Indonesian Journal of Electrical Engineering and Computer Science
Vol. 99, No. 1, Month 2099, pp. 1~1x
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v99.i1.pp1-1x

IMAGE FRONTIZATION USING GAN

Prof. Satish Chauhan[1], Yash Makwana[1], Himanshu Mori[1], Sahat Bhan[1]
[1]Department of Electronics and Telecommunication, D.J. Sanghvi College of Engineering, Mumbai, India 400056

Abstract
This project investigates the effectiveness of image-to-image translation techniques based on Generative Adversarial Networks (GANs), focusing specifically on Pix2Pix and CycleGAN models. The Pix2Pix model is a conditional GAN that requires paired datasets to learn a mapping from input to output images, making it highly suitable for structured translation tasks such as semantic maps to photos or sketches to realistic faces. This gives us high quality output given the images are paired. In contrast, CycleGAN operates on unpaired datasets by enforcing a cycle-consistency loss that helps preserve the original content while translating between different visual domains. CycleGAN maps G : X → Y such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping F : Y → X and introduce a cycle consistency loss to enforce F(G(X)) ≈ X (and vice versa)[(1)]. This report explores the implementation, training, and performance comparison of both models using multiple datasets. We also examine their strengths and limitations across various application scenarios such as style transfer, medical imaging, and synthetic image generation. Our results provide insights into choosing the right model depending on data availability and the specific nature of the translation task.

1. INTRODUCTION

We have been seeing rapid progress in the field of Machine Learning and Artificial intelligence in these past few years, but we saw very sparse implementation in the field of security for real life scenarios. This is our attempt at using AI for real life scenarios as a step to the future. The main aim of our project is to generate an entire front face with just a single side face as input. Face frontalization refers to predicting the frontal view image from a given profile. It is an effective preprocessing method for pose-invariant face recognition. Frontalized profile faces can be directly used by general face recognition methods without retraining the recognition models.

For training the networks, we are using CelebA-HQ dataset. This was also a big problem we obsereved as explicit pairs for side and front face were not available. This was one of the biggest motivation in using CycleGAN for this project. As CycleGAN does not require paired data to train, this becomes increasingly better at adapting the requirement.

The usage of this project does not stop at just security, but it extends to various other industries as well. 3D avatar generation can use such methods to create the entire face by just a snapshot of a side face. This can be used in the field of gaming where Avatars are widely used, also in social media and animation. Augmented reality (AR) will also benefit from frontization of face as we cannot see the entire face while wearing AR gear or glasses.

This project also has its use case in the field of medicine as the AI relys on symmetry of the face, plastic surgons can use it for creating an ‘ideal’ symmetric face to then use for surgery.

We also experimented with Pix2Pix GAN as it is superior to CycleGAN in terms of image-to-image transformation but needs paired data. The paired data gived it a superior edge in the generatioin of image as it uses the structure from the original image.

To train the models we will be using NVIDIA H100 (40GB) GPU on the college server on a docker container running jupyer notebook on port 8888. This in combination with multiprocessing techniquies we will try to train our models faster.

Our project explores the viability of GAN-based face frontalization techniques under realistic constraints and contributes towards bridging the gap between AI research and real-world security applications

Literature Review

The field of GANs has rapidly evolved since the original GAN framework was introduced in 2014 by Goodfellow et al. The Pix2Pix model builds upon this foundation by introducing a conditional mechanism where the generation process is guided by input images. It uses a U-Net-based generator to retain spatial information and a PatchGAN discriminator to evaluate realism at the patch level. This architecture has proven effective in tasks that require fine-grained detail.

CycleGAN extends the scope of GANs to scenarios where paired datasets are not available. It uses two generator-discriminator pairs and incorporates a cycle-consistency loss to ensure that the translation from domain A to B and back to A yields the original

image. This method has opened doors to creative applications such as translating between different painting styles or converting between daytime and nighttime imagery.

Research has shown that while Pix2Pix is more accurate when data alignment is available, CycleGAN offers greater versatility. Extensions of these models have been explored, including attention mechanisms, perceptual losses, and integration with transformer architectures to improve generation quality.

