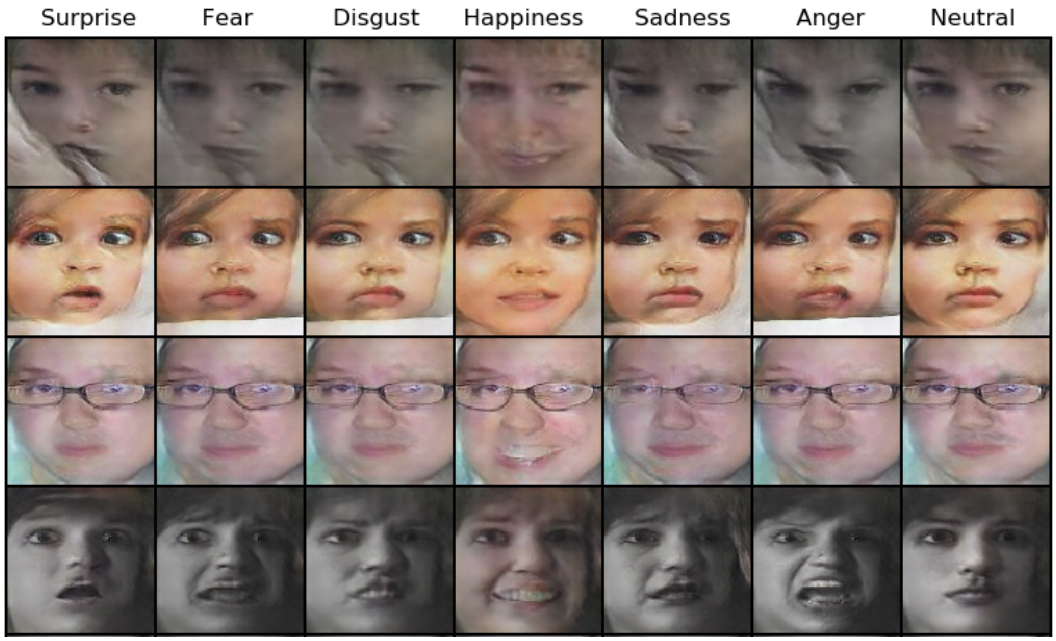

Figure 5: "PCDCGAN images with alpha=0.01"

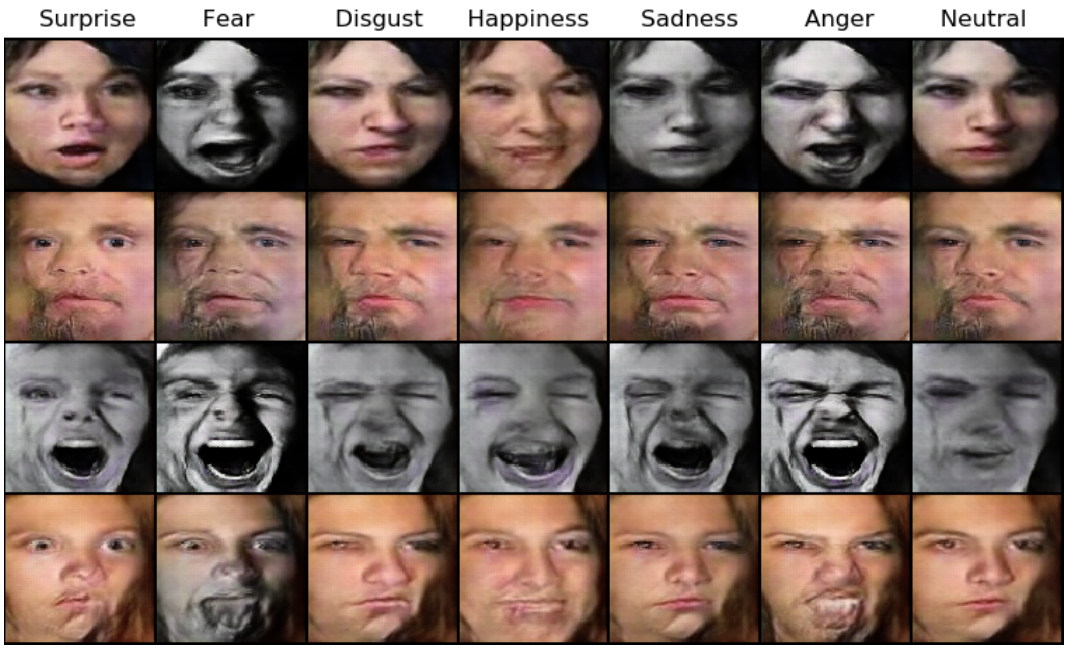

Figure 6: "CDCGAN images"

bigger datasets, but the results were unequivocal. The only downside of the PCDCGAN is the need to tune 2 more hyper parameters, but from our experience setting beta to 0.3 is good enough, and the real challenge is to find a good value for alpha.

4.1 Further research

We believe that there are advantages for using a pre-trained classifier for computing the predictions instead of using the discriminator for it. While the training might be a little slower, we think it can increase the stability of the models. In this case, the discriminator will only have to predict if the image is real or fake like in the traditional GAN, and the classifier will be used to compute the cross entropy loss between the label and the predictions of the generator.

5 Code

The code can be viewed in the GitHub repository: https://github.com/eran88/PCDCGAN

We used the following online codes:

1. Fréchet Inception Distance function
2. Inception Distance function
3. Weights initialization function

6 References

1. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio: Generative Adversarial Nets, 2014
2. Mehdi Mirza, Simon Osindero: Conditional Generative Adversarial Nets, 2014
3. Alec Radford, Luke Metz, Soumith Chintala: Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

4. Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, Jose M. Álvarez: Invertible Conditional GANs for image editing, 2016

5. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen: Improved Techniques for Training GANs

6. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler: GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2018

7. Li, Shan and Deng, Weihong and Du, JunPing: Reliable Crowdsourcing and Deep Locality-Preserving Learning for Expression Recognition in the Wild, 2017