# Frame interpolation using Convolutional NN and Generative Adversiral Network

## Intro

- I am working on Frame interpolation using CNN and GANs and trying to implement a research paper provided in the References.
- I have studied GANs and how to implement the basic gans using pytorch and i have also gone through the literature or theory required for the following project.
- I have also scrapped through many implementation wich are required as a subtask of the project.
- The Refinement network used in the research paper does'nt really clarify how it's implemented and how its taking inputs when asked with the mentor it was decide to just drop the network as it seem sto produce reasonable results without that (according to the paper)

## Dataset

- The Dataset to be used in Frame Interpolation should be in form of "Three consecutive Frames of video(F0, F1 and F2)".
- F0 and F2 are used as an input to the Generator and The F2 is used as a real Image in Discriminator.

The Dataset Used is derived form [Xiph.org :: Derf's Test Media Collection] link.

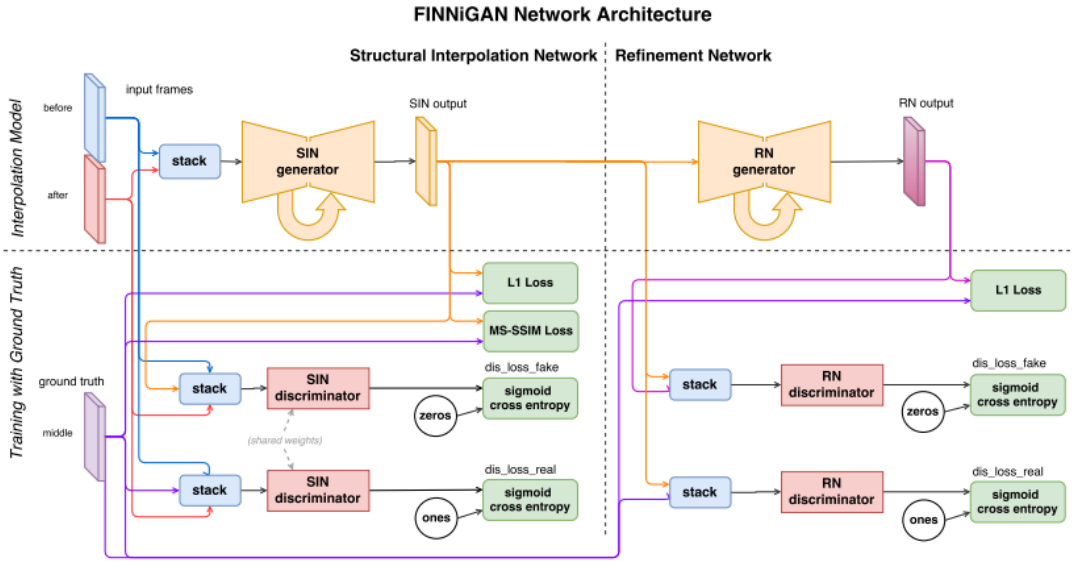
It consists of various videos ( in .y4m format) each consisting about 200-300 frames on average.

- I'll have to extract a decent amount of frames from the data set and arrange it in F0,F1,F2 format per data in a folder.

- More better way of forming the dataset is to sequentially take the frames and follow the above diagram.

## Model Architecture

I will be following the following Architecture from this [317.pdf (stanford.edu)] paper.




The Model contains 4 major NNs:

- SIN Generator
- SIN Discriminator
- RN Generator
- RN discriminator

**The SIN-GAN** : Takes in two F0 and F2 frames and Outputs the generated middle frame. The Generator's Loss function consists of a series of weighted sum of various losses:

1. MS-SSIM loss
2. L1  loss
3. Clipping Loss
4. Discriminator Loss

The use of MS-SSIM loss do'nt produce well coloured images (according to the paper).

**The RN-GAN** : It is used to tackle the above problem of MS-SSIM loss to improve the coloured images.

I will be using the following Repo for MS-SSIM and SSIM implementation in Pytorch: [VainF/pytorch-msssim: Fast and differentiable MS-SSIM and SSIM for pytorch. (github.com)]

since Pytorch don't provide an internal implementation.