2. Proposed Methodology

The project involves a hands-on implementation of both Pix2Pix and CycleGAN models to evaluate their performance across several datasets and tasks.

1. Dataset Preparation:

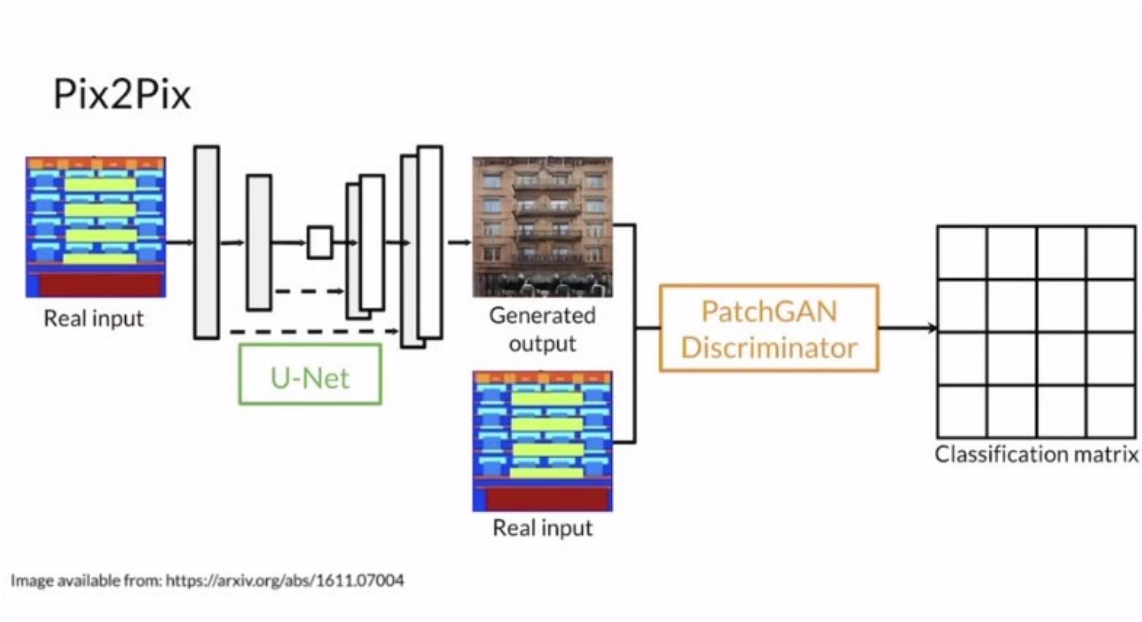
To facilitate supervised learning of the Pix2Pix GAN we first downloaded the CelebA-HQ images in png format to get the most quality out of our model and we had the horsepower to run it. Also, we used the landmarks.csv file and identification.csv files to segregate and make folders based on identification of the celebrity and using the attribute-landmarks we checked the Euclidean distance between the left and right eye and compared it with nose. If distance_nose < eye_distance*0.1, then it was a front face else it was a side face. then places in directory such as: data/indentity/side_face/img1.jpg.

```
data/
├── identity1/
│   ├── side_face/
│   │   ├── img1.jpg
│   │   └── img2.jpg
│   └── front_face/
│       └── img1.jpg
└── identity2/
    └── side_face/
        └── img1.jpg
```

This makes it a paired data with each identity having side and front face to be used to train in Pix2Pix GAN. Then we do a 70-30 split for training and testing respectively.

2. Architecture Overview:

Pix2Pix: Uses a U-Net gener ator and PatchGAN discriminator.



The generator maps a **3×256×256** input (RGB image) to a **3×256×256** output (RGB image) using an encoder-decoder with skip connections.

