ECE 579 Intelligent Systems, Fall 2023
Final Project Report

Lung X-Ray Data Augmentation using Generative Adversarial Networks

Student Names: Abhiram Varma Gadhiraju (UMID: 36893508)
Meghanath Payasam (UMID: 37720961)
Bhuvana Chandra Jammu (UMID: 40033893)

Department Name: Data Science

Responsibilities of each student:

- Abhiram Varma Gadhiraju (36893508): EDA, Data cleaning and preparation.
- Meghanath Payasam (37720961): Model building and customized training
- Bhuvana Chandra Jammu (40033893): Building visualization & evaluating the model.

Introduction:

The infections related to the rise in deathrate related to Lungs is pneumonia. Prior detection of the infection is crucial in reducing the mortality rate. As the diagnosis of the infection is majorly based on the chest Xray of the individual, detecting the infection can be serious task involved. This can be solved using advanced tools like Machine Learning based Algorithms. ML models such as CNNs use bigger datasets to acquire the desired accuracy and precision in detecting infection. As extracting many X rays is an expensive task, the need for data synthesis is raised. Classic data augmentation techniques are image duplications, CNN models. One of the advanced techniques for augmenting new images is GAN. In this project, we would like to propose a GAN model which generates synthesized images of the Lung Xray dataset provided. The dataset used is a Lung Xray dataset of training samples 3883 images with a dimension of 28x28x1.

Topic of Project:

In this study chest X-rays showing pneumonia patients are shown using generative adversarial networks (GANs) in the field of computer vision. GANs which consist of a generator and a discriminator are used to produce artificial images that closely resemble the training data. The discriminator in GAN is designed using a CNN and generator acts as counterpart of the CNN where it takes noise and generates an image. This project presents an intelligent system involving neural networks designed to create augmented images of real images using unsupervised learning and briefly involvement of transfer learning.

Background Information:

Classic data augmentation addresses are widely used especially in machine learning to increase the number of datasets before the invention of Generative Adversarial Networks (GANs). By increasing the amount of data these techniques aim to improve the durability of machine learning models. Traditional data augmentation methods like color space manipulation were used in the past even before GAN's which have their limitations to generate unique and highly accurate data samples. Later, CNNs are implemented to achieve better resolution images. These technologies gave control over feature preservation and loss reduction. Particularly in areas like picture generation and data augmentation GANs can produce data that is identical to the source dataset improving model performance and adaptability.

In this report, we shall present the GAN model designed in our project and evaluate the model using different metrics involving generator loss & discriminator loss, and Inception score.

Description of technologies related to the project:

CNN: CNNs convolutional layers are used for processing the pixels of images through kernels which transforms and condenses the pixel while preserving the required features in the image. CNNs are mainly used in operations of image detection and classifications due to their property of training directly from the data.

Generative Adversarial Network(GANs): They are a neural networks which can generate synthetic data from the real data. GANs are implemented based on the principle of CNNs. It has two blocks namely generator and discriminator. Both blocks implement CNN architecture in opposite way to complete the desired training process.

Generator: The Generator creates new synthetic images from a random noise. It takes random data and transforms it into images that resemble actual data using the designed layers of CNNs in transposed way. The objective of generator is to make these newly created images as good as the ones from where it was trained on. The synthesized images are then passed to the discriminator for classification process. In generator's CNN layers, the random noise given as input will propagate through various convolution layers, which eventually gets up sampled to get the desired image of specified dimensions.

