Few-shot Knowledge Transfer for Fine-grained Cartoon Face Generation

Group : R53

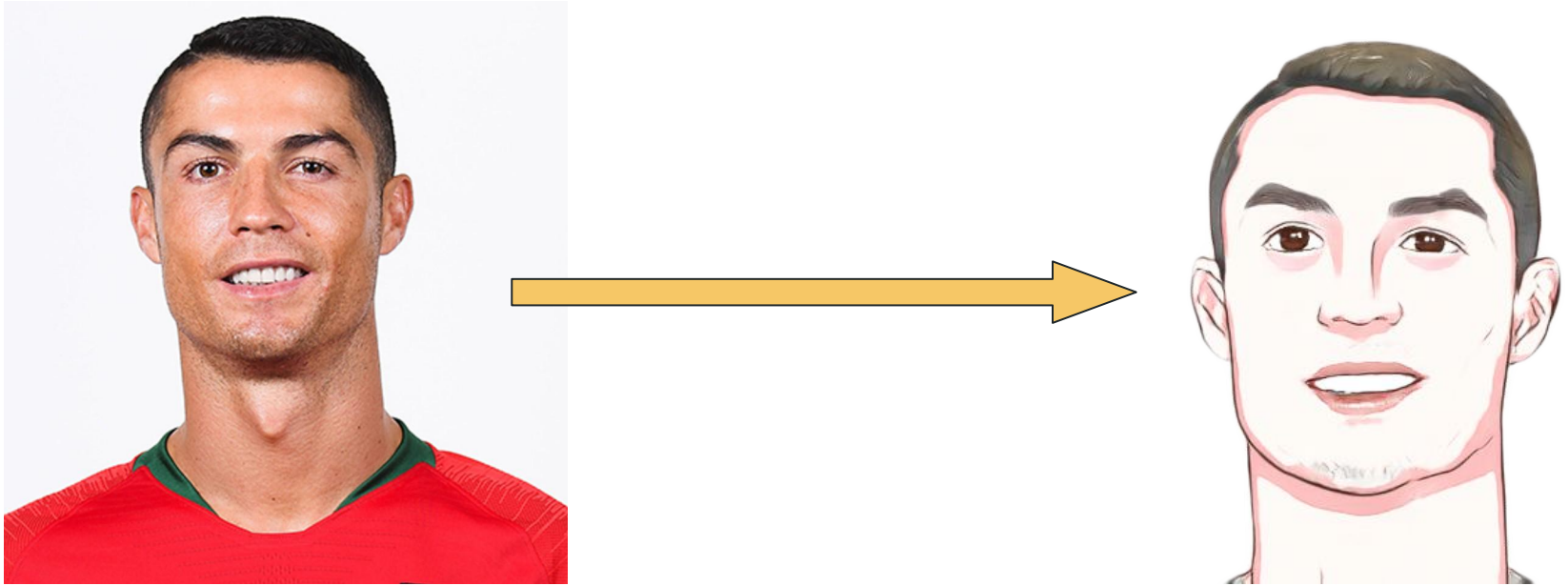
Presented by : Dhruv Deshmukh (11940380)
Anupam Kumar (11940160)

Introduction

- Cartoon faces can be seen everywhere in our daily life.
- Widely used on social media platforms as profile pictures or even in stickers used in messaging apps.
- But drawing a cartoon face is difficult and time consuming, may take several hours even for a professional artist to make one.
- Our project tries to design a GAN type network for converting real images to fine grained cartoon images.

Objective

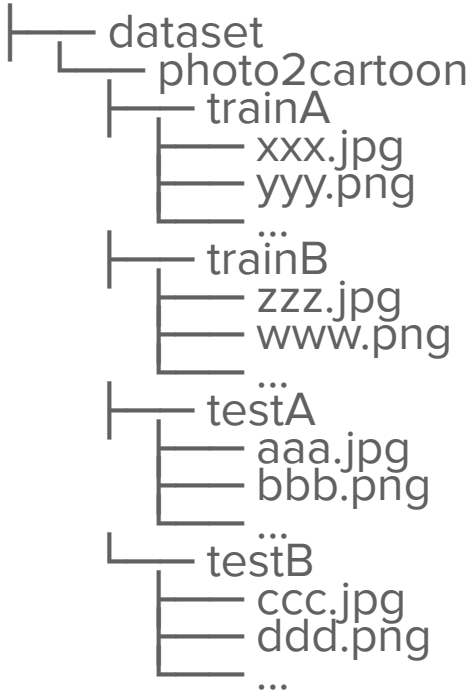
- The prime objective is to generate **cartoon face image** (.jpg/.png file) corresponding to a **real image** (.jpg/.png file) which must contain face of a person. An example is shown below:



Dataset Used

- 204 cartoon images provided in the code repo itself and images provided in the paper.