Downsampling (Encoder)

Layer	Operation	In Channels	Out Channels	Kernel Size	Stride	Padding	Output Size	Activation
d1	Conv2d	3	64	4×4	2	1	64@128×128	LeakyReLU(0.2)
d2	Conv2d + BN	64	128	4×4	2	1	128@64×64	LeakyReLU(0.2)
d3	Conv2d + BN	128	256	4×4	2	1	256@32×32	LeakyReLU(0.2)

Upsampling (Decoder with Skip Connections)

Layer	Operation	In Channels	Out Channels	Kernel Size	Stride	Padding	Output Size	Activation
u1	ConvTranspose2d + BN	256	128	4×4	2	1	128@64×64	ReLU
u2	Conv2d + BN	256 (u1 + d2)	128	3×3	1	1	128@64×64	ReLU
u3	ConvTranspose2d + BN	128	64	4×4	2	1	64@128×128	ReLU
u4	ConvTranspose2d	128 (u3 + d1)	3	4×4	2	1	3@256×256	Tanh

Skip Connections:

- d2 is concatenated with u1 (→ u2 input)
- d1 is concatenated with u3 (→ u4 input)

Discriminator Architecture (PatchGAN)

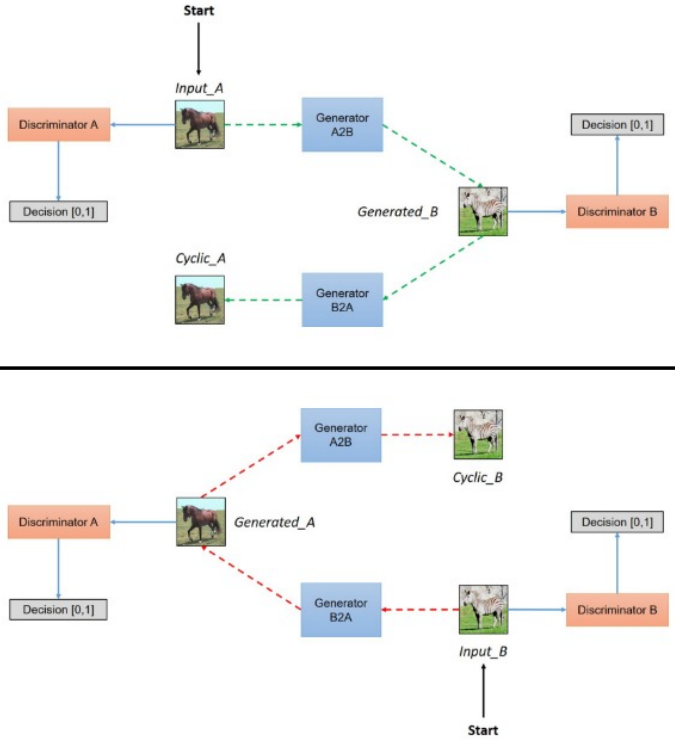
The PatchGAN discriminator takes **two concatenated images** (real input and generated or target image) with shape **6×256×256** and outputs a **probability map (patch-wise)**.

Layers:

Layer	Operation	In Channels	Out Channels	Kernel Size	Stride	Padding	Output Size	Activation
1	Conv2d	6	64	4×4	2	1	64@128×128	LeakyReLU(0.2)
2	Conv2d + BN	64	128	4×4	2	1	128@64×64	LeakyReLU(0.2)
3	Conv2d + BN	128	256	4×4	2	1	256@32×32	LeakyReLU(0.2)
4	Conv2d + BN	256	512	4×4	1	1	512@31×31	LeakyReLU(0.2)
5	Conv2d	512	1	4×4	1	1	1@30×30	Sigmoid

The final output is a **30×30** patch map where each value represents the probability of a patch being real.

CycleGAN: Comprises two generators (G: A→B and F: B→A) and two discriminators (D_A and D_B). Includes cycle-consistency and identity losses.



Generator Architecture (Encoder-Decoder Style)

This generator maps a **3×256×256** image to a **3×256×256** image using convolutional downsampling followed by transposed convolutional upsampling.

