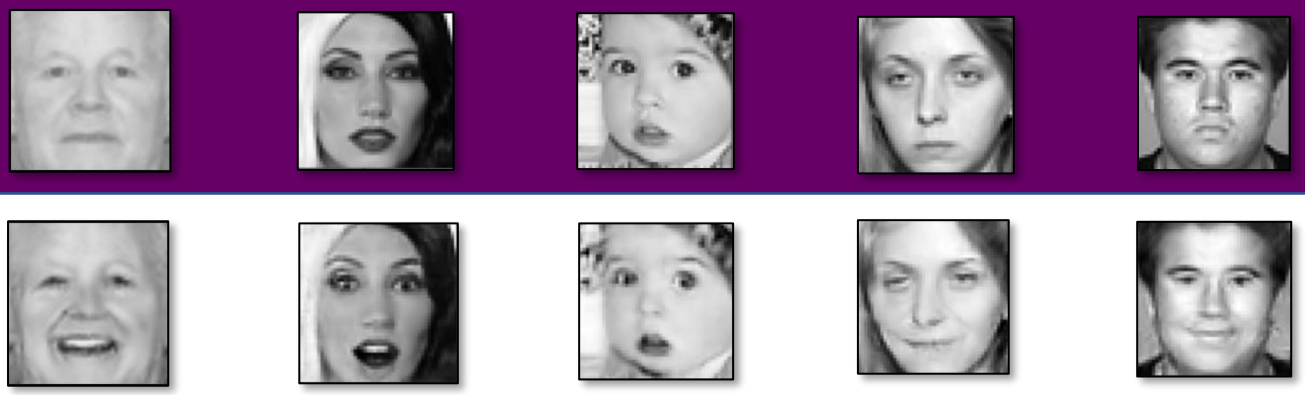
Data Augmentation Using GANs



Project 236754

Dima Birenbaum

Supervisors:

Yaron Honen, Gary Mataev

Contents

Abstract

The main goal is to generate synthetic data for projects, in the machine learning field, that deals with face emotions classification.

To classify images with multiple class labels using only a small number of labeled examples is a difficult task. Especially when the label (class) distribution is imbalanced. In face emotion classification we have imbalanced label distribution because some classes of emotions are relatively rare comparing to others. For example, disgust emotion is more rare than happy or sad.

In this work, we propose a data augmentation method using generative adversarial networks (GAN). It can complement and complete the data manifold and find better margins between neighboring classes. Specifically, for this task, we are using classifiers based on Convolutional Neural Networks (CNN) and a variation of cycle-consistent adversarial networks such as CycleGAN, Improved CycleGAN, and The Wasserstein CycleGAN. The CycleGAN is a direct implementation of Emotion Classification with Data Augmentation Using Generative Adversarial Networks paper.

In order to improve the results and avoid problems that we faced we employ different variations of CycleGANs. We show, that our empirical results can obtain a ~5% increase in the classification accuracy, after employing the GAN-based data augmentation techniques.

The Data

We are using FER2013 dataset. This dataset contains images of size 48x48 pixels and 7 emotion expressions: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.

The data distribution in FER2013 dataset is:

Emotion	Amount
Angry	4593
*Disgust	547
Fear	5121
Happy	8989
Sad	6077
Surprise	4002
Neutral	6198

* - as you can see, the disgust class, contains a small number of images, which makes it hard to learn to any neural network. The approach in this project, generates additional data from Disgust Class distribution, in order to unify data distribution.

Data samples from FER2013

In this work, we propose 3 different approaches for augmented data generation.

The Cycle GAN Model

The project uses CycleGAN architecture, as a method, for image-to-image style transfer.

CycleGAN - is a two-way GAN, that consists of 2 *Discriminators* and 2 *Generators*.

The idea is to transfer input from one domain to another back and forth.

Theory Background:

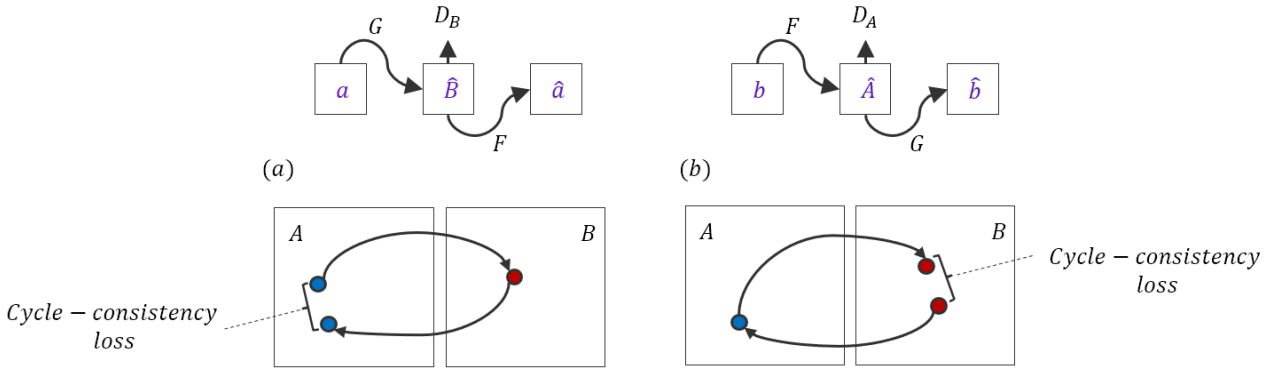
Domains A, B, mapping functions: $G: A \to B$, $F: B \to A$, associated adversarial discriminators D_A, D_B

D_B encourages G to translate A into outputs indistinguishable from domain B, and vice versa, for D_A and F.

