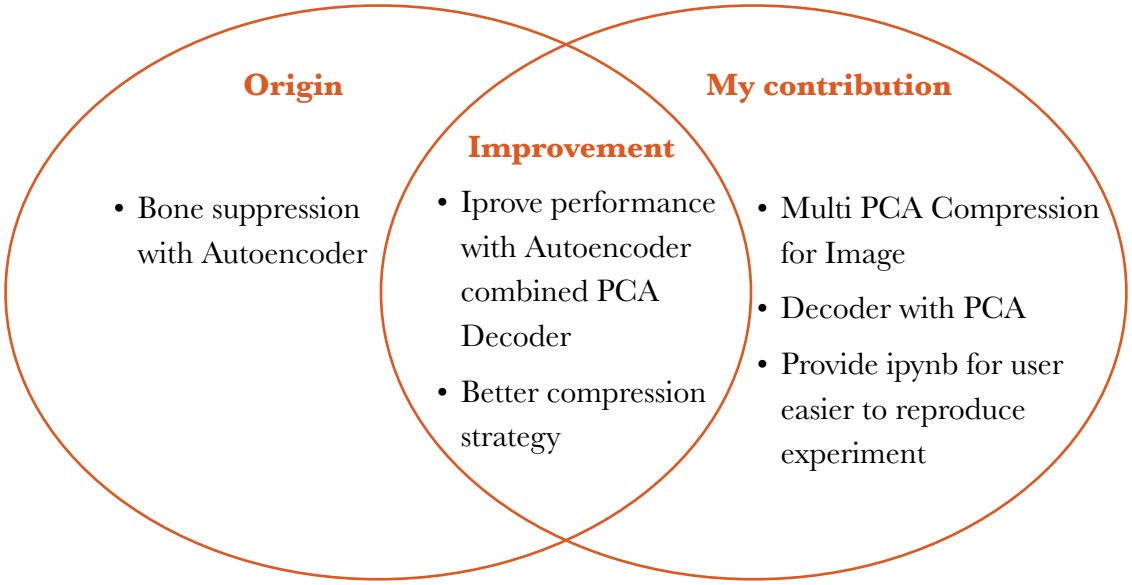
Yi-Hsiang, Chin
yc5859
December 21, 2022

Bone Suppression with Auto-encoder and PCA compression

1. Quad Chart

Purpose	Results
• **Improvement of Bone Suppression Task** • **Discover more way of PCA application**	• **Successfully Represent Image with 0.02% Size** • **Get No Improvement on PCA Decoder** • **Improve Performance and Efficiency with Autoencoder with PCA**
Procedures	**Analysis**
• **Design 2-dim and testify PCA Compression** • **Implement Whole Idea in PyTorch by Myself** • **Experiment on Purely PCA Decoder** • **Experiment on Autocencoder with PCA**	• **Combine ML and DL work on this task** • **The PCA extract main component and not suitable for too delicate purpose** • **Making Subproblem to its original task does help improve the**



2. Introduction & Motivation

During this semester, I got a hemothorax and was hospitalized for a few days. In order to check the physical condition of recovery after surgery, I took plenty of chest X-ray and discussed with my doctor. At that time, I found the boney structure would obscure the some region and make people harder to observe or diagnose the X-ray image. Since then, I become interested in how to optimize the x-ray image and remove the boney structure. At the same time, teacher happened to teach the autoencoder concept in class, which made me decide to find the research that combine these two concept. Therefore, I found the paper Chest X-Ray Bone Suppression for Improving Classification of Tuberculosis-Consistent Findings and decide to use it as my base idea model.

In the meanwhile, PCA, also a common method to compress data, was introduced in class. Therefore I them come up with the idea that what would happen if I replace one of the other or combine them together. By finding the approach, we might save the computation cost and improve the performance on the task. At the beginning, I utilize multi PCA (see 4.2) to get the 1d compression array, and try to generate the bone suppression image by taking it as the input of decoder, kind of like replacing the encoder part from original autoencoder. In the next stage, I keep the the whole auto encoder structure and add the PCA decoder. In this task, there would be two kind of inputs for one sample, the image and the PCA result of the image.

To complete the above task and make it easier to verify various subsequent experiments, I refer the idea of paper to implement it in PyTorch all by myself, and make the repository on my github (Jonathan-Chin328/bone_suppression). Also, I attach a colab notebook for user could run whole experiment easier.

3. Dataset

I use X-ray Bone Shadow Suppression from gaggle as the dataset. This dataset provide the X-ray image from same person in png with two format, with bone shadow and without bone shadow, which is good fit to my experiment goal. However, the grayscale way of the dataset is different. To better fit the real-world application, I first transfer the image color into the the more common form (Fig 1).

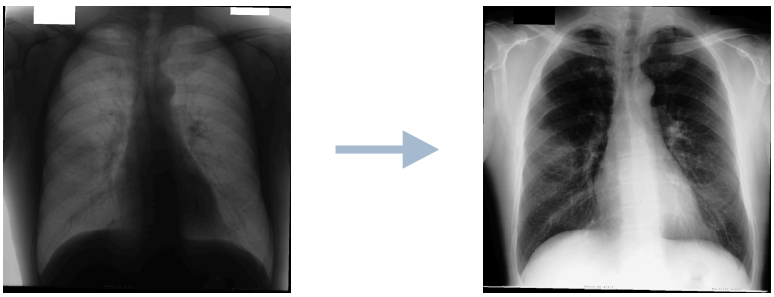


Fig 1. Convert the black-white in pixel and use equalize histogram to transfer the X-ray image into common format

4. Experiment

4.1. Autoencoder

An autoencoder is a type of artificial neural network (ANN) that is used to learn efficient data encoding in an unsupervised manner. It consists of two main parts: an encoder and a decoder. The encoder part of the autoencoder takes in an input data point and maps it to a lower-dimensional representation, and the decoder part of the auto-encoder then takes the latent representation and maps it back to the original input space. It is usually used in image compression, segmentation and unