CheerMeUp - using Hidden treasures

[redacted] **Divyanshu Mund**
A53281339
dmund@eng.ucsd.edu [redacted]

[redacted]

Abstract

Through this work, we want to model a generative network that can generate an image given based on given inputs. This controlled image synthesis is achieved using latent space vectors representing facial features. Our network is essentially able to interpolate between different facial features hairstyle, age, hair colour etc by making the latent vectors transparent. Hence the name Transparent Latent GAN (TL-GAN). We trained our GAN on FERG256 dataset [1] and were able to interpolate between various emotions of a given image. Given a GAN synthesized image from random noise vector, we essentially found disentangled representative axis along which we vary the noise vector to continuously interpolate between different facial emotions and faces in the generated images. We attribute the main idea to this blog on TL-GAN [3]. Our repository is available at Github repo

1 Introduction

In order to understand how neural networks work in various recognition tasks, researchers are coming up with various architectures to piece out the working of DNN. One such task Generative Adversarial Networks (GAN), is to understand the distribution from which input data is produced. From the initial paper by Ian GoodFellow. GANs have come along way since then. Today,GANS are one of the most popular algorithms in deep learning based on learning, having a wide variety of applications in fields ranging from automated vehicles to medical imaging. Basic idea of GANs is to have two neural networks competing with one another, (i.e "adversarial") in such a way that image produced by one network (generator) tries to fool the other network (discriminator) discriminating between real and synthetic images. This is turn generates new synthetic instances of data which are same/close to real data as if it is sampled from input data distribution. Since their inception in 2014, there have been a lot of improvements and modifications in their architectures.

2 Motivation

Usual GAN approaches do not have any control over synthesized images w.r.t latent input noise. Our approach wants to find a disentangled axes over which if we vary our latent code, we can control the image synthesized. We will train a GAN which learns to produce samples from real data distribution. So our generator model G produces a mapping from z to synthetic images x_{gen}. We want to find a mapping from z to image labels y_{gen}, where y_{gen} represent emotion in the produced synthetic image. Since we don't have the direct mapping from supervised data, we will learn the mapping. We will first learn a classifier F which learns the mapping from x_{real} to y_{real}. Now given

an input noise z, we will produce a synthetic image x_{gen} and use our classifier F to get y_{gen}. Using this method we will get a mapping from z to y_{gen}. We will use a Generalized Linear Model (GLM) to perform regression between latent vectors z and features y_{gen}. The regression slope becomes the feature axes. The problem with this method is that the axes do not correspond to a disentangled representation towards a particular feature. To remedy this, we will use simple linear algebra trick of orthogonal projection to find axes which are linearly independent, so that changing input noise z along a particular axes will only affect a particular a feature.

3 Related work

In deep learning there has been rapid progress on generative models. Three key players are: autoregressive models, variational autoencoders (VAE) and generative adversarial networks (GAN),[2]

We are dealing with GANs in this work. In earlier versions of GAN and many popular models like DC-GAN and pg-GAN can be used as unsupervised learning models. After training, the generator network takes random noise as input and produces a realistic image that is hardly distinguishable from the training data. But, we do not have any control over the features of the generated images. There are many applications in which we need to control these features. Like to generate samples with custom features like age, hair color, facial expression, etc and tuning each feature continuously. There is work in two directions here: style-transfer networks and conditional generators.

First, Style-transfer networks, such as CycleGAN [7] and pix2pix [4], are used to model images from one domain to another. But we cannot continuously tune one feature gradually between two discrete states. Also, one network is trained for one type of transfer, so it requires ten different neural networks to tune 10 features.

Second, Conditional generators, such as conditional GAN, AC-GAN, and Stack-GAN, jointly learn images with feature labels during training time, enabling the image generation to be conditioned on custom features. But due to this, when we want to add new features to the generation process, we have to retrain the whole model which is an expensive task. Also, we have to rely on a single dataset that contains all the custom feature labels to perform the training, and hence we can't leverage different labels from multiple datasets.

