Data Augmentation Using GANs

Project 236754, Dima Birenbaum

Supervisors: Yaron Honen, Gary Mataev
Main Goal

Our main goal is to generate faces with specific emotions. This generated data will serve as an external data source that helps improve a classifier.
Part A

Synthetic Data Generation by emotion transition using Generative Adversarial Networks
The Data

For middle stage, FER2013 dataset was chosen.

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:

<table>
<thead>
<tr>
<th>Emotion</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angry</td>
<td>4593</td>
</tr>
<tr>
<td>*Disgust</td>
<td>547</td>
</tr>
<tr>
<td>Fear</td>
<td>5121</td>
</tr>
<tr>
<td>Happy</td>
<td>8989</td>
</tr>
<tr>
<td>Sad</td>
<td>6077</td>
</tr>
<tr>
<td>Surprise</td>
<td>4002</td>
</tr>
<tr>
<td>Neutral</td>
<td>6198</td>
</tr>
</tbody>
</table>

* - we will discuss further
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 an input from one domain to another back and forth.
Theory Background

Domains A, B, mapping functions: $G:A \rightarrow B$, $F:B \rightarrow A$, 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: $a \rightarrow G(a) \rightarrow F(G(a)) \approx a$

(b) Backward cycle-consistency: $b \rightarrow F(b) \rightarrow G(F(b)) \approx b$
Target and loss functions

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

- Cycle consistency loss:
  \[
  \mathcal{L}_{cyc}(G, F) = \mathbb{E}_{a \sim p_{data}(a)}[\|F(G(a)) - a\|_1] + \mathbb{E}_{b \sim p_{data}(b)}[\|G(F(b)) - b\|_1]
  \]

- Full objective:
  \[
  \mathcal{L}(G, F, D_A, D_B) = \mathcal{L}_{GAN}(G, D_B, A, B) + \mathcal{L}_{GAN}(F, D_A, A, B) + \lambda \mathcal{L}_{cyc}(G, F)
  \]

- Target function:
  \[
  \hat{G}, \hat{F} = \arg\min_{G, F} \max_{D_A, D_B} \mathcal{L}(G, F, D_A, D_B)
  \]
The Model: *Forward Cycle A2B*

- **Start**
  - **Discriminator A**
    - Decision [0, 0.9]
  - **Input A**
  - **Generator A2B**
  - **Generated B**
  - **Generator B2A**
  - **Reconstructed A**
  - **Discriminator B**
    - Decision [0, 0.9]
The Model: *Backward Cycle B2A*

```
Start

Input B → Discriminator B → Decision [0, 0.9] → Generator B2A → Generated A → Discriminator A → Decision [0, 0.9] → Generator A2B → Reconstructed B
```
Both cycles together

- **Cycle A**
  - Real Image (A)
  - Fake Image (B)
  - Discriminator A
  - Generator A2B
  - Generator B2A
  - Reconstructed (A)
  - Reconstructed (B)

- **Cycle B**
  - Fake Image (A)
  - Real Image (B)
  - Discriminator B
  - Generator A2B
  - Generator B2A
  - Cycle B

Decision [0, 0.9]
The Architecture

Anatomy of Cycle GAN
Generator and Discriminator
The Networks:

**Generator**

The Generator consists of 3 parts:

- Decode (downsampling)
- Transferring (6 residual blocks)
- Encode (upsampling).
The Networks: **Discriminator**

The Discriminator - a simple CNN network, that determinates if the image is fake or real.
First results

- Neutral -> Happy transition
Problems

- “Dirty” dataset, unbalanced classes, mislabeled data
- Similarity between classes (for example: fear-angry, sad-neutral)
- Lack of data (Disgust Class – 550 images)
- Discriminator learns faster than the Generator.
- Vanishing gradient
- Quality and artifacts of output images
Solution – Weighted Cycle Loss

- 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 is 0.9 instead of 1
- Improving quality by changing cycle loss to:

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

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

- So final objective updated to:

\[
\mathcal{L}(G, F, D_A, D_B) = \mathcal{L}_{GAN}(G, D_B, A, B) + \mathcal{L}_{GAN}(F, D_A, A, B) + \lambda \mathcal{L}_{cyc}(G, F, D_A, A, \gamma) + \lambda \mathcal{L}_{cyc}(G, F, D_B, B, \gamma)
\]
Results of improved model
The Conclusion

So as we saw, we have a lot of problems, such as:

- model instability;
- vanishing gradient;
- dirty or small dataset;
- control over the training;
- battle between generator and discriminator etc.
So how can we improve 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 bottle-neck is to create data, that has same distribution as 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 $q$ to the data distribution $p$. The Wasserstein distance for the real data distribution $Pr$ and the generated data distribution $Pg$ is mathematically defined as the greatest lower bound (infimum) for any transport plan.
Theory Background - *The Wasserstein distance*

- 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 \(1 - \text{Lipschitz function}\) 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.