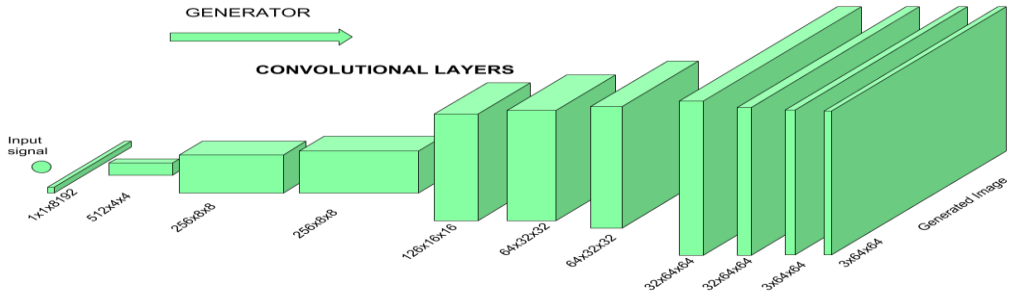

Fig: Generator Network in GAN

Discriminator: The Discriminator is used to determine whether the image is real or fake. Differentiating between actual images from the collection and artificial images created by the Generator is its main task. Discriminator is a standard CNN classifier which condenses, and extracts features from the image while the image is passed through different convolutional layers. It becomes better at identifying the fake images as it views a greater number of images in.

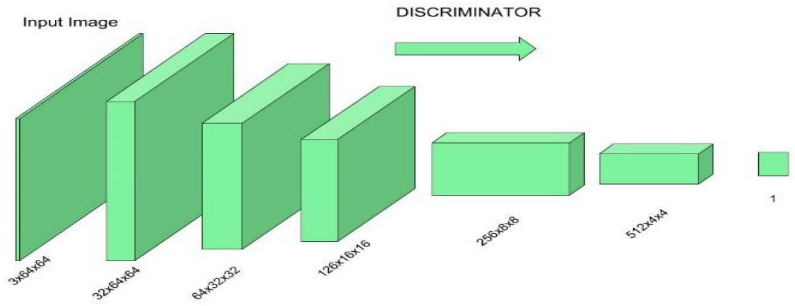

Fig: Discriminator Network in GAN

InceptionV3 model

The Inception Score is used to evaluate the quality of images generated by GANs. Based on the created image's quality and variety it calculates their diversity. The Inception Score uses the pre trained InceptionV3 model, a convolutional neural network that is primarily used for tasks related to image classification. With more training on a large dataset InceptionV3 can identify a wide range of objects and patterns in photos. The Generator component is used to create artificial images.

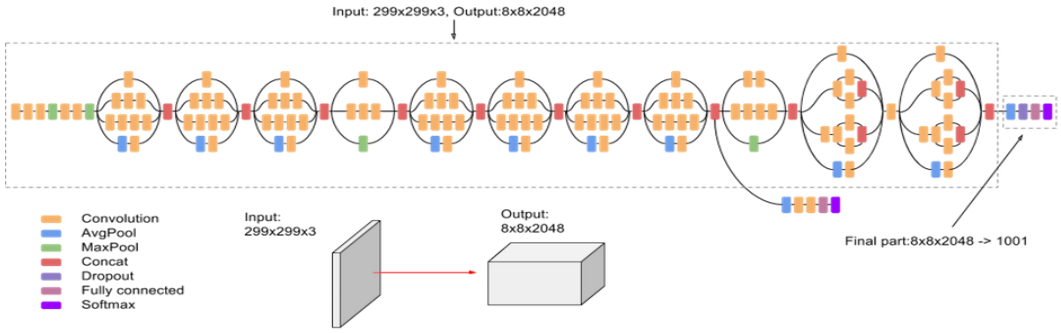

Fig: Convolution layers of InceptionV3

Methods used for data augmentation with GANs:

Libraries used: Pandas, Cv2 are using in data extraction and storing. Numpy is used in converting image tensors to numpy arrays. tensorflow.data.Dataset is used in building image tensors tensorflow.keras is used for model building.

Preparing data for the model: The dataset used for training the model is an image Xray dataset with the size of 3883 samples. As it an unsupervised model the data is not split into test train basis.

Cv2 is the module used to perform image processing steps in loading the data. The images loaded are stored in a list with 28x28x1 as dimensions of each image. The image list is converted to a tensorflow.data.Dataset to achieve the required batching and normalization of the images. The batch size is set to 256 samples and each sample is normalized to scale of [-1,1]. As CNN models required input tensors as NumPy arrays, the dataset is converted to numpy() using dataset_to_numpy () method.