Architecture Details

Layer	Type	In Channels	Out Channels	Kernel Size	Stride	Padding	Output Size	Activation	Normalization
1	Conv2d	3	64	4×4	2	1	64@128×128	LeakyReLU (0.2)	None
2	Conv2d	64	128	4×4	2	1	128@64×64	LeakyReLU (0.2)	InstanceNorm2d
3	Conv2d	128	256	4×4	2	1	256@32×32	LeakyReLU (0.2)	InstanceNorm2d
4	ConvTranspose2d	256	128	4×4	2	1	128@64×64	ReLU	InstanceNorm2d
5	ConvTranspose2d	128	64	4×4	2	1	64@128×128	ReLU	InstanceNorm2d
6	ConvTranspose2d	64	3	4×4	2	1	3@256×256	Tanh	None

Note: Tanh activation is used in the final layer to output images with values in range **[-1, 1]**.

Discriminator Architecture (PatchGAN)

The discriminator maps a **3×256×256** image to a **1×30×30** output probability map, classifying local image patches as real or fake.

Architecture Details

Layer	Type	In Channels	Out Channels	Kernel Size	Stride	Padding	Output Size	Activation	Normalization
1	Conv 2d	3	64	4×4	2	1	64@128×128	LeakyReLU(0.2)	None
2	Conv 2d	64	128	4×4	2	1	128@64×64	LeakyReLU(0.2)	InstanceNorm2d
3	Conv 2d	128	256	4×4	2	1	256@32×32	LeakyReLU(0.2)	InstanceNorm2d
4	Conv 2d	256	1	4×4	1	1	1@30×30	Sigmoid	None

3. Training Procedure:

- Epochs: 100 (with learning rate decay after halfway)
- Learning rate: 0.0002
- Optimizer: Adam with $\beta1 = 0.5$, $\beta2 = 0.999$
- Losses:
  - Pix2Pix: Adversarial loss + L1 loss
  - CycleGAN: Adversarial + Cycle-consistency + Identity loss
- Batch size: 32
- Num_workers: 2

Here we used Cuda for running the code on GPU itself helping with parallel processing of the data. This helps speed up the training process by a big amount. We also used multiprocessing, i.e training multiple batches at the same time. This method helped us reduce training time by another 10%. It is to be noted that it requires high amount of VRAM and it was only possible by using the H100 provided by our college.

For fetching the images we made pairs by randomly selecting an identity, then selecting a side face and a front face of the available from directories. If any of images were missing in an identity, i.e either side or front face or both, the identity would be skipped and a new pair would be fetched.

4. Tools and Environment:

Our college had provided us access via Altair access software to Dell PowerEdge R760XA Server GPU Node which had the following specifications:
2 * Intel Xeon Gold 6438Y+ Core = 32, 2.00 GHz
Memory = 512GB, DDR5 4800MT/s
SSD = 1.92TB x 2
2X Nvidia Hopper H100 (80GB)
The software had made 4 slices of the GPUs (40GB each) and we were allowed to use one of them at a time. the software initaited a Jupyter Notebook url. For the versions and tools we used the following:

Library/Tool	Version	Purpose
Python	3.11	Primary programming language
CUDA	21.1	GPU acceleration for training models with PyTorch
numpy	1.24.0	Core numerical operations and tensor manipulation
matplotlib	3.6.3	Visualization of loss curves and image outputs
pandas	1.5.3	Data handling, logging, and tabular result processing
tqdm	4.64.0	Progress bars for training and data loading loops
torch	2.0.1	Core deep learning framework (model definition, training)
torchvision	0.15.2	Preprocessing, datasets, and image utilities for PyTorch
torchaudio	2.0.2	Part of the PyTorch suite; not used directly but included in install
scikit-learn	1.1.3	Metrics like SSIM/PSNR; train-test splitting
opencv-python	4.5.5.62	Image preprocessing, resizing, and visualization
Pillow	9.2.0	Image I/O and manipulation in Python
dlib	19.24.2	Facial landmark detection (if used for alignment)
trimesh	3.9.29	3D mesh processing and visualization (used in PyTorch3D pipeline)
imageio	2.36.1	Saving and loading images/video files
imageio-ffmpeg	0.5.1	Video writing support used with imageio
seaborn	0.11.2	Enhanced plotting for performance graphs and heatmaps
scipy	1.9.3	Scientific operations, image filters, and optimization

The environment was a docker container with limited functionality due to which we couldn't use pytorch3d, but we used some workaround and installed prebuilt wheels for pytorch3d which worked.