In this task we are using TL GAN architecture along with DC GAN to create faces with different emotions conditionlly based on a given input image. Transparent Latent-space GAN (TL-GAN), addresses the problems we discussed in last two paragraphs above. With TL GAN we can gradually tune one or multiple features using a single network. Also, we can add new tunable features very efficiently and quickly.

4 Method: TL GAN

Motivated from TL GAN idea we worked with components as discussed below and shown in Figure1. For our models, we experimented and tried different architectures for the components to achieve good results which we discuss in later sections.

4.1 Components

Generator
We used DC GAN as our generator architecture. DC GAN is one of the popular and successful network design for GAN, used to generate synthetic images. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling. We use a DC-GAN [5] architecture with five conv layers along with batch-norm and relu layers as described below and shown in Figure2:

conv →batchnorm →relu →conv →batchnorm →relu →conv →batchnorm →relu →conv →batchnorm →relu →conv →tanh

Discriminator
Discriminator part is our classifier model to distinguish between real and synthetic images. We are using a CNN architecture for this -

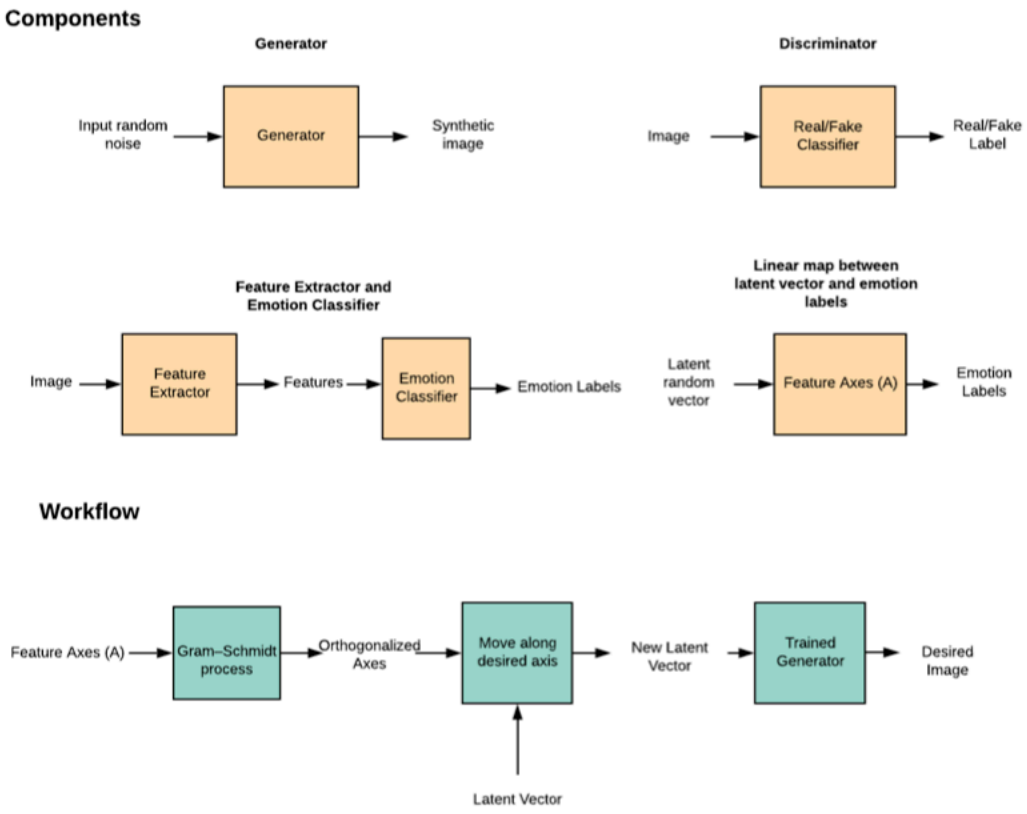


Figure 1: Work flow

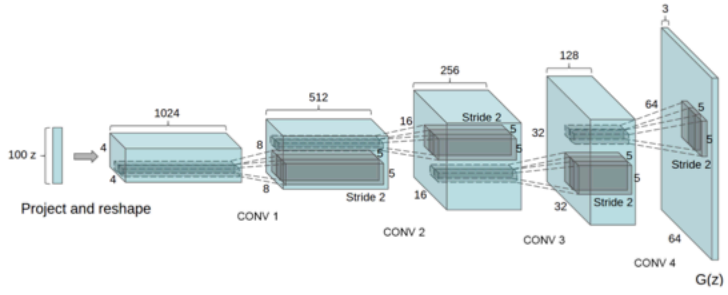


Figure 2: DC-GAN generator

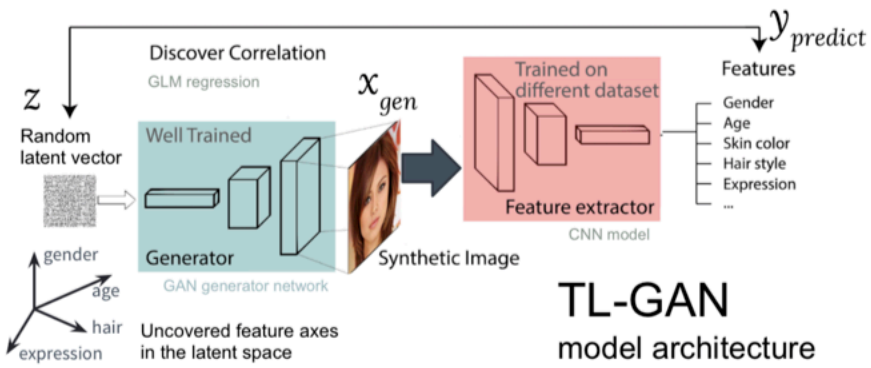


Figure 3: TL-GAN architecture

Conv →LeakyRelu →Conv →BatchNorm →LeakyReLu →Conv →BatchNorm →LeakyReLu →Conv →BatchNorm →LeakyReLu →Conv

Loss
Initially we started with BCE loss to train our network. But from literature survey, we noticed that MSE loss gave better results (LS GAN)[6]. Hence, we incorporated MSE loss.

The objective for our generator-discriminator architecture is as follows:
The discriminator seeks to minimize the sum of squared difference between predicted and expected values for real and fake images.

$$\text{Discriminator loss} : (D(x)\text{–}1)^2 + (D(G(z)))^2$$

The generator seeks to minimize the sum squared difference between predicted and expected values as though the generated images were real.

$$\text{Generator loss} : (D(G(z))\text{–}1)^2$$

We maintained the class labels of 0 and 1 for fake and real images respectively, and minimized the least square loss (MSELoss from Pytorch)

$$\text{L2 loss} = \sum (y_{predicted} - y_{true})^2$$

Feature Extractor
We used pretrained alexnet and finetuned it to extract features from the real images that are then fed to our emotion-classifier(we'll talk about classifier in next few paragraphs) to get the emotion-labels. However since we only have labels for real dataset, we'll train our emotion-classifier on that.

Emotion Classifier
We use a simple architecture for our classifier as mentioned below. This simple enough architecture is able to give us 100% accuracy on our test-set for feature to emotions mapping. This is due to the nature of our dataset.

features →fc layer →ReLu →batchnorm →softmax

Linear map between latent vector and emotion-labels
Since we want to control our emotion-labels by varying our input latent vectors, we need to learn a mapping between them. Therefore, we learn latent-vector to emotion-label mapping. We achieve this by using a linear model. By linear regression, we are able to get a matrix of size *dim(latent-vector) X dim(emotion-labels)*, where each column would correspond to axis for an emotion.