To further regularize the mappings, used *two-cycle consistency loss*. The main intuition, that when translating from one domain to another and back again, the model should arrive at where it started. Two-cycle consistency loss consists of:

(a) Forward cycle-consistency: $a \to G(a) \to F(G(a)) \approx a$

(b) Backward cycle-consistency: $b \to F(b) \to G(F(b)) \approx b$



Target and loss functions

- Adversarial loss:

$$\boldsymbol{\mathcal{L}}_{GAN}(G, D_A, A, B) = \mathbb{E}_{a\sim p_{data}(a)}[(D_A(a) - 1)^2] + \mathbb{E}_{b\sim p_{data}(b)}\left[\left(D_A(G(b))\right)^2\right]$$

- Cycle consistency loss:

$$\boldsymbol{\mathcal{L}}_{cyc}(G, F) = \mathbb{E}_{a\sim p_{data}(a)}\left[\left\|F(G(a)) - a\right\|_1\right] + \mathbb{E}_{b\sim p_{data}(b)}\left[\left\|G(F(b)) - b\right\|_1\right]$$

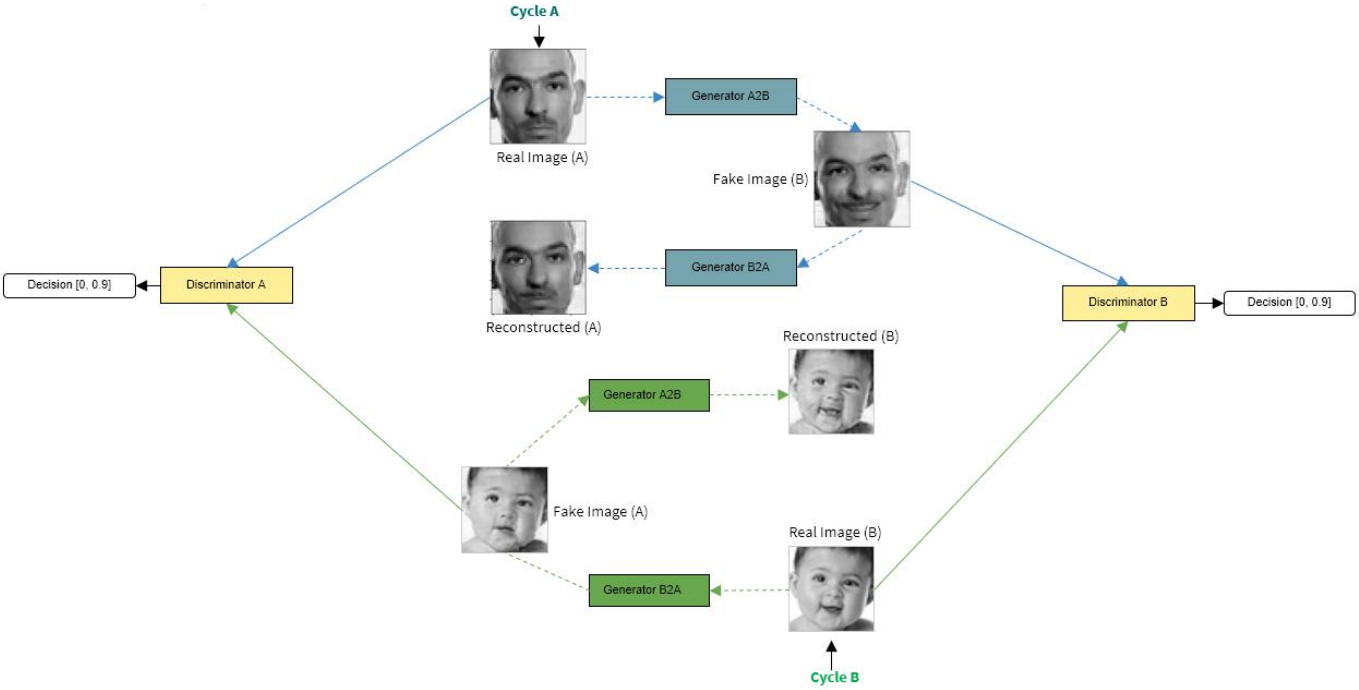
- Full objective:

$$\boldsymbol{\mathcal{L}}(G, F, D_A, D_B) = \boldsymbol{\mathcal{L}}_{GAN}(G, D_B, A, B) + \boldsymbol{\mathcal{L}}_{GAN}(F, D_A, A, B) + \lambda \boldsymbol{\mathcal{L}}_{cyc}(G, F)$$

- Target function:

$$\hat{G}, \hat{F} = arg \min_{G,F} \max_{D_A, D_B} \boldsymbol{\mathcal{L}}(G, F, D_A, D_B)$$

What is Cycle in CycleGAN?