- Subset of images from dataset at https://www.kaggle.com/greatgamedota/ffhq-face-data-set.



Preprocessing Steps

1. Detect the face and key features.
2. Rotate the face to correct it according to the key.
3. Expand the key bounding box at a fixed scale and crop out the face area.
4. Whiten the background using the portrait split model.

1 2 3 4

Methodology

Models Available: **CycleGAN, UNIT** for unpaired image translation

Model Used: The above modartiels have tendency to be unstable hence a model called **U-GAT-IT** was used which uses an adaptive Normalization Function(**AdaLIN**) and an attention module(**CAM**) to give better results.

Reason for choosing U-GAT-IT(lower KID values)

Model	selfie2anime	horse2zebra	cat2dog	photo2portrait	photo2vangogh
U-GAT-IT	**11.61 ± 0.57**	**7.06 ± 0.8**	**7.07 ± 0.65**	1.79 ± 0.34	4.28 ± 0.33
CycleGAN	13.08 ± 0.49	8.05 ± 0.72	8.92 ± 0.69	1.84 ± 0.34	5.46 ± 0.33
UNIT	14.71 ± 0.59	10.44 ± 0.67	8.15 ± 0.48	**1.20 ± 0.31**	**4.26 ± 0.29**
MUNIT	13.85 ± 0.41	11.41 ± 0.83	10.13 ± 0.27	4.75 ± 0.52	13.08 ± 0.34
DRIT	15.08 ± 0.62	9.79 ± 0.62	10.92 ± 0.33	5.85 ± 0.54	12.65 ± 0.35
AGGAN	14.63 ± 0.55	7.58 ± 0.71	9.84 ± 0.79	2.33 ± 0.36	6.95 ± 0.33
CartoonGAN	15.85 ± 0.69	-	-	-	-
Model	anime2selfie	zebra2horse	dog2cat	portrait2photo	vangogh2photo
U-GAT-IT	**11.52 ± 0.57**	**7.47 ± 0.71**	**8.15 ± 0.66**	1.69 ± 0.53	5.61 ± 0.32
CycleGAN	11.84 ± 0.74	8.0 ± 0.66	9.94 ± 0.36	1.82 ± 0.36	**4.68 ± 0.36**
UNIT	26.32 ± 0.92	14.93 ± 0.75	9.81 ± 0.34	**1.42 ± 0.24**	9.72 ± 0.33
MUNIT	13.94 ± 0.72	16.47 ± 1.04	10.39 ± 0.25	3.30 ± 0.47	9.53 ± 0.35
DRIT	14.85 ± 0.60	10.98 ± 0.55	10.86 ± 0.24	4.76 ± 0.72	7.72 ± 0.34
AGGAN	12.72 ± 1.03	8.80 ± 0.66	9.45 ± 0.64	2.19 ± 0.40	5.85 ± 0.31