The reason for using a linear model to learn latent-vector to emotion-label mapping is motivated from an experiment that we run to understand our label space. When we varied our input latent-vectors we observed that there is a slight yet continuos shift in emotions. Inspired from this observation we obtain our feature axes by linear regression.

However, in order to have control on a particular emotion-label, we want to change our features corresponding to that particular emotion keeping everything else the same. Thus, we want our feature axes to be orthogonal. We used gram-schmidt method to orthogonalize our extracted features. Now changing these features will change only a particular emotion.

4.2 Work Flow

In order to obtain a generated image with desired emotion, we feed our learnt feature-axes to component which first orthogonalize the axes using gram-schmidt process as described in the above paragraph. Then we move our latent vector in the desired feature direction and obtain a new latent vector. This new vector when fed to the already trained generator, gives our the desired image.

All the components discussed above along with the work flow consitutes our TL GAN architecture, which is shown in Figure 3.

To summarize we follow these nine steps-
Step I : Train a GAN on the dataset
Step II : Train a emotion classifier
Step III: Using the GAN and the classifier, generate (Z,Y) samples
Step IV : Linear Regression on (Z,Y). The coefficients are the feature axis for that emotion
Step V : Disentangle the feature axes(Gram Schmidt)
Step VI : Sample a Z and generate the image
Step VII: Choose an emotion that we want to impart
Step VIII: Select the disentangled corresponding feature axes F
Step IX : Vary Z as Z $\rightarrow$Z + kF

5 Results

5.1 Data

For this task we are using FERG-DB dataset found on web. FERG-DB consists of around 60,000 images of sizes:256x256 labeled with different facial emotions. This dataset consists of 6 different identities. Pertaining to our task's focus, we choose to work with smaller sized images here. Therefore, we resize our images to 64x64 pixel.
The emotion-labels are as follows:
Emotions: ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surprise']

5.2 Training

We first train our generator-discrimator by feeding in generated random latent vector of dim = 100 to the generator. We see that our generator model improves its generated images with training time. We obtain good generated images in around 250 epochs. The loss plots for our generator is show in Figure4.

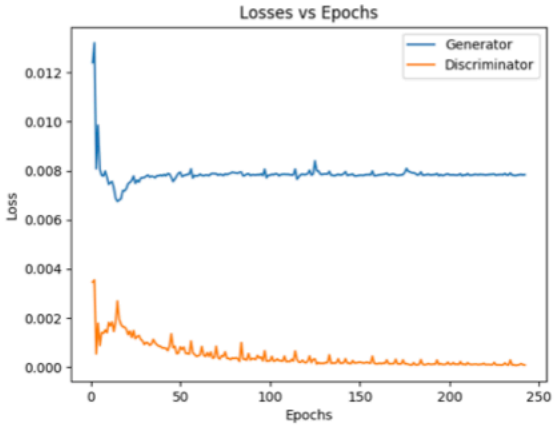


Figure 4: Losses plot for training DC-GAN

We then learn labels of the generated images using the classifier. Then we learn slope vectors by fitting a linear regression model on random latent vectors and our labels for generated images. This slope vectors are then orthogonalised using Gram-Schmidt process so that we vary only a specific label while also ensuring that we are not affecting the others. We run a sanity check after orthogonalizing the feature vectors. We then study the correlation for these orthogonalised vectors. The heat map for the correlation matrix obtained is show in Figure5. The heat map is obtained as expected, with a vector's correlation of one with itself and zero with all others. This serves as a sanity check for orthogonalization step.

Initially we follow the above process using the complete FERG-DB dataset. However, we find that on varying the features learnt, we do not always get smooth and good transitions. For eg, when we try to change emotion 'joy', we see that the character is also transformed in some cases. We faced this issue as we had a very spiky distribution ie. consisting various modes in our real dataset. To alleviate this issue, we used dataset pertaining to a single character and hence used a single modal dataset. We expect and want our label space to be smooth, so that we can control variations in a continuous fashion by varying our latent-vectors. Therefore, we only choose to work with emotions(as label space) because our dataset is rich enough with variations in emotions.

5.3 MultiModal distribution

We randomly choose some generated(using all characters) images shown in a grid of 8x16 cells in the **Figure 7**. We also see a good variation in our generated images. Looking at the images data, we

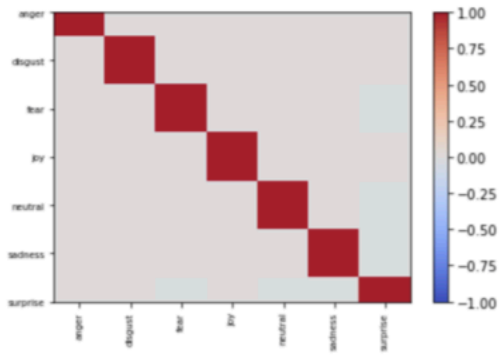


Figure 5: Correlation heat map for orthogonalized feature vectors

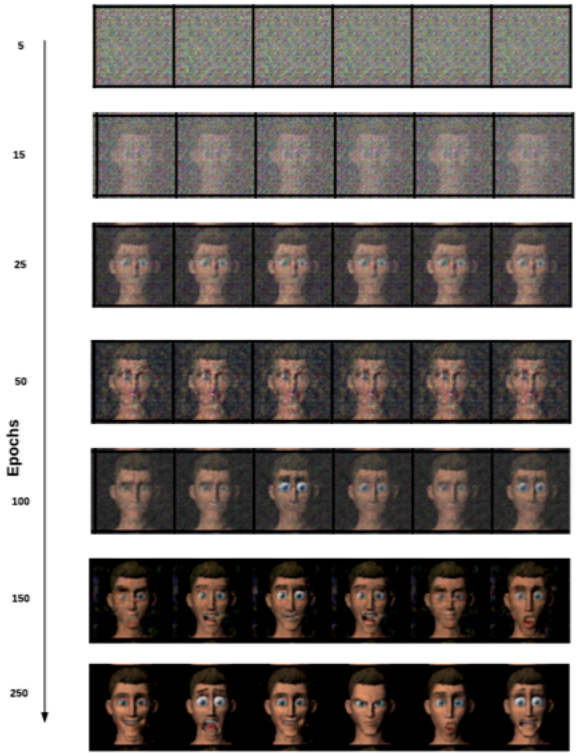


Figure 6: Images generated with training time(/epochs)