* - This score can be interpreted as how real the input images are.
The Networks: **Generator**

The Generator same as in Cycle GAN:

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

**Critic**

Same as Discriminator, but without Sigmoid activation at the end.

```
Conv2D(64, 4x4, s=2)  # Input(48x48x1)
Batch Normalization
LeakyReLU(0.2)
Conv2D(128, 4x4, s=2)
Batch Normalization
LeakyReLU(0.2)
Conv2D(256, 4x4, s=2)
Batch Normalization
LeakyReLU(0.2)
Conv2D(512, 4x4, s=2)
Batch Normalization
LeakyReLU(0.2)
Linear(1)
Scalar
```
Results of Wasserstein Cycle Gan
The Results

Let’s see visual results of the work.
Other results: Teenager
Other results: Women
Other results: Men
Other results: Old
Other results: Asian
Other results: Noisy
Other results: Noisy - Watermarks
Results on transformed data
Other results: Children #1
Other results: Children #2
Part B

- Classifiers on FER2013 Dataset
- Fake Neutral Images Generator Using DCGAN
The Classifier

Classification task on “dirt” dataset, maybe challenging.

Results were checked on two different classifiers:
- Simple (~65%)
- Current State of the art (73%)

FER2013
<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input image</td>
<td>48<em>48</em>1</td>
</tr>
<tr>
<td>Convolution&amp;ReLU</td>
<td>[3, 3, 1, 64] s=1</td>
</tr>
<tr>
<td>Max-Pooling&amp;Norm</td>
<td>[1, 3, 3, 1] s=2</td>
</tr>
<tr>
<td>Convolution&amp;ReLU</td>
<td>[3, 3, 64, 128] s=1</td>
</tr>
<tr>
<td>Max-Pooling&amp;Norm</td>
<td>[1, 3, 3, 1] s=2</td>
</tr>
<tr>
<td>FC*2</td>
<td>256</td>
</tr>
<tr>
<td>Softmax</td>
<td>[256, 7]</td>
</tr>
<tr>
<td>Output logits</td>
<td>[7]</td>
</tr>
</tbody>
</table>
Surreal (Paper) Classifier - Architecture

Doesn't work

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input image</td>
<td>48<em>48</em>1</td>
</tr>
<tr>
<td>Convolution&amp;ReLU</td>
<td>[3, 3, 1, 64] s=1</td>
</tr>
<tr>
<td>Max-Pooling&amp;Norm</td>
<td>[1, 3, 3, 1] s=2</td>
</tr>
<tr>
<td>Convolution&amp;ReLU</td>
<td>[3, 3, 64, 128] s=1</td>
</tr>
<tr>
<td>Max-Pooling&amp;Norm</td>
<td>[1, 3, 3, 1] s=2</td>
</tr>
<tr>
<td>FC*2</td>
<td>256</td>
</tr>
<tr>
<td>Softmax</td>
<td>[256, 7]</td>
</tr>
<tr>
<td>Output logits</td>
<td>[7]</td>
</tr>
</tbody>
</table>
## Simple Classifier - Architecture