Designing Generator/ Discriminator:

Generator: The generator is designed to be a sequential model with an input of a dense layer with 7x7x256. As generator takes a random noise as input seed, the dense layer is designed to map a 100-dimensional random noise vector in a higher-dimensional space. The dense layer is followed by 3 transposed convolution layers with 128,64,1 filters respectively. Each convolution layer is composed with Batch Normalization layer to apply stability and faster training of the model. The out 7x7x256 from dense layer is up sampled through the three convolution layers. We used LeakuReLU as activation function over relu to avoid dying state situation during the leaning process. As LeakyReLU allows some negative values to pass through, the network will not be halted with only zero as output in case of standard ReLU function.

Discriminator: The discriminator is designed to be a CNN layer which starts with a dimension of 28x28x1 and outputs a tensor of shape (None,1). It contains two convolution layers each with 64 filters and 128 filters respectively. A dropout of 30% is used to gain regularization and reduce overfitting. The strides are set to (2,2) and LeakyReLU is used as activation function for the same reason as generator. The CNN layers are followed by flattened and dense layers to obtain the required dimensions to classify the image as fake or real by the discriminator. The total number of parameters in discriminator

network is 212865 which is less compared to generator network as the generator needs a more complex network to generate a synthetic image from random noise.

Building custom training loop for GAN:

Training a GAN involved parallel adaptation to the training data by both generator and discriminator. Initially the loop begins with generation of random noise using tf.random.randn() or tf.random.normal(). Parallelly, the discriminator is trained with the real image for every epoch. The optimizers used are Adam, RMSprop and loss function used is Binary Cross Entropy. As Adam is also said to be best for its faster convergence and better performance it is used in both generator and discriminator to compare the two optimizers. Here, the tf.GraduentTape() is used to calculate the autodiff for generator and discriminator loss for respective parameters and this gradient is inputted to optimizer to get better trainable parameters. In training the GAN, we also used RMSprop optimizer to compare the impact of optimizer when changed from Adam to RMSprop.

Evaluation metrics:

Inception score is an important matric used in evaluating the GAN model. Using Adam optimizer gives 10 times less inception score compared to using RMSprop optimizer. (Adam = 1.0008/RMSprop = 1.0076).

Experiments & Results:

a) Generator/ Discriminator Loss

We experimented with the GAN model by changing the epochs and changing the optimizer to examine the performance of the model. We recorded the Inception score using InceptionV3 for both optimizers at epoch of 50. The learning rate is 0.001 for all models and loss is Binary cross entropy is considered for all models.

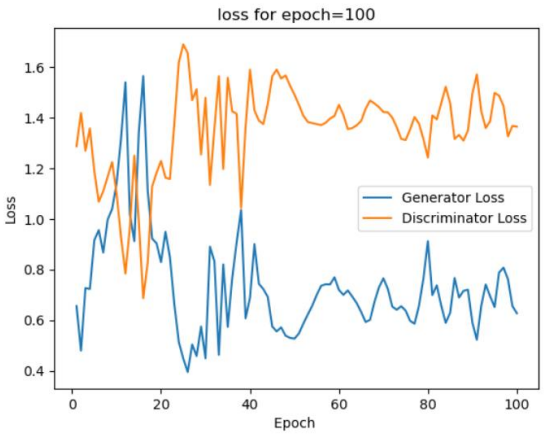

Fig1: Losses (Optimizer: Adam)

Using Adam optimizer has huge fluctuation in generator and discriminator losses. This indicates that the optimizer is reducing the loss effectively while model training.

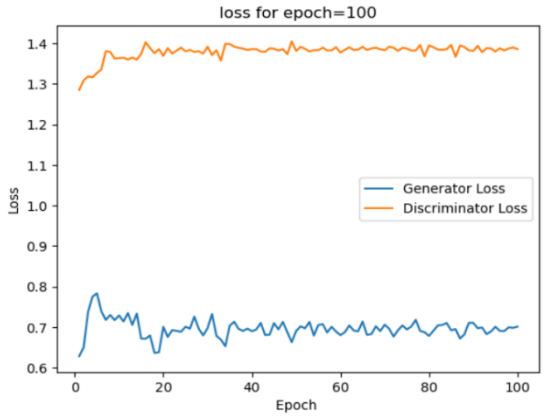

Fig2: Losses (Optimizer: RMSprop)