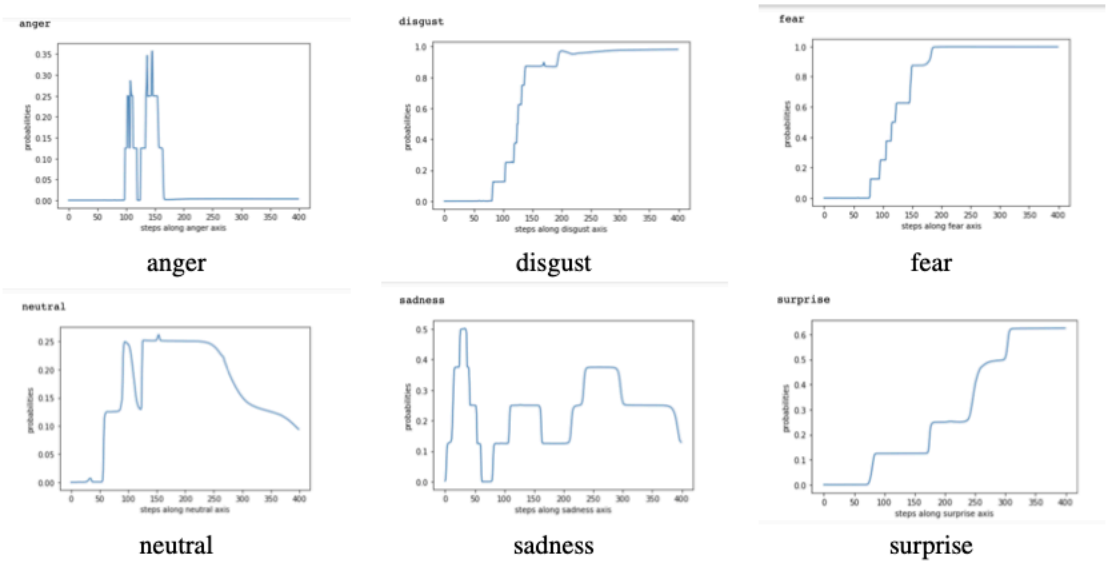
Figure 7: Generated images

see that we have emotions for 6 characters.
Once we obtain axes, we translate along an emotion-axis. To show some image-examples we include Figure 8 & 9 which show generated images as we move along 'surprise' axis.
We can see that we get our desired emotion change in 2nd and 2nd last image. However, some images (example 3rd and 4th) completely change as we translate the noise vector. This is because our dataset itself is not continuous enough and GAN is not able to interpolate facial features smoothly. We think this is because the data distribution is intrinsically multimodal, ie, it contains 6 'modes' which are densely populated by really far apart from one another.

Figure 8: Initial images

Figure 9: Images after translation along 'surprise' axis

Figure 10: GAN results with training on just one character



anger disgust fear

neutral sadness surprise

Figure 11: Plots of classification probabilities varying across the emotion axis

5.4 UniModal distribution

To smoothly translate among emotions, we decided to first tackle the problem only for a particular face. We retrained our GAN on all images of a particular character. We see that with training time, we obtain better and better generated images as shows in Figure6. After training, the generated images are shown in Figure10.
After training GAN for just one character(Jules) and using classifier which was previously trained on the complete FERG-DB dataset, the next step was to create data samples for learning $z = f(Y)$. In our case, this translates to (z, Y) samples where z is the random noise(input to the GAN) and Y is the softmax probability vector corresponding to the image generated by GAN. With these as our data points, we train a linear regressor which we claim to be having vector coefficients(corresponding to each of the emotions) such that they represent the feature axis. This means moving z along that axis would mainly change the corresponding emotion.

5.5 Variation along feature-axis

We translate noise vector along various emotion-axes. As we vary along the feature axes we observe that the confidence of our emotion-classifier in that particular emotion increases and it's probability reaches to 1. The variation in probability of emotion for a generated image(as obtained from our trained classifier) is show in plots in Figure11. For better visualization we also include a link to a gif that basically shows snaps of the generated images as we move along the joy axis. YouTube Link

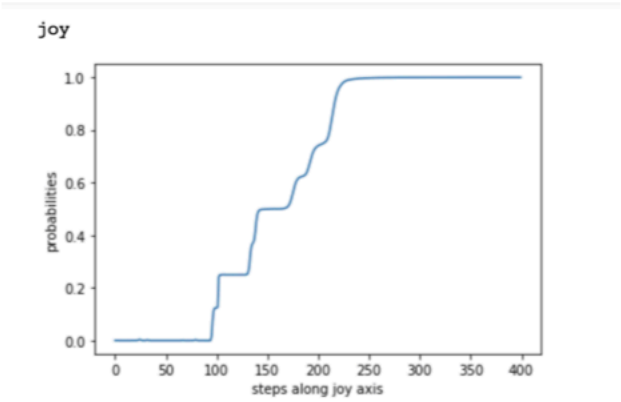


Figure 12: Plot showing probability of joy increases as we move z along the emotion axis

Figure 13: State 1 : Randomly generated images

Figure 14: State 2 : Intermediate state images

Figure 15: State 3 : Final state images

6 Discussion

Experimenting with the latent noise vector, we found that most points in the latent space generate decent images. However, since there are only 6 faces in the dataset, linear interpolation in multimodel setup is not smooth. For example, small change in latent noise for 'happy' face will change the emotions but a large change will change emotion on an entirely different face. This is due to spiky distribution of our dataset which has various modes correponding to the characters. But training GAN for just one character(Jules) we get the results which were desired and expected. Moving z along that emotion axis changes the corresponding emotion keeping the underlying character constant. We get a constant increase in probabilities for emotions like joy, surprise, fear, neutral and disgust as we move along the emotion axis.