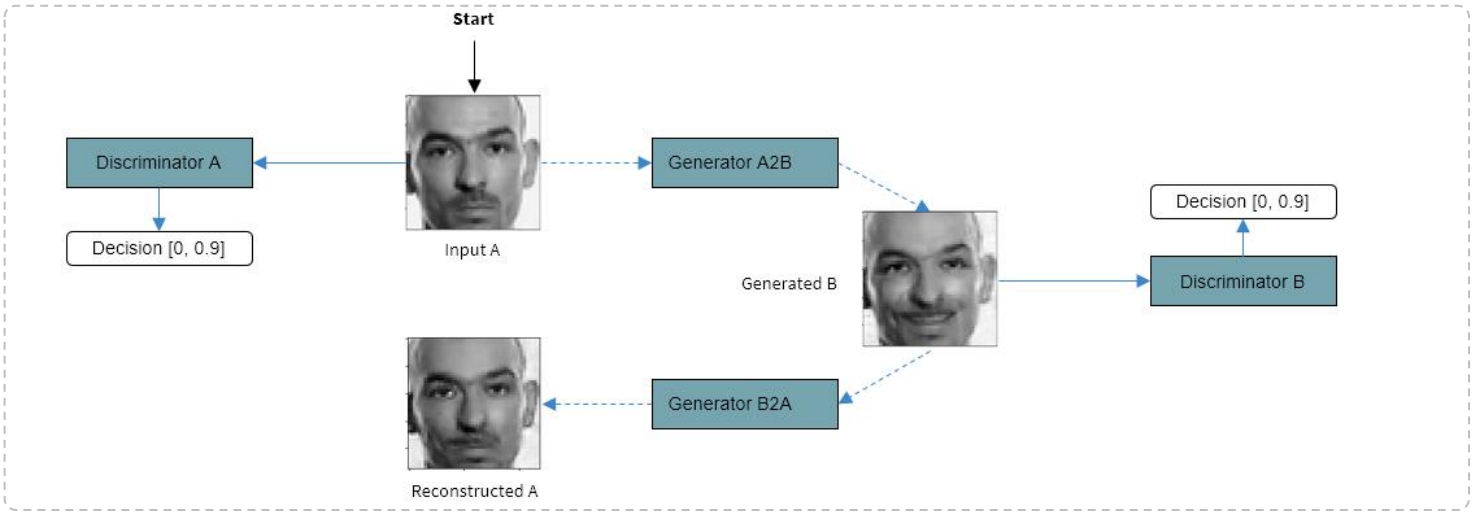
In a paired dataset, every image, say img_A, is manually mapped to some image, say img_B, in target domain, such that they share various features. Features that can be used to map an image (img_A/img_B) to its correspondingly mapped counterpart (img_B/img_A).

Basically, pairing is done to make input and output share some common features. This mapping defines the meaningful transformation of an image from one domain to another. So, when we have paired dataset, generator must take an input, say $input_A$, from domain D_A and map this image to an output image, say gen_B, which must be close to its mapped counterpart. But we don't have this luxury in the unpaired dataset, there is no pre-defined meaningful transformation that we can learn, so, we will create it. We need to make sure that there is some meaningful relation between input image and generated. So, Generator will map input image ($input_A$) from domain D_A to some image in the target domain D_B, but to make sure that there is a meaningful relationship between these images, they must share some feature, features that can be used to map this output image back to the input image, so there must be another generator that must be able to map back this output image back to the original input. So, you can see this condition defining a meaningful mapping between $input_A$ and gen_B.

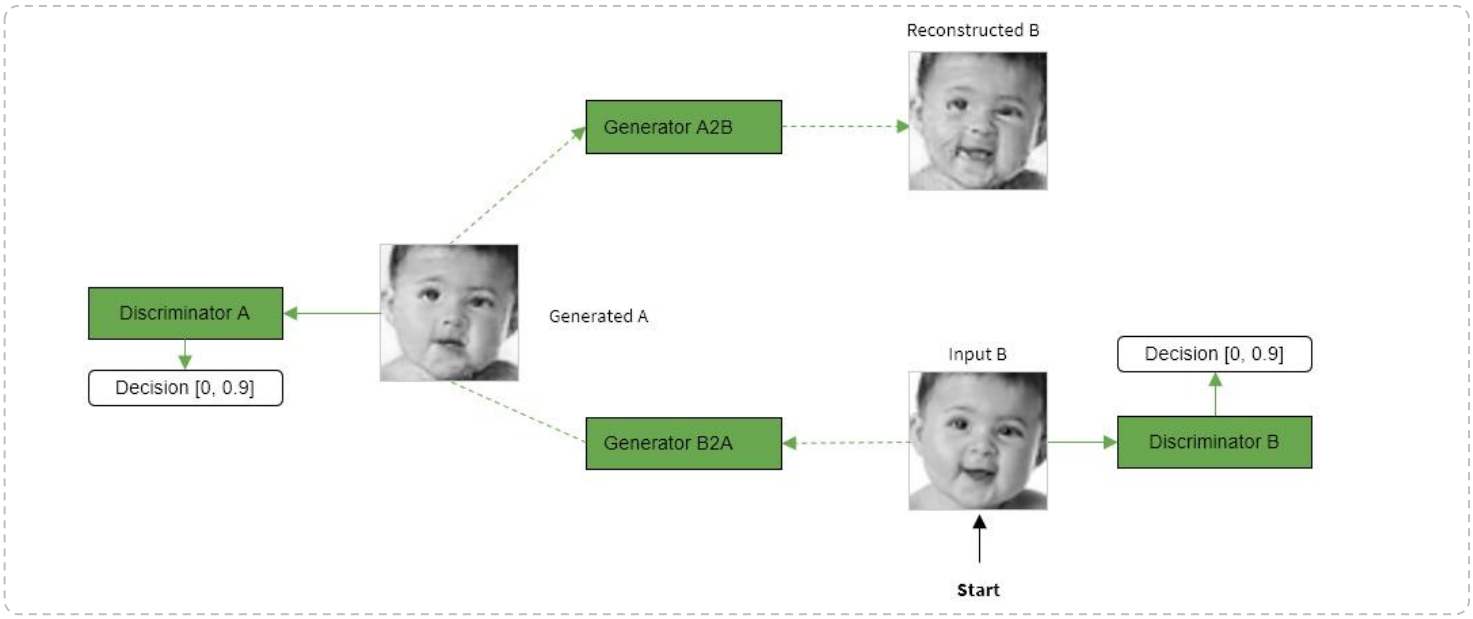
In a nutshell, the model works by taking an input image from domain D_A which is fed to first generator $GeneratorA2B$ whose job is to transform a given image from domain D_A to an image in the target domain D_B. This new generated image is then fed to another generator $GeneratorB2A$ which converts it back into an image, $Cyclic_A$, from the original domain D_A. And as we discussed before, this output image must be close to the original input image to define a meaningful mapping that is absent in an unpaired dataset.