When Adam optimizer replaced with RMSprop, the losses are stabilized. The generator loss has decreasing trend from 0 to 50 epoch which later maintained even more stability.

b) Synthetic images from Generator

Below are the figures showing generated images at 50 & 100 epochs. Fig3, Fig5 displays synthetic images generated using Adam optimizer at epochs 50,100 respectively. Fig4 and Fig6 display synthetic images generated using RMSprop optimizer at epochs 50,100 respectively. It is evident that using RMSprop with same convolution layers will improve the model performance.

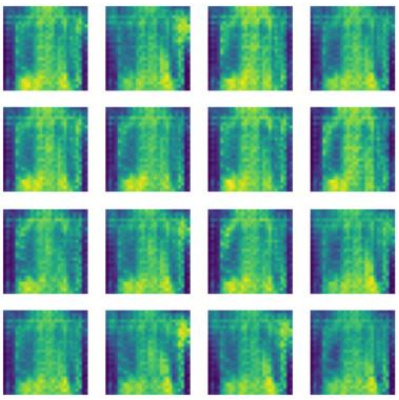

Fig3: Using Adam(epoch=50)

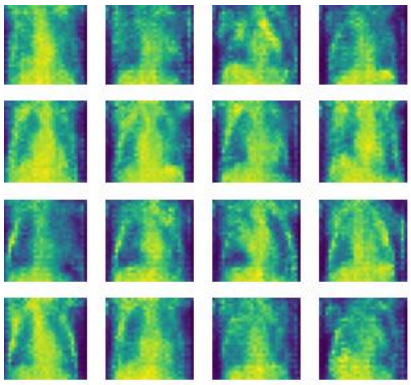

Fig4: Using RMSprop(epoch=50)

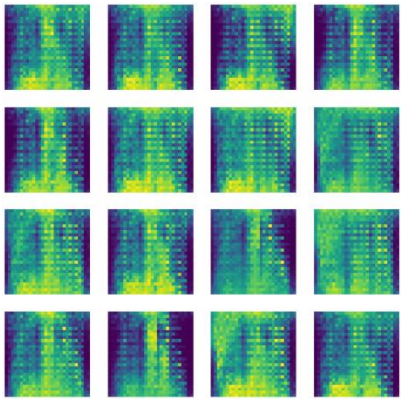

Fig5: Using Adam(epoch=100)

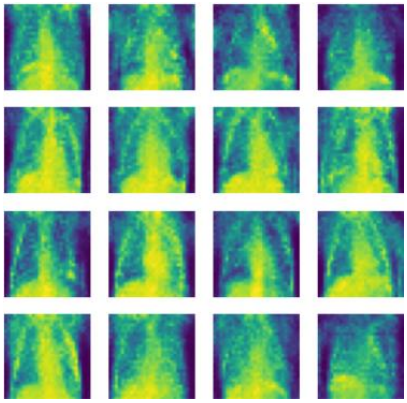

Fig6: Using RMSprop(epoch=100)

Conclusion:

We built a GAN model using unsupervised learning and experimented using different optimizers and epoch rates. It is evident that the optimizer functions impact learning. Also, we used InceptionV3 model as reference to calculate the inception score of the models with different optimizers.

Learning Aspects: Through this project we learned the architecture required to build an intelligent system and also discovered various aspects of achieving better results from the model. By performing evaluation metrics, we discovered mapping of losses and transfer learning using InceptionV3.

References:

[1]: Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. Data Augmentation Using GANs. In Proceedings of Machine Learning Research, 2019.

[2]: Fadaee, Marzieh, Bisazza, Arianna, Monz, Christof. Data augmentation for lowresource neural machine translation. arXiv:1705.00440, 2017.

[3]: L. Engstrom, D. Tsipras, L. Schmidt and A. Madry, A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations, 2017.

[4]: J. Jin, A. Dundar and E. Culurciello, Robust convolutional neural networks under adversarial noise, 2015.