U-GAT-IT-Model Architecture Overview

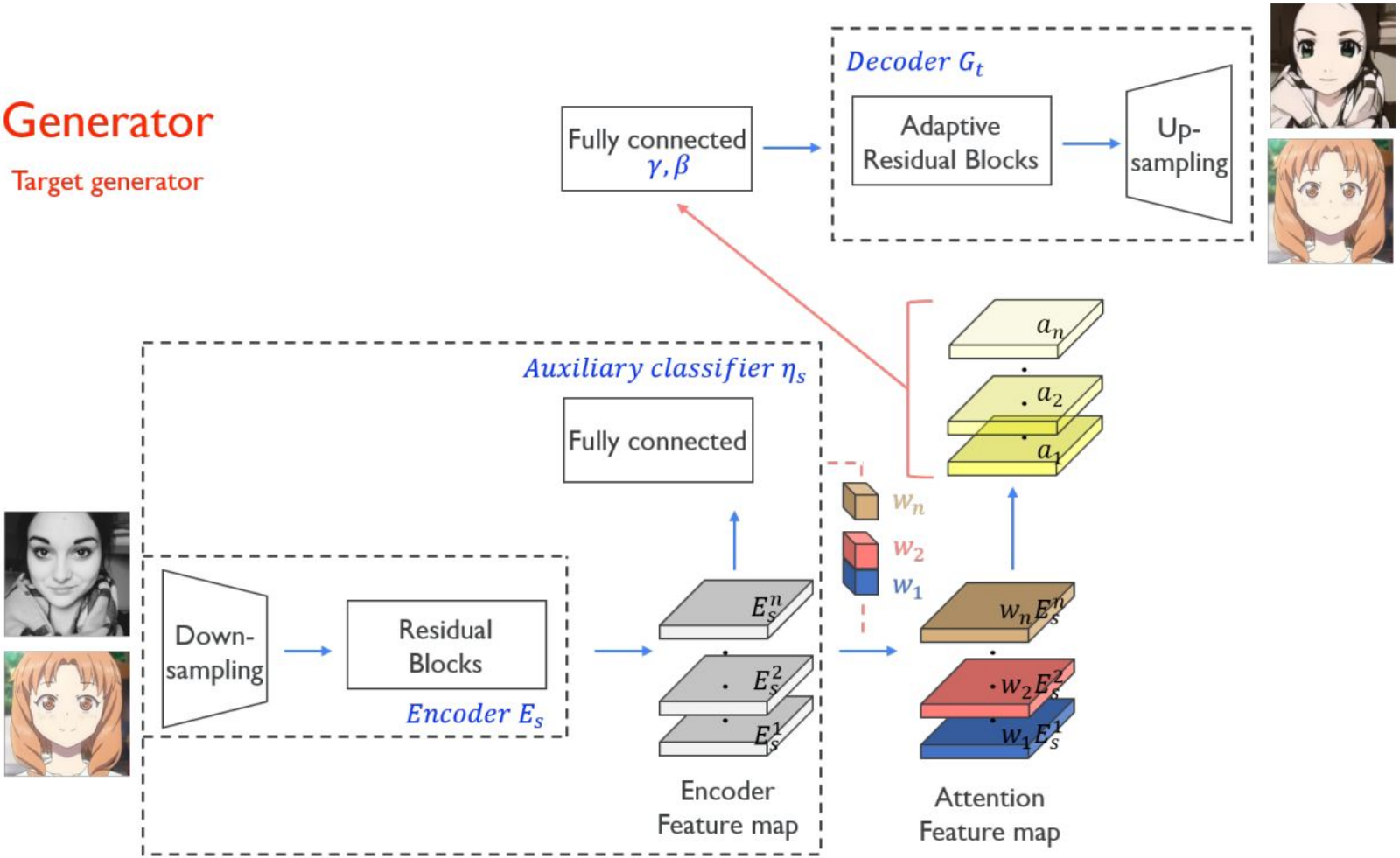
- The model consists of a Generator and Discriminator.
- The Generator itself has variational autoencoder consisting of and encoder and decoder.
- The autoencoder converts a real face image into a cartoon face image.
- The discriminator differentiates between real and fake cartoon images.
- The details of each of these are explained in the coming slides.

Generator

- Encoder:
  - Down-sampling of datasets into Residual blocks with Instance Normalization
  - Generation of Encoder Feature Map which represents feature in the images
- Auxiliary Classifier :
  - Learning of weights of feature map of encoder using sigmoid activation
- Generation of Attention Feature Map which is the product of the feature maps to their respective weights, according to their importance
- Decoder :
  - The fully connected layer are passed into the Decoder
  - Features embedded in Adaptive Residual Blocks with the help of AdaLIN
  - Generation of Target domain images by Up-Sampling Convolution Blocks