**Generator** : The internal implementation of the Generator and Discriminator are given in the paper.

The SIN architecture was largely inspired by the work done by Isola et. al [6], and is shown in Figure 1. Given a sequence of frames, {F1, F2, F3}, the Generator takes the image doublet {F1, F3} as a 6 channel input. Let $C_K^{F-S}$ denote a convolution-batchnorm-lrelu-convolution-batchnorm-lrelu block with $K$ filters, a $F \times F$ filter size, and a $S \times S$ stride for each convolution. Additionally, let $T_K^{F-S}$ represent a resize-convolution-batchnorm-lrelu-convolution-batchnorm-lrelu block with $K$ filters, a $F \times F$ filter size, and a $S \times S$ stride for each convolution. The resize operation is a bilinear resize. Finally, let $D_K^{F-S}$ represent $T_K^{F-S}$ with dropout applied on the output of the last lrelu block. The Generator architecture is $C_{32}^{3-1}$-$C_{64}^{3-1}$-$C_{64}^{3-1}$-$C_{128}^{3-1}$-$D_{128}^{4-1}$-$D_{64}^{4-1}$-$D_{64}^{4-1}$-$D_{32}^{4-1}$. All Leaky ReLUs use 0.2 for the leak slope, and the dropout probability is 0.5. The generator uses a U-net structure [11], so the output of the $i^{th}$ convolution block is stacked with the output of the $j-1$ deconvolution block as the input to the $j^{th}$ deconvolution block, where i and j count from the ends toward the middle. For example, the output of $C_{32}^{3-1}$ is stacked with the output of the second $D_{64}^{4-1}$ as the input to $D_{32}^{4-1}$. This allows the network to retain structural information during upsampling that may normally be lost. The output of the generator is a 3 channel image.

The U-net structure also needs to be implemented because of no such class provided in pytorch.

**Discriminator**:

This simply is an Image classification model , build upon CNN and uses the Middle (F1) and the Generated images from the Generator to produce the Discriminator Loss.

Let $C_K^{F-S}$ denote a convolution-batchnorm-lrelu block with $K$ filters, a $F \times F$ filter size, and a $S \times S$ stride for each convolution. The architecture for the Discriminator is $C_8^{4-2}$-$C_{16}^{4-2}$-$C_{32}^{4-2}$-$C_{64}^{4-2}$-$C_1^{4-2}$.

## Pipeline Description

Workflow:

1. Parsing the dataset and compiling it into Test and Train dataset.
2. Use of a parser, which will be just a python script to extract frames from the data.
3. Designing the model based on the architecture described above.
4. The model will be written in form of Generator and Discriminator classes in a python script.
5. **Note:** I will be dropping the Refinement Network due to the reason mentioned in the abstract.
6. I will be using the U-Net based model and will be implementing it in the pytorch.
7. Writing the discriminator and the Generator losses is another file, since there are about 4 losses we need to have a file which does this.
8. The losses would be MS-SSIM loss, L1 loss, clipping loss, discriminator loss.
    a. L1 Loss: It will be calculated with help of the **Generated image** and the **target(F1)**.

$\ell_1$ **loss:** The $\ell_1$ loss is computed as

$$\ell_1 = \frac{1}{HWD}\sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{k=1}^{C}\frac{1}{255}|p_{i,j,k}^{SIN} - p_{i,j,k}|$$

    b. MS-SSIM: we will have to pre-process the **generated image** and **target** so they are valid inputs to this implementation, we add back the mean images, clip the image to a valid range, and convert the images to gray-scale, in order to apply the MS-SSIM.
    c. Clipping loss: Since the MS-SSIM's inputs need to be clipped which reduces the gradient of the image, we use clipping loss which compares between the **clipped** and the **Unclipped images.**

$$\text{Loss}_{\text{clipping}} = \frac{1}{HWD}\sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{k=1}^{C}(\bar{p}_{i,j,k}^{SIN} - p_{i,j,k}^{SIN})^2$$

    d. Discriminator Loss:
        i. Discriminator_fake_loss: Takes in **F0,F2 frames** and uses **Sigmoid_cross_entropy** on it to compute loss **with** the **F1 frame.** It

basically tells us how far is discriminator from telling the image is fake.

ii. Discriminator_real_loss: Takes in **F0,F2 frames** and uses **Sigmoid_cross_entropy** on it to compute loss **with** the **Generated frame.**It basically tells us how far is discriminator from telling the image is real.

iii. We need the fake_loss and real_loss both to decrease since better the discriminator telling the image to be real better it is ast telling it is fake and better the Generator becomes.

$$\text{Loss}_{\text{gen}} = -\log D_{\theta_d}(G_{\theta_g}(F1, F3)) \tag{1}$$

$$\begin{aligned}\text{Loss}_{\text{dis}} = &-\log D_{\theta_d}(F1, F2, F3) + \\ &-\log(1 - D_{\theta_d}(F1, G_{\theta_g}(F1, F3), F3)\end{aligned} \tag{2}$$

iv. Loss_gen will go into the the generator, and the Loss_dis will be coded in the discriminator itself.