As you can see in above figure, two inputs are fed into each discriminator (one is original image corresponding to that domain and other is the generated image via a generator) and the job of discriminator is to distinguish between them, so that discriminator is able to defy the adversary (in this case generator) and reject images generated by it. While the generator would like to make sure that these images get accepted by the discriminator, so it will try to generate images which are very close to original images in Class D_B. (In fact, the generator and discriminator are playing a game whose Nash equilibrium is achieved when the generator's distribution becomes same as the desired distribution)

Forward and Backward Cycles zoom in



The Model: Forward Cycle A2B

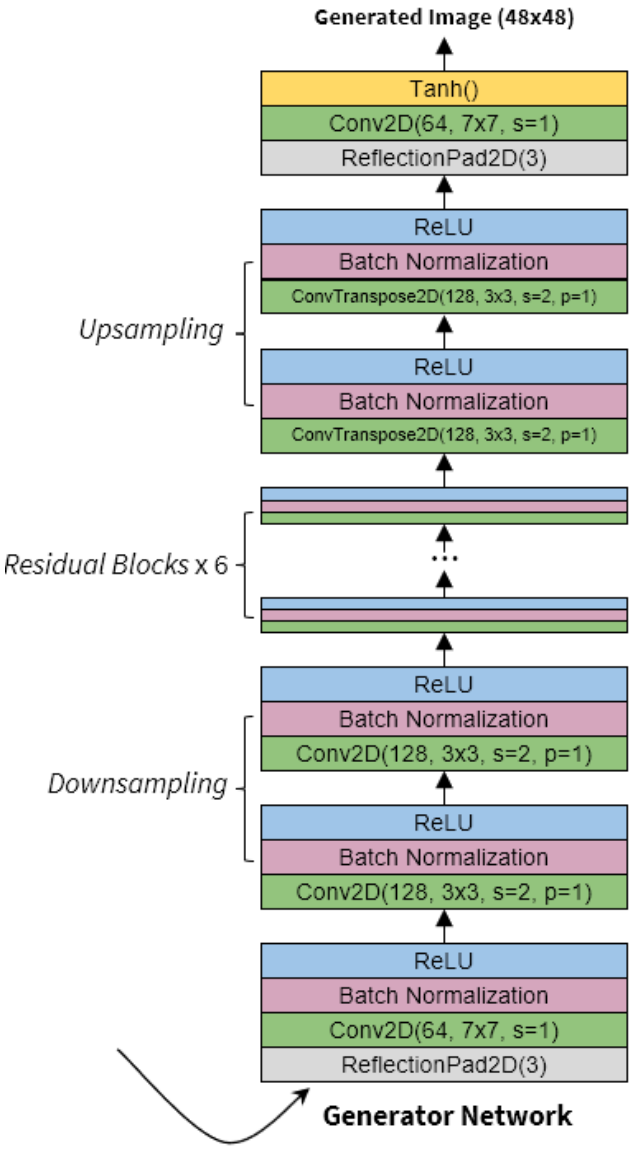


The Model: Backward Cycle B2A

The Generator:

The Generator consists of 3 parts:

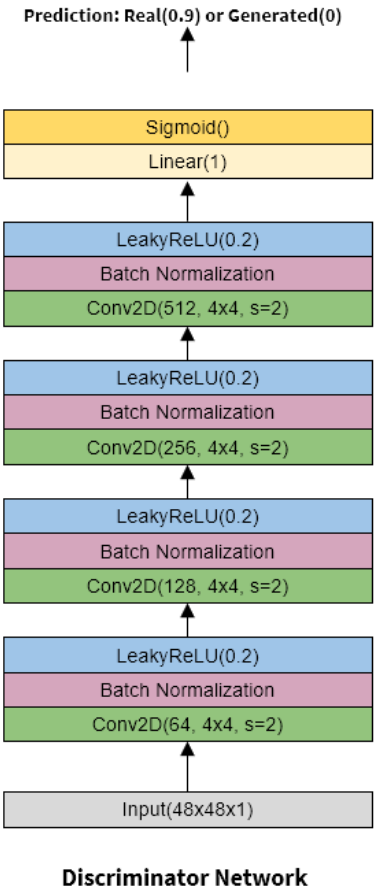
- Decode (downsampling)
- Transferring (6 residual blocks)
- Encode (upsampling).