Generator

Target generator

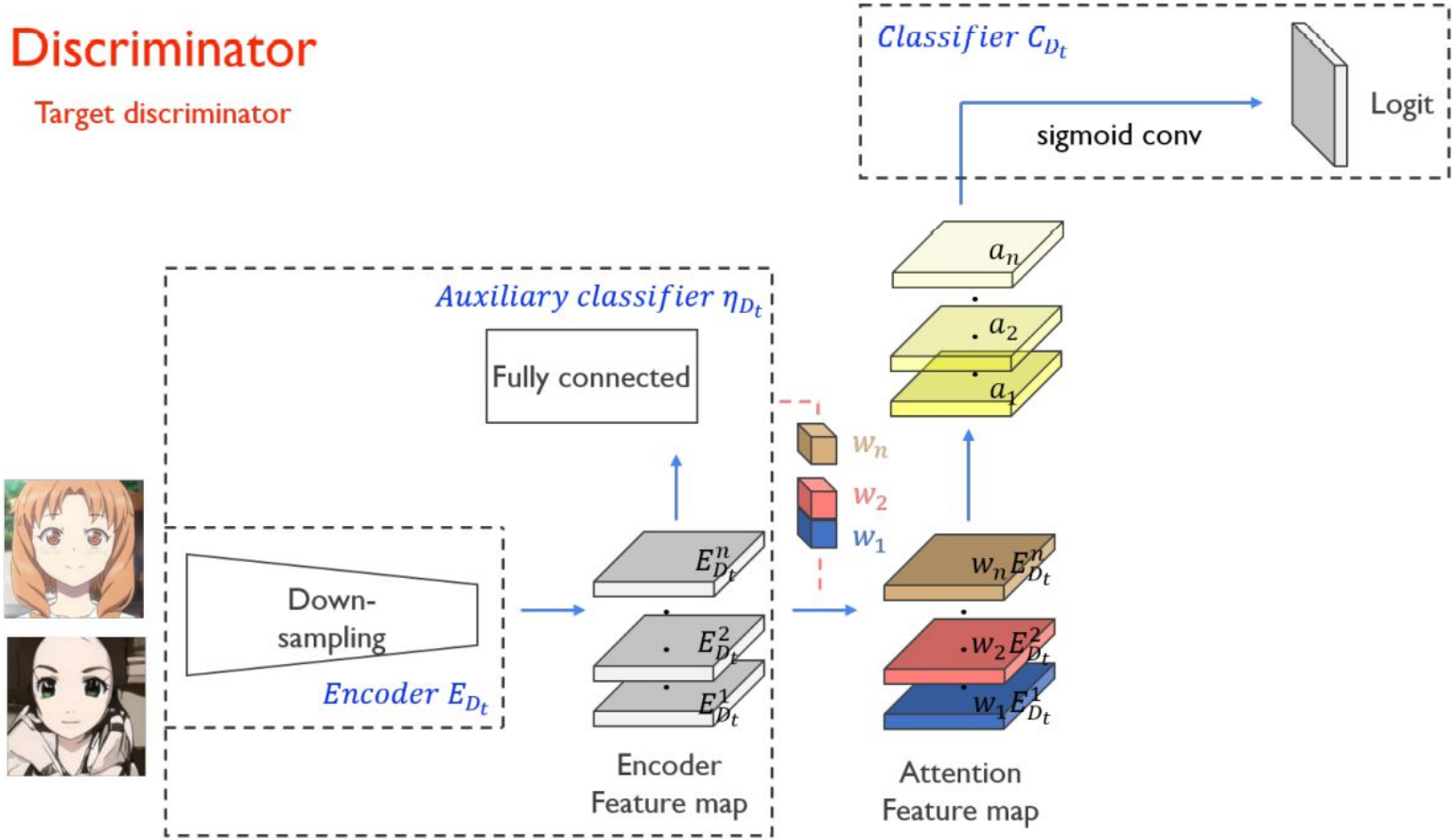


Discriminator

- Encoder :
  - Down-Sampling of datasets to generate Encoder Feature Map
  - Encoder Feature Map represents the features in the down-sampled images
- Auxiliary Classifier :
  - Helps to learn the weights of the feature map from Encoder
- Attention Feature Map helps in fine-tuning by focusing on the difference between real image and fake image in target domain
- Classifier :
  - Receives the results from Attention Feature and uses it to differentiate between real and fake images

Discriminator

Target discriminator



AdaLIN

$$AdaLIN(a,\gamma,\beta) = \gamma \cdot (\rho \cdot \hat{a_I} + (1-\rho) \cdot \hat{a_L}) + \beta,$$
$$\hat{a_I} = \frac{a-\mu_I}{\sqrt{\sigma_I^2+\epsilon}}, \hat{a_L} = \frac{a-\mu_L}{\sqrt{\sigma_L^2+\epsilon}},$$
$$\rho \leftarrow clip_{[0,1]}(\rho - \tau\Delta\rho)$$

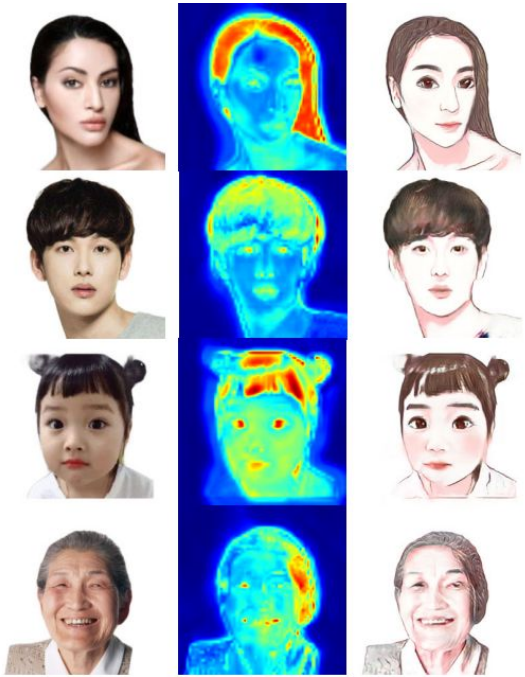
where μI , μL and σI , σL are channel-wise, layer-wise mean and standard deviation respectively, γ and β are parameters generated by the fully connected layer, τ is the learning rate and Δρ indicates the parameter update vector (e.g., the gradient) determined by the optimizer and ϵ is a small positive constant to avoid division by zero.