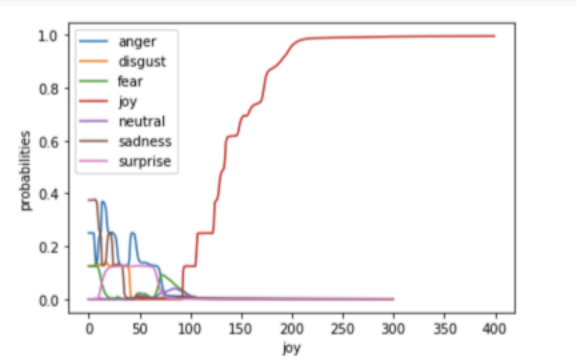


Figure 16: All emotion confidence level while varying joy axis

6.1 Explaining the interpolation with plots

From the plots in Figure11, it is clear that the probabilities increase moving along the corresponding feature axis except for all emotions except anger and surprise emotions. We suspect this is due to the underlying correlation between some of these emotions which makes it harder to orthogonalize these emotions along a particular axis.

In Figures 13, 14, 15 we also show the transition for 8 randomly generated images as we move z along the feature axis corresponding to joy emotion. Three different states are shown for each of them and we can see that the results are consistent with what we expect or wanted to achieve.

From linear mapping between latent vector and orhogonalized features, We obtain the disentangled feature representation. Linear map's coefficient become the axis along which we can mode latent random vector to interpolate between emotions. We considered our target emotion to be "joy" and found the corresponding feature axis.increasing the value of latent vector along this axis, we are able to see the generated image becoming visually more joyful. We also wanted to see the probabilities of different expression labels during this interpolation (figure 16). We can see that as the value along the joy's feature axis increased, we are seeing probability of image's expression being "joy" increases. This reinforced our idea that latent vector space is smooth in terms of expressions and can be interpolated smoothly between various expressions.

7 Conclusion

We observed that we are able to continuously vary features and observed good transitions in emotions on the generated faces. We are able to exploit TL GAN architecture to achieve our expected results. We observed that for multimodel, since distribution is spiky we don't observe smooth transitions. We do observe expected results in unimodal setup. From our work, we conclude that we are able to smooth move in label space and can control this motion by varying our input latent vectors. For future steps, our work can be extended to achieve the same for real image(instead of generated image). It will be interesting to see control over discrete features as well.

8 Author's contributions

Everyone contributed equally towards coding, discussion, debugging, training and monitoring models.

[redacted]

Dataloader, Dataset analysis, Report, Regression, Latent Space Analysis

8.2 Divyanshu Mund

Literature Survey, DC-GAN architecture, Latent space analysis

[redacted]

Data preprocessing, GAN training, Classifer Train

[redacted]

Helper functions, logging, plotting, visualization of interpolated images

[redacted]

Report, ablation studies, Normalization and orthogonal feature axes,

References

[1] Deepali Aneja, Alex Colburn, Gary Faigin, Linda Shapiro, and Barbara Mones. Modeling stylized character expressions via deep learning. In *Asian Conference on Computer Vision*, pages 136–153. Springer, 2016.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[3] Shaobo GUAN. Generating custom photo-realistic faces using ai. Medium Blog post.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[6] Haoran Xie Raymond Y.K. Lau Zhen Wang Stephen Paul Smolley Xudong Mao, Qing Li. Least squares generative adversarial networks, 2016.

[7] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.