The Discriminator:

The Discriminator - a simple CNN network,

that determinates if the image is fake or real.



The Problems with native CycleGAN approach

Problems that have been encountered during the project:

- "Dirty" dataset, unbalanced classes, mislabeled data
- Similarity between classes (for example fear-angry, sad-neutral)
- Lack of data (for example Disgust Class – 550 images)
- During the training, the Discriminator learns faster than the Generator.
- Vanishing gradient
- Quality and artifacts of output images

Possible solutions are:

- Data augmentation, transform on training
- Different learning rates for generator and discriminator: 0.0002, 0.0001
- Learning rate decay
- Soft labels for discriminator: Real target: 0.9 (instead of 1)
- Or, a different approach…

The Improved Cycle GAN Model

Sometimes early in training, we observe cases where the generated image is very unrealistic and cycle consistency doesn't even make sense. In fact, once stuck in local modes, the generators are unlikely to escape due to the cycle consistency loss. Therefore, enforcing cycle consistency on cycles where generated images are not realistic hinders training. To solve this issue, we need to weight cycle consistency loss by the quality of generated images, which we obtain using the discriminators' outputs. So the new cycle consistency loss:

$$\mathcal{L}_{cyc}(G,F,D_A,A,\gamma) = \mathbb{E}_{a\sim p_{data}(a)}\left[D_A(a)\cdot\left[\gamma\cdot\left\|f_{D_A}\left(F\left(G(a)\right)\right)-f_{D_A}(a)\right\|_1+(1-\gamma)\cdot\left\|F\left(G(a)\right)-a\right\|_1\right]\right]$$

Where: $\gamma \in [0,\ 1]$ – *linearly increase with epochs, to 1,* $f_{D_{(\cdot)}}$ *- is the future extractor using the last layer of* $D_{(\cdot)}$

Such a change dynamically balances GAN loss and cycle consistency loss early in training. It essentially urges the generators to first focus on outputting realistic images and to worry about cycle consistency later. It is worthwhile to note that the gradient of this loss should not be backward propagated to $D(\cdot)$ because cycle consistency is a constraint only on generators.

So final objective updated to:

$$\mathcal{L}(G,F,D_A,\ D_B) = \mathcal{L}_{GAN}(G,D_B,\ A,B) + \mathcal{L}_{GAN}(F,D_A,\ A,B) + \lambda\cdot\mathcal{L}_{cyc}(G,F,D_A,A,\gamma) + \lambda\cdot\mathcal{L}_{cyc}(G,F,D_B,B,\gamma)$$

So how can we improve the stability of training? The answer is - The Wasserstein distance.

Wasserstein CycleGAN - is a two-way Wasserstein GAN, that consists of 2 *Critics* and 2 *Generators*.

The idea is, for distribution of mass $\mu(x)$ on a space X, we wish to transport the mass in such a way that it is transformed into the distribution $\nu(x)$ on the *same* space.

Our main goal and bottleneck are to create data, that has the same distribution as the targeted domain, one of the most suitable and available methods for this task is *The Wasserstein distance.*

The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution $\boldsymbol{q}$ to the data distribution $\boldsymbol{p}$. The Wasserstein distance for the real data distribution $\boldsymbol{Pr}$ and the generated data distribution $\boldsymbol{Pg}$ is mathematically defined as the greatest lower bound (infimum) for any transport plan.

The Wasserstein distance loss:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

Where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ – denotes the set of all joint distributions $\gamma(x, y)$, whose marginals are respectively Pr and Pg.

However, the equation for the Wasserstein distance is highly intractable. Using the *Kantorovich-Rubinstein duality*, we can simplify the calculation to:

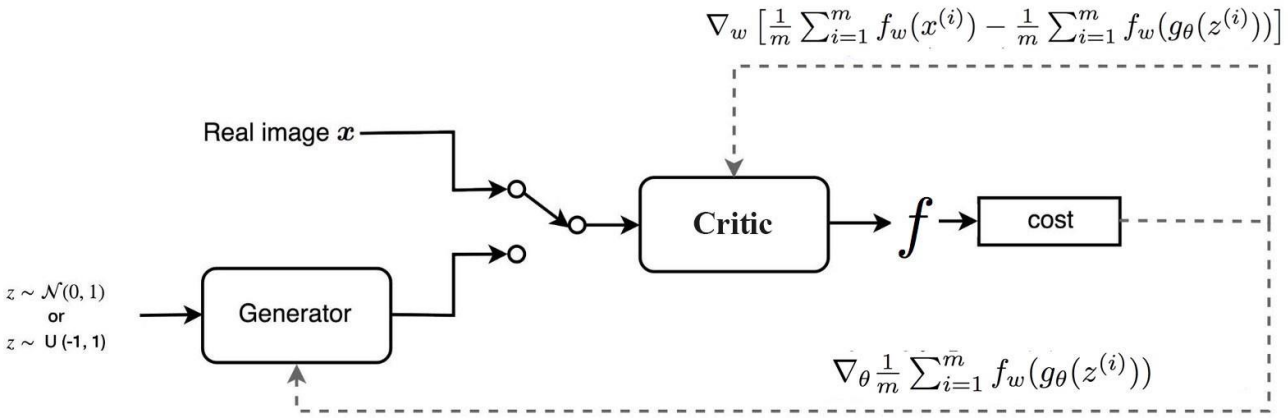
$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