Need of AdaLIN

- Need to transfer features from real domain to cartoon domain.
- WCT(Whitening and Coloring Transform) - optimal algorithm but high computational cost.
- AdaIN - fast computation but gives sub-optimal results as it does not consider the correlation between different channels. Hence transferred features still have real domain influence.
- Layer Normalization considers the correlation between all the channels but sometimes fails to maintain the content structures from original domain as it aggregates all channels into one.
- Hence AdaLIN is proposed which incorporates advantages of both to give better results.

Class Activation Map (CAM)

- Helps the model to focus on more important regions distinguishing between source and target domains using Global Average Pooling
- Attention map of generator from real-face to cartoon face to generate Cartoon image



Losses Considered

Adversarial loss: An adversarial loss is employed to match the distribution of the translated images to the target image distribution

Cycle loss: A cycle consistency is applied to constraint to the generator as CycleGAN to alleviate the mode collapse problem

Identity loss: An identity consistency constraint is used to ensure that the color distributions of input image and output image are similar

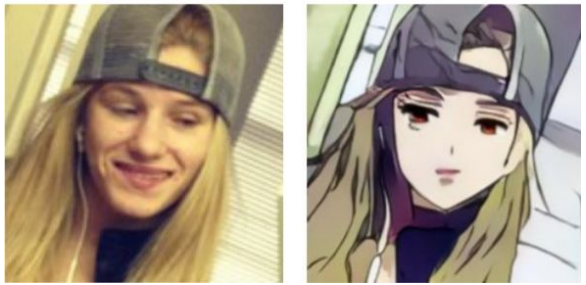
CAM loss: Loss so that auxiliary classifiers give correct weights to attention maps.

Face ID loss: To enforce the corresponding constraints between real faces and cartoon faces, the cosine distance of features between real-face image and cartoon-face image is used

MobileFaceNet for Face ID loss

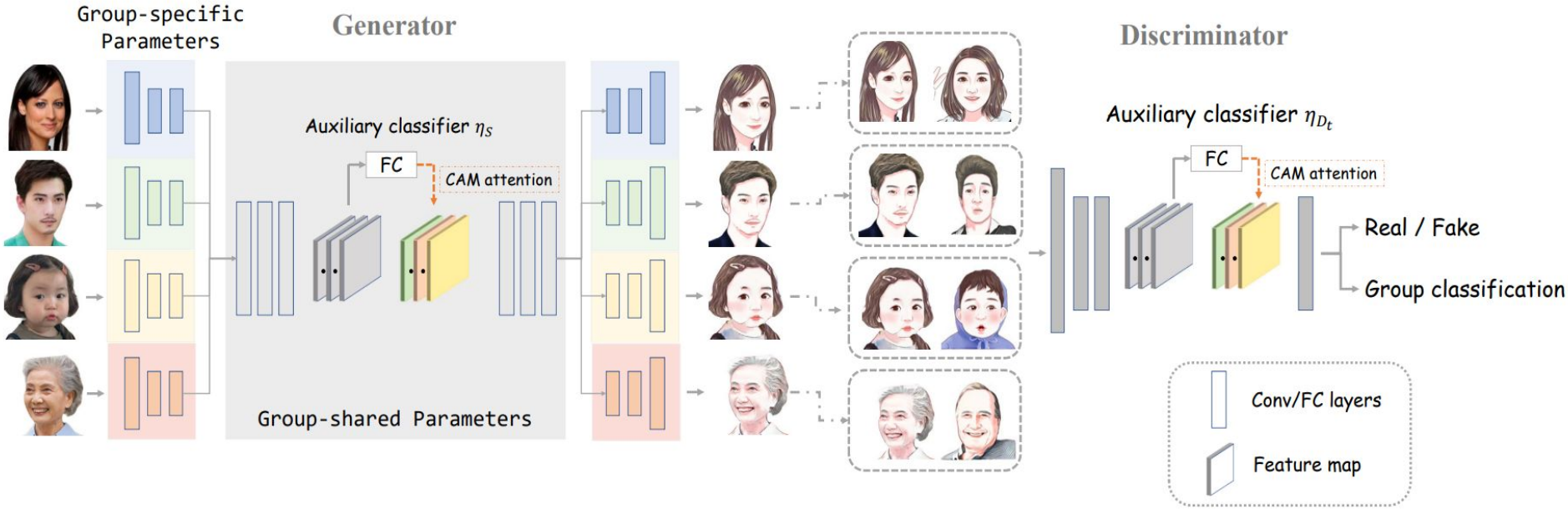
- To get the face features required to calculate the Face ID loss the MobileFaceNet is used.
- First the real image is passed to get its features and then the converted cartoon image is passed to get its features.
- These features used for getting the Face ID loss
- This loss is used so that the cartoon face generated still has the identity of the person from the real image.