9. Finally, the model will be trained for certain amount of Epoch on the Google Colab notebook since it is more suitable to debug the code there rather than running the entire code again and again in pytorch.
10. The final thing would be to add certain metrics to track the progress of the training. One of the metric would obviously be to track all the losses, The paper also uses SSIM and PSNR as a metric for the model.

## Literature Review

**Convolutional Neural Network:**

The utilization of convolutional layers in conjunction with GANs has demonstrated remarkable results, as observed in DCGANs. Therefore, implementing similar architectural and structural designs offers a viable approach to achieving satisfactory outcomes.

The only difference being in DCGANs we upsample the random noise but here we first downsample the images (input images) and then upsample the results to generate the images.

**L1/L2 losses:**

L1 and L2 losses are frequently utilized to distinguish color intensity variations between pixels of generated and actual images. However, these metrics are not capable of assessing the structural integrity of images since humans perceive groups of pixels, rather than individual ones.

Here, we are only gonna use L1  as a loss in conjunction with other losses.

$$\ell_1 = \frac{1}{HWD} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{C} \frac{1}{255} |p_{i,j,k}^{SIN} - p_{i,j,k}|$$

Where H, W, C are the dimensions of an image and |p SIN i,j,k − pi,j,k| is the absolute difference between a specific pixel of the generated image and the target image

**General Adverserial Network:**

Frame interpolation has been shown to achieve significant results using only CNNs through the encoder-decoder process. However, the main challenge lies in image generation, and GANs have recently produced relatively high-quality images. Therefore, this implementation is focused on a GAN setup based on CNNs to address this problem.

GANs involve an adversarial process between two models, the Generator and the Discriminator. The Generator attempts to create images, while the Discriminator tries to distinguish between real and fake images. Both models work in opposition to each other to improve their performance.

**Batch Normalisation layers:**

Normalizing the data has been shown to be an effective method for speeding up the training cycle. By achieving an evenly distributed data, the gradient descent algorithm rarely experiences abnormal jumps and approaches the optimal value at a relatively smooth rate. Batch normalization is a technique that normalizes the activation of each layer before inputting it into the subsequent layers..

**Dropout Layers**:

Dropout layers are a regularization technique used in neural networks to prevent overfitting by randomly dropping out neurons during training

**SSIM and MS-SSIM Loss:**

The Structural Similarity method is commonly used for evaluating image similarity based on human perception. The Structural Similarity Index (SSIM) quantitatively measures this similarity. Nevertheless, the SSIM has a disadvantage of being suitable for a specific display resolution and viewing distance, making it a single-scale approach. To address this problem, the Multi-Scale Structural Similarity Index (MS-SSIM) applies low-pass filters, down-samples the image at different stages, and combines the results geometrically through iterative processes.

**U-Net Structure:**

UNet is a neural network architecture that is commonly used for image segmentation tasks. Its structure is composed of a contracting path, which encodes the input image into a low-dimensional feature representation, and an expansive path, which decodes the features back into the original image resolution while preserving the spatial information. The network's U-shaped structure allows for skip connections between the encoder and decoder, enabling it to capture both local and global features effectively.

## Tech Stack Used

Languages: Python

Libraries and frameworks used:

- pytorch(torch.nn): To be used for making the model.
- torch.nn,matplotlib,torchvision, google colab for training purposes.
- OpenCV and finn/datasets.py at master · apoorva-sharma/finn (github.com) for frames extraction from the Dataset
- jaxony/unet-pytorch: U-Net implementation for PyTorch based on https://arxiv.org/abs/1505.04597 (github.com), for Implementing the U-Net Structure.
- VainF/pytorch-msssim: Fast and differentiable MS-SSIM and SSIM for pytorch. (github.com) , for SSIM and MS-SSIM loss calculation.

## References


- How Convolution Neural Networks interpret images | by Aseem Kashyap | Towards Data Science
- (229) Deep Learning with PyTorch: Zero to GANs - YouTube
- 317.pdf (stanford.edu)
- Implementing Bilinear Interpolation for Image Resizing | by meghal darji | Medium

- apoorva-sharma/finn: Deep Convolutional Neural Network for Video Frame Interpolation. (github.com)
- Creating and training a U-Net model with PyTorch for 2D & 3D semantic segmentation: Model building [2/4] | by Johannes Schmidt | Towards Data Science