Where sup is the least upper bound and f is a $1-Lipschitz\ function$ following this constraint:

$$|f(x_1) - f(x_2)| \leq 1 \cdot |x_1 - x_2|$$

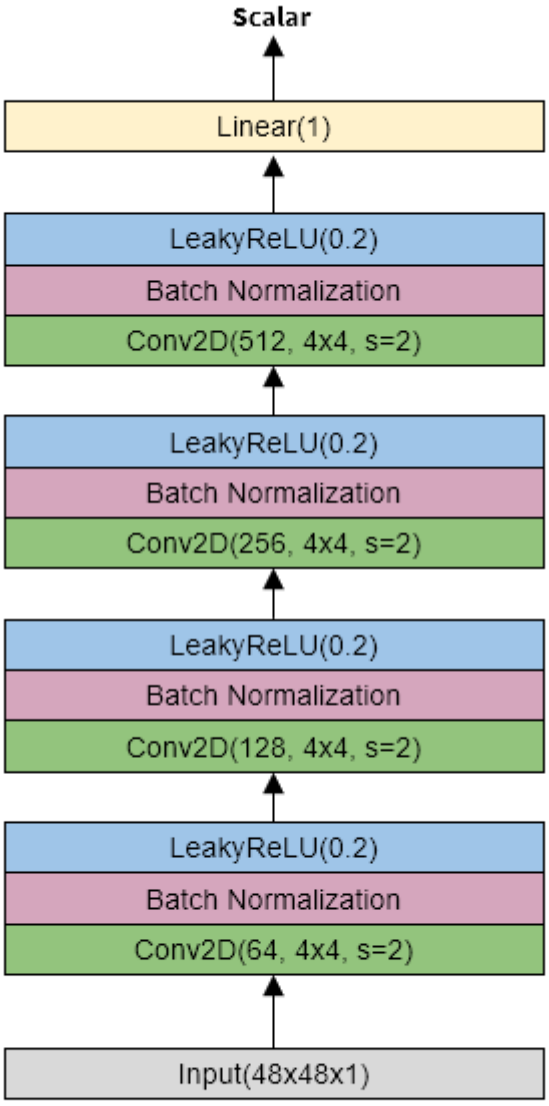
So, to calculate *the Wasserstein distance*, we just need to find a *1-Lipschitz function*.

We build a deep network to learn it. This network is very similar to the discriminator D, just without the sigmoid function and outputs a scalar score rather than a probability.



The Wasserstein GAN Architecture

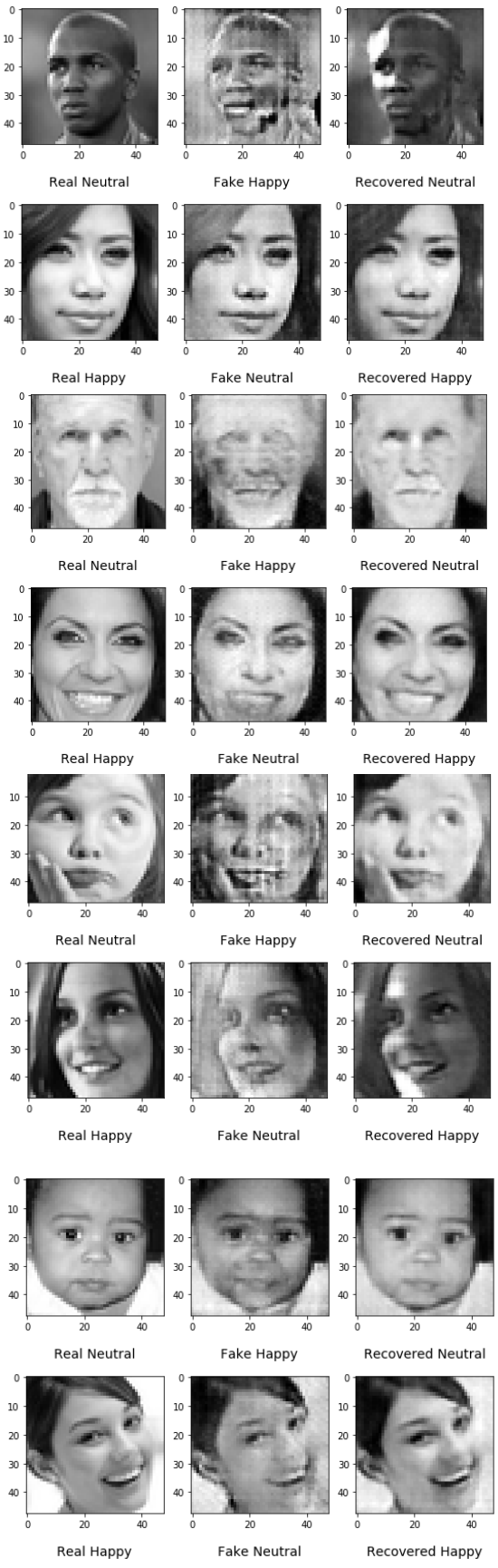
The main change in the Wasserstein GAN model is that discriminator is changed. We removed the Sigmoid activation function at the end. This discriminator called - Critic. The generator is the same as in CycleGAN.