Without Face ID loss

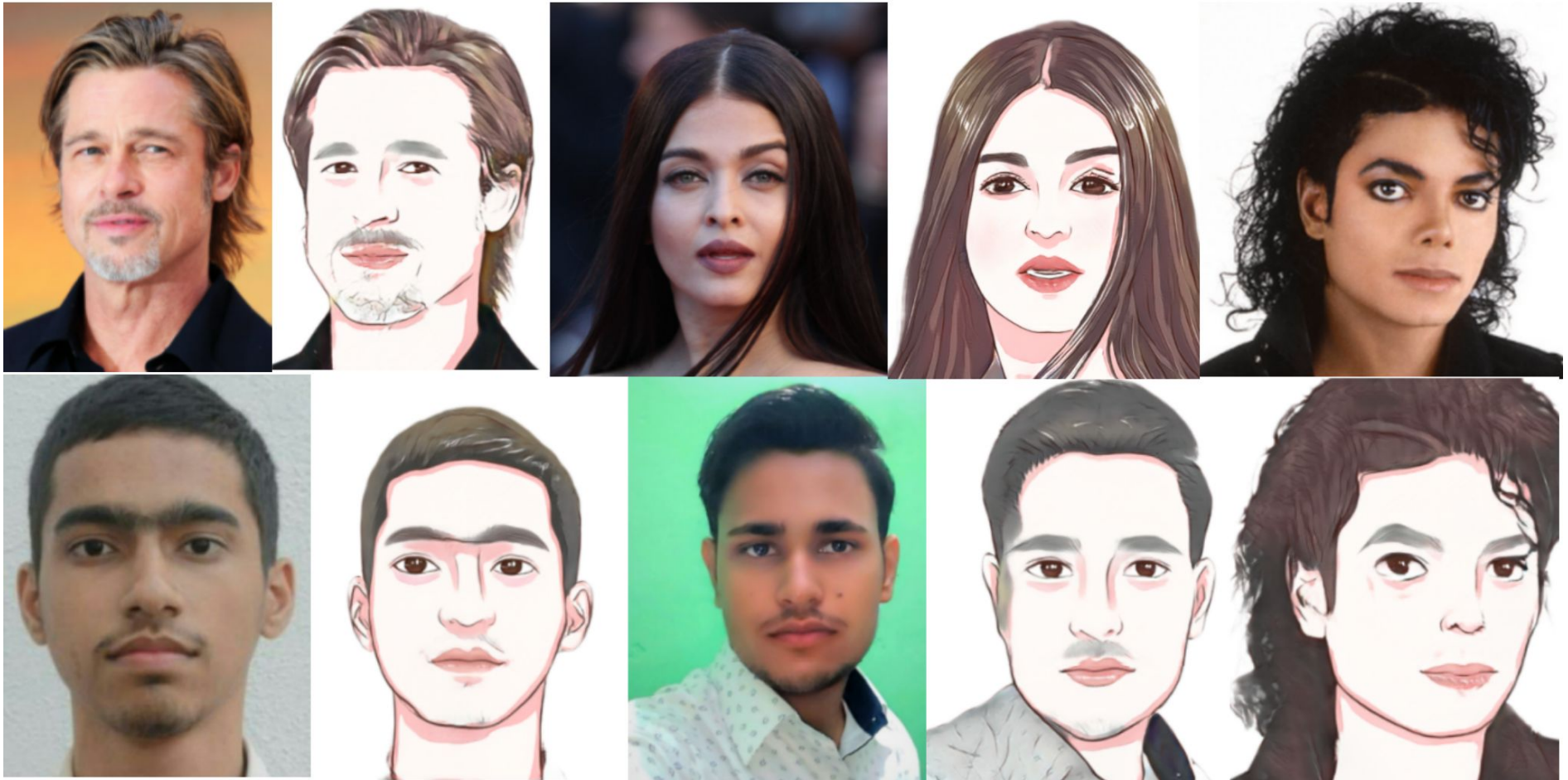


With Face ID Loss

Training Methodology

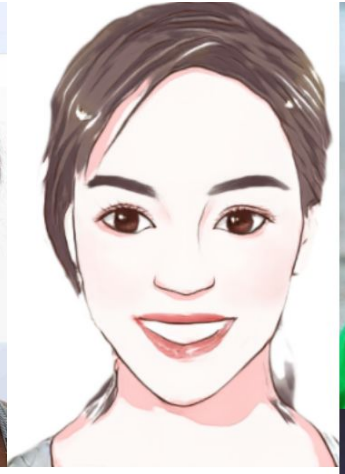
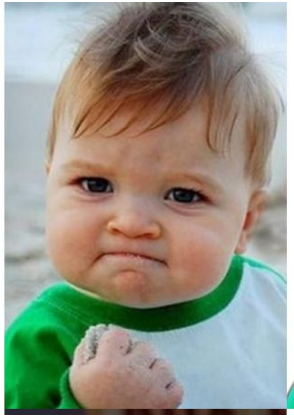
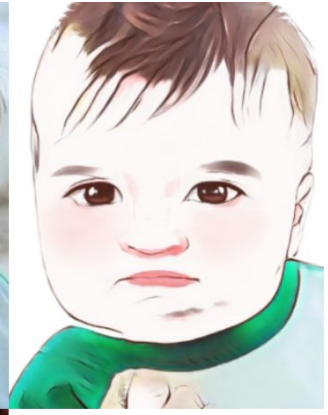


Results - The Good Ones

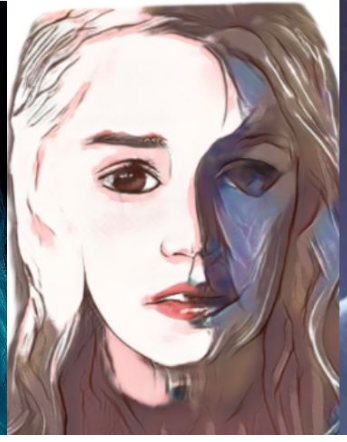


Results - The Bad Ones







Conclusion

- Objective: Real face photo → Cartoon face photo
- Dataset used and Preprocessing
- Reason U-GAT-IT was chosen
- Model Architecture
- Concept behind AdaLIN
- Concept behind CAM
- Use of MobileFaceNet for Face ID loss
- Knowledge Transfer based training method
- Results obtained from the model

Future Directions

- The results obtained clearly states the improvements required in the model.