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 46, 46, 64)</td>
<td>640</td>
</tr>
<tr>
<td>conv2d_2 (Conv2D)</td>
<td>(None, 46, 46, 64)</td>
<td>36928</td>
</tr>
<tr>
<td>batch_normalization_1 (Batch)</td>
<td>(None, 46, 46, 64)</td>
<td>256</td>
</tr>
<tr>
<td>max_pooling2d_1 (MaxPooling2D)</td>
<td>(None, 23, 23, 64)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_1 (Dropout)</td>
<td>(None, 23, 23, 64)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_3 (Conv2D)</td>
<td>(None, 23, 23, 128)</td>
<td>73856</td>
</tr>
<tr>
<td>batch_normalization_2 (Batch)</td>
<td>(None, 23, 23, 128)</td>
<td>512</td>
</tr>
<tr>
<td>conv2d_4 (Conv2D)</td>
<td>(None, 23, 23, 128)</td>
<td>147584</td>
</tr>
<tr>
<td>batch_normalization_3 (Batch)</td>
<td>(None, 23, 23, 128)</td>
<td>512</td>
</tr>
<tr>
<td>max_pooling2d_2 (MaxPooling2D)</td>
<td>(None, 11, 11, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_2 (Dropout)</td>
<td>(None, 11, 11, 128)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_5 (Conv2D)</td>
<td>(None, 11, 11, 256)</td>
<td>295168</td>
</tr>
<tr>
<td>batch_normalization_4 (Batch)</td>
<td>(None, 11, 11, 256)</td>
<td>1024</td>
</tr>
<tr>
<td>conv2d_6 (Conv2D)</td>
<td>(None, 11, 11, 256)</td>
<td>590880</td>
</tr>
<tr>
<td>batch_normalization_5 (Batch)</td>
<td>(None, 11, 11, 256)</td>
<td>1024</td>
</tr>
<tr>
<td>max_pooling2d_3 (MaxPooling2D)</td>
<td>(None, 5, 5, 256)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_3 (Dropout)</td>
<td>(None, 5, 5, 256)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_7 (Conv2D)</td>
<td>(None, 5, 5, 512)</td>
<td>1180160</td>
</tr>
<tr>
<td>batch_normalization_6 (Batch)</td>
<td>(None, 5, 5, 512)</td>
<td>2048</td>
</tr>
<tr>
<td>conv2d_8 (Conv2D)</td>
<td>(None, 5, 5, 512)</td>
<td>2359808</td>
</tr>
<tr>
<td>batch_normalization_7 (Batch)</td>
<td>(None, 5, 5, 512)</td>
<td>2048</td>
</tr>
<tr>
<td>max_pooling2d_4 (MaxPooling2D)</td>
<td>(None, 2, 2, 512)</td>
<td>0</td>
</tr>
<tr>
<td>dropout_4 (Dropout)</td>
<td>(None, 2, 2, 512)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_1 (Flatten)</td>
<td>(None, 2048)</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 512)</td>
<td>1049088</td>
</tr>
<tr>
<td>dropout_5 (Dropout)</td>
<td>(None, 512)</td>
<td>0</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 256)</td>
<td>131328</td>
</tr>
<tr>
<td>dropout_6 (Dropout)</td>
<td>(None, 256)</td>
<td>0</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>(None, 128)</td>
<td>32896</td>
</tr>
<tr>
<td>dropout_7 (Dropout)</td>
<td>(None, 128)</td>
<td>0</td>
</tr>
<tr>
<td>dense_4 (Dense)</td>
<td>(None, 7)</td>
<td>903</td>
</tr>
</tbody>
</table>

Total params: 5,985,863  
Trainable params: 5,982,151  
Non-trainable params: 3,712
Simple Classifier - 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, we diminish Neutral Class, so what can we do? Generate Neutral Class, more in future works section.
SOTA Classifier (VGG19) - Results

Can we achieve state of the art?!
The Fake GAN

So how can we supply more data, with the same distribution? The answer is, we will create it.
Fake GAN using DCGAN, WGAN-GP
Generator - Architecture
The generator is designed to map the latent space vector \( Z \) to data-space. Since data are images, converting \( Z \) to data-space means ultimately creating an image with the same size as the training images (i.e. 1x48x48). In practice, this is accomplished through a series of strided two dimensional convolutional transpose layers, each paired with a 2d batch norm layer and a ReLU activation.
Discriminator - Architecture

Discriminator - is a binary classification network that takes an image as input and outputs a scalar probability that the input image is real (as opposed to fake).

Discriminator takes a 1x48x48 input image, processes it through a series of Conv2d, BatchNorm2d, and LeakyReLU layers, and outputs the final probability through a Sigmoid activation function.
Overview

Real Faces

Random Noise $z \sim \mathcal{N}(0,1)$

Generator

Generated Faces

Discriminator

Fake

Real

$\mathbb{G}$eneraten $\mathcal{F}$aces

$\mathbb{R}$andom $\mathcal{N}$oise $\mathcal{N}(0,1)$
Overview

Generate Image From Noise → Find Faces → OpenCV Face Detector → Filter Neutral Faces

Neutral Faces → Sad → Surprised → Happy → Angry → Fear
Future work

- Further work with generated data:
  - Analyze distribution
  - Analyze similarity of generated and original images, by using $ssim()$
- Can we improve 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 difference between Cycle GAN, Improved Cycle Gan and Wasserstein GAN
- Testing performance on generated data while training on original and vice versa
The END