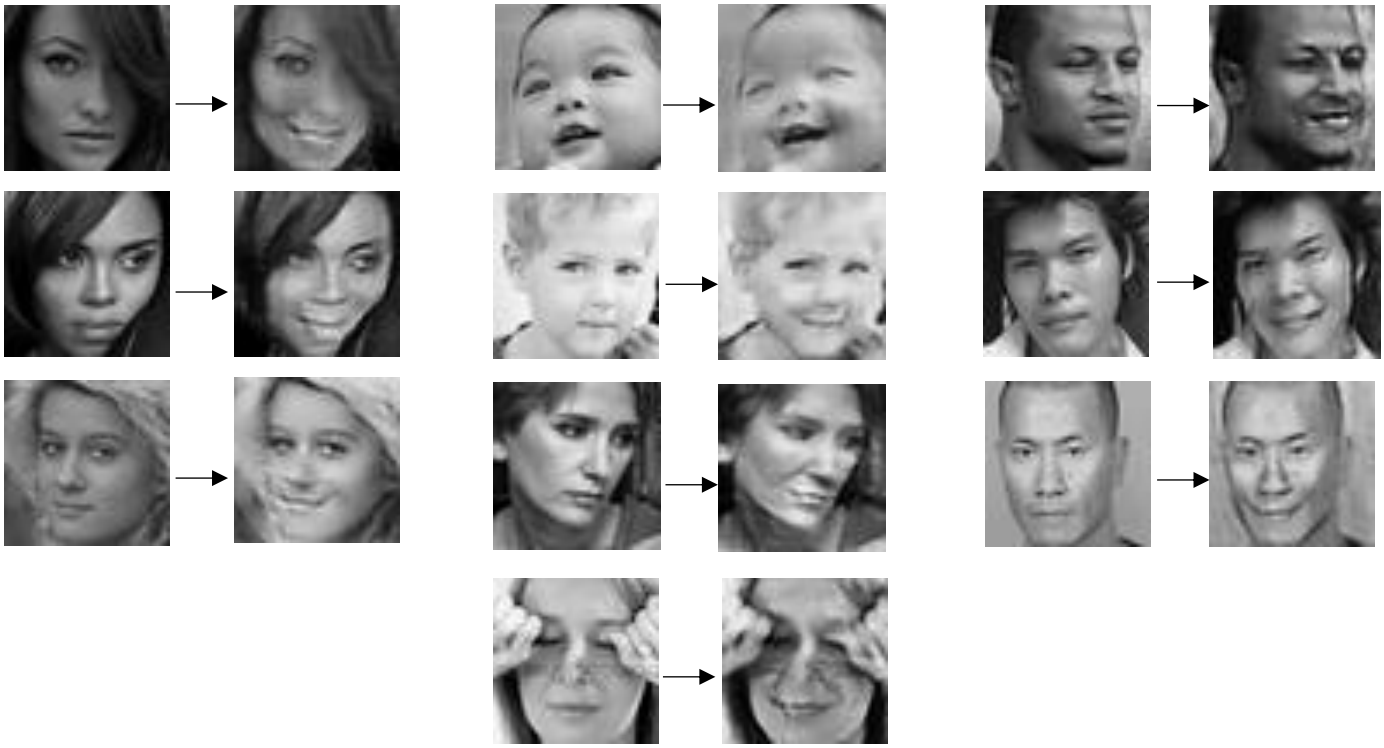
Critic Network

The Visual Results

Implemented paper results, CycleGAN




Fake Angry
Fake Happy
Fake Surprise
Fake Sad
Real Neutral



The Classifier

As we saw, we can see the transformation in the proper direction, but does it suits our goals? We need some numeric results on our work, to see does it improves the dataset, or not. For this purpose we will create a classifier, to see how data augmentation actually affects classification results. We added 500 images from Angry, Surprise, Sad, Happy and Disgust classes.

Results were checked on two different classifiers:

- Simple (~65%)
- Current State of the art (73%)

Simple classifier model architecture and results

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 46, 46, 64)	640
conv2d_2 (Conv2D)	(None, 46, 46, 64)	36928
batch_normalization_1 (Batch	(None, 46, 46, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 23, 23, 64)	0
dropout_1 (Dropout)	(None, 23, 23, 64)	0
conv2d_3 (Conv2D)	(None, 23, 23, 128)	73856
batch_normalization_2 (Batch	(None, 23, 23, 128)	512
conv2d_4 (Conv2D)	(None, 23, 23, 128)	147584
batch_normalization_3 (Batch	(None, 23, 23, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 11, 11, 128)	0
dropout_2 (Dropout)	(None, 11, 11, 128)	0
conv2d_5 (Conv2D)	(None, 11, 11, 256)	295168
batch_normalization_4 (Batch	(None, 11, 11, 256)	1024
conv2d_6 (Conv2D)	(None, 11, 11, 256)	590080
batch_normalization_5 (Batch	(None, 11, 11, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 5, 5, 256)	0
dropout_3 (Dropout)	(None, 5, 5, 256)	0
conv2d_7 (Conv2D)	(None, 5, 5, 512)	1180160
batch_normalization_6 (Batch	(None, 5, 5, 512)	2048
conv2d_8 (Conv2D)	(None, 5, 5, 512)	2359808
batch_normalization_7 (Batch	(None, 5, 5, 512)	2048
max_pooling2d_4 (MaxPooling2	(None, 2, 2, 512)	0
dropout_4 (Dropout)	(None, 2, 2, 512)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 512)	1049088
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dropout_6 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32896
dropout_7 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 7)	903