- Implementation of the model to generate cartoon images for any scenery.
- The main features to focus for improvement will be :
  - Dealing with facial hair.
  - Dealing with shadows on face.
  - Persisting the colour complexity of the real image in its cartoon image.

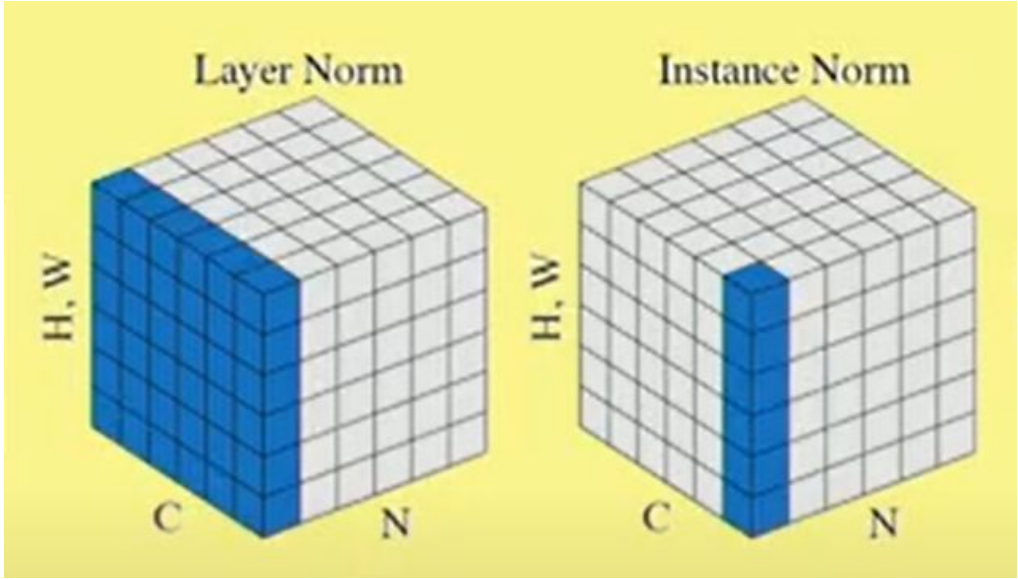
Thank you

Slides not presented but created for explanation

Layer vs Instance Normalization Brief Explanation

Normalization is the process of making the std dev as 1 and mean as 0 for a particular set of values. Layer and Instance normalization differ on the basis of the set of parameters they use for normalization.

$$Z = \frac{x - \mu}{\sigma}$$



Adaptive Instance Normalization(AdaIN)

Reparameterization done as:

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

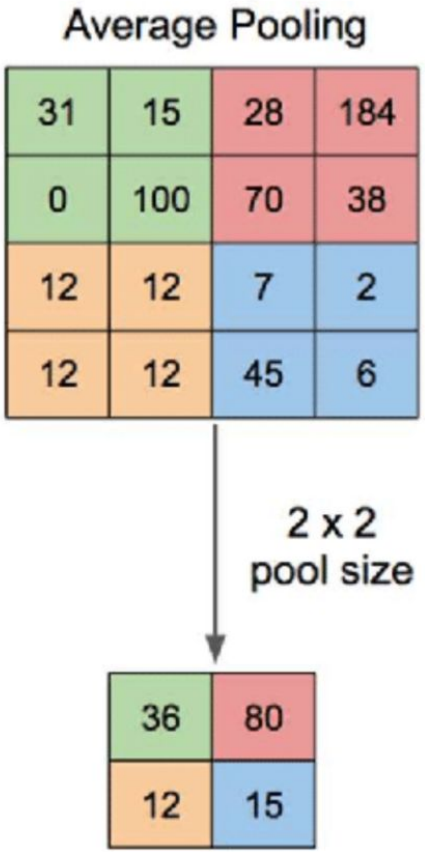
X is the source domain and Y is the target domain.

This is to convert distribution from source domain to the target domain.

Average Pooling

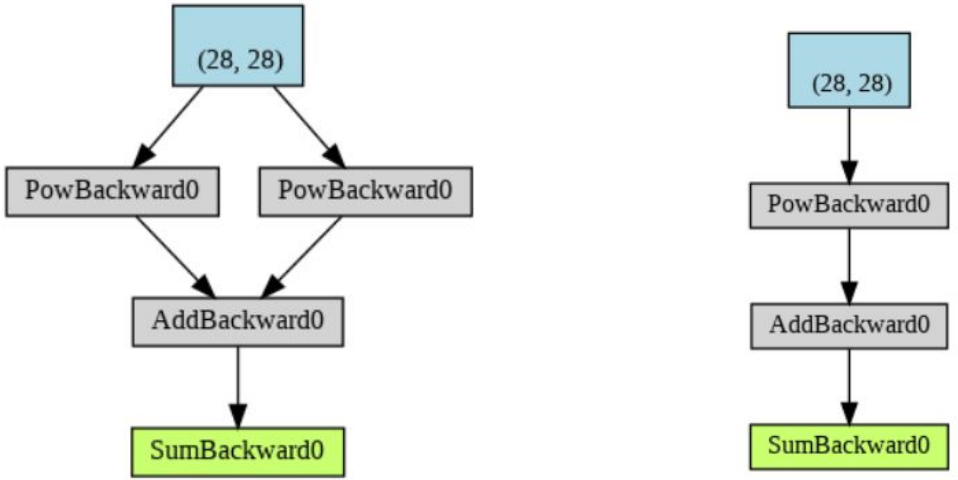
- Average Pooling is a pooling operation that calculates the average value for patches of a feature map, and uses it to create a downsampled (pooled) feature map.

- It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs.



Selective Backpropagation(detached clone)

- Pytorch has functions tensor.detach() and tensor.clone() for achieving this.
- With clone a copy of given tensor is created but the computation path is same so during the backpropagation the gradients flow to original tensor.
- With detach the new tensor is created and is detached from the computation graph



Losses

Adversarial loss: $$L^{s\to t}_{lsgan} = (E_{x\sim Xt}\,[(D_t(x))^2\,] + E_{x\sim Xs}\,[(1 - D_t(G_{s\to t}(x)))^2\,])$$

Cycle loss: $$L^{s\to t}_{cycle} = E_{x\sim Xs}\,[|x - G_{t\to s}(G_{s\to t}(x)))|_1]$$

Identity loss: $$L^{s\to t}_{identity} = E_{x\sim Xt}\,[|x - G_{s\to t}(x)|_1]$$

CAM loss: $$L^{s\to t}_{cam} = -(E_{x\sim Xs}\,[\log(\eta_s(x))] + E_{x\sim Xt}\,[\log(1 - \eta_s(x))])$$

$$L^{Dt}_{cam} = E_{x\sim Xt}\,[(\eta_{Dt}\,(x))2\,] + E_{x\sim Xs}\,[(1 - \eta_{Dt}\,(G_{s\to t}(x))^2\,].$$

Face ID loss:

$$L_{face} = E_{x\sim Xs}\,[1 - \cos(F(x), F(T^i_{s\to t}\,(x)))] + E_{x\sim Xt}\,[1 - \cos(F(x), F(T^i_{t\to s}\,(x)))]$$