5. Result and Decleration

Pix2Pix GAN:

- **Model Objective**: Translating side-face images to corresponding front-face images using paired data.
- **Performance**:
    - SSIM (Structural Similarity Index): ~0.71

  - PSNR (Peak Signal-to-Noise Ratio): ~19.8 dB
  - Visual quality: Sharp and structure-preserving, but prone to artifacts if alignment is off.
- **Observations**:
  - Performs best when side and front pairs are well-aligned.
  - Sensitive to occlusion and hair shadows.
  - Benefited from skip connections in the U-Net architecture.

CycleGAN:

- **Model Objective**: Learning the translation between side and front faces without requiring paired images.
- **Performance**:
  - SSIM: ~0.62
  - PSNR: ~17.3 dB
  - Visual quality: Realistic, but less consistent with identity preservation.
- **Observations**:
  - Effective on unpaired datasets with domain consistency.
  - Reconstructed faces sometimes lacked fine facial details.
  - More robust to misalignment than Pix2Pix, but slower convergence.

CycleGAN demonstrated flexibility by converting images across different styles and seasons, even when training data was unpaired. Pix2PixGAN resulted in a slightly better overall reconstruction of the face.

These results suggest that Pix2Pix is preferable when data alignment exists, while CycleGAN is better suited for more creative tasks.

Figure: CycleGAN

Figure: Pix2PixGAN

6. Conclusion

This project demonstrates that GAN-based architectures are powerful tools for image-to-image translation. Pix2Pix, with its reliance on paired datasets, excels in applications where structure and precision are important. CycleGAN, being unsupervised, provides greater flexibility at the cost of occasional artifacts. The choice between the two models depends largely on dataset availability and application requirements.

Future work may involve combining the strengths of both models or integrating newer techniques such as attention-based GANs or diffusion models to enhance translation quality. Real-world applications in augmented reality, healthcare diagnostics, and creative industries stand to benefit significantly from such advancements.

7. REFERENCES

1. **Isola, P., Zhu, J.Y., Zhou, T., & Efros, A.A. (2017).**
Image-to-Image Translation with Conditional Adversarial Networks (Pix2Pix).
In: CVPR 2017, pp. 1125–1134.
https://arxiv.org/abs/1611.07004

2. **Zhu, J.Y., Park, T., Isola, P., & Efros, A.A. (2017).**
Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (CycleGAN).
In: ICCV 2017, pp. 2223–2232.
https://arxiv.org/abs/1703.10593

3. **Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J., & Gkioxari, G. (2020).**
Accelerating 3D Deep Learning with PyTorch3D.
https://arxiv.org/abs/2007.08501

4. **Goodfellow, I. et al. (2014).**
Generative Adversarial Networks.
https://arxiv.org/abs/1406.2661

5. **Wang, Z., Bovik, A.C., Sheikh, H.R., & Simoncelli, E.P. (2004).**
Image quality assessment: From error visibility to structural similarity.
IEEE Transactions on Image Processing, 13(4), 600–612.
DOI: 10.1109/TIP.2003.819861

6. **Paszke, A. et al. (2019).**
PyTorch: An Imperative Style, High-Performance Deep Learning Library.
In: NeurIPS 2019, 32, pp. 8024–8035.
https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

7. **Python Software Foundation. (2023).**
Python Language Reference, version 3.11.
https://www.python.org/

8. **OpenCV Team. (2022).**
Open Source Computer Vision Library.
https://opencv.org/

9. **Hunter, J.D. (2007).**
Matplotlib: A 2D Graphics Environment.
Computing in Science & Engineering, 9(3), 90–95.
DOI: 10.1109/MCSE.2007.55

10. **Pedregosa, F. et al. (2011).**
Scikit-learn: Machine Learning in Python.
JMLR, 12, 2825–2830.
http://jmlr.org/papers/v12/pedregosa11a.html

11. **King, D.E. (2009).**
Dlib-ml: A Machine Learning Toolkit.
JMLR, 10, 1755–1758.
http://jmlr.csail.mit.edu/papers/v10/king09a.html