Total params: 5,905,863
Trainable params: 5,902,151
Non-trainable params: 3,712

The results:

Baseline:

```
Accuracy of the network on the 3589 test images: 65.09 %

Accuracy of Angry : 53 % of 262 / 491 total
Accuracy of Disgust : 60 % of 33 / 55 total
Accuracy of  Fear : 46 % of 244 / 528 total
Accuracy of Happy : 85 % of 750 / 879 total
Accuracy of   Sad : 44 % of 262 / 594 total
Accuracy of Surprise : 78 % of 327 / 416 total
Accuracy of Neutral : 73 % of 458 / 626 total
```

Baseline + Synthetic Data:

```
Accuracy of the network on the 3589 test images: 66.26 %

Accuracy of Angry : 57 % of 282 / 491 total    +4%
Accuracy of Disgust : 65 % of 36 / 55 total    +5%
Accuracy of  Fear : 51 % of 271 / 528 total    +5%
Accuracy of Happy : 87 % of 767 / 879 total    +2%
Accuracy of   Sad : 45 % of 271 / 594 total    +1%
Accuracy of Surprise : 78 % of 328 / 416 total ~0%
Accuracy of Neutral : 67 % of 423 / 626 total  -6%(*)
```

As you can see, data augmentation slightly improved classifier accuracy on classes that we added.

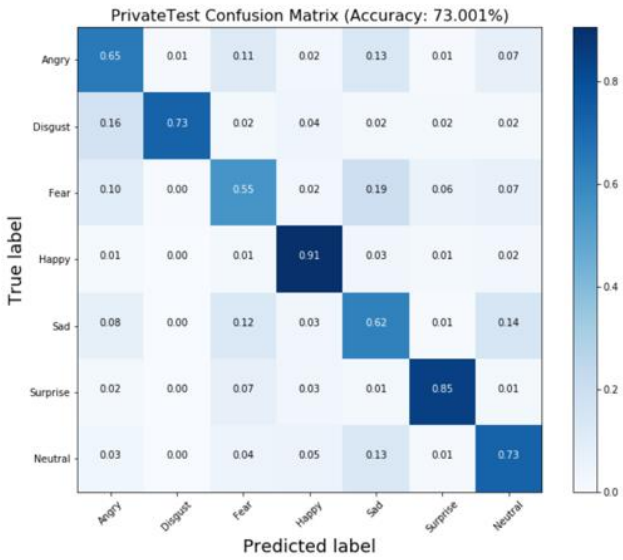
But, the neutral class, that are a reference class, has lower accuracy.

State of the Art classifier model architecture

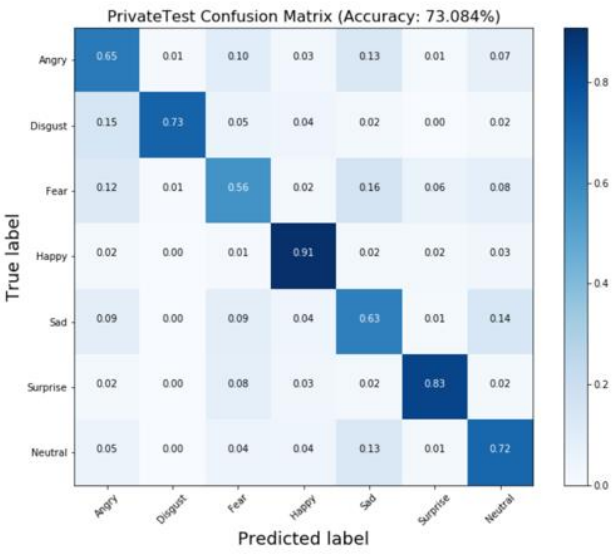
The state-of-the-art classifier is $VGG19$.

The results:

Baseline:



Baseline + Synthetic Data:



Sadly, this approach doesn't improve the state-of-the-art but also doesn`t decrease the baseline accuracy. So there a lot of future work that can improve the results.

Future Work and Summary

Conclusion:

This project has showed that we can get an astonishing result and create additional data. However, to be sure that we are creating suitable data, we need to ensure, that the distribution of generated data is the same as targeted one, so classifiers can learn more from it.

The very challenging task is to improve the state-of-the-art results by this approach, so more steps need to be done in order to get better results.

Future work:

- Further work with generated data:
    - Analyze distribution
    - Analyze similarity of generated and original images, by using ssim()
- Can we improve the state-of-the-art results?
- Generation of Neutral Class for FER2013, using Fake GAN
- Improvement Fake GAN by using WGAN-GP
- Put all together:
    - Use Fake Gan as part of Cycle GAN architecture
    - Analyze the difference between Cycle GAN, Improved Cycle Gan and Wasserstein GAN
- Testing performance on generated data while training on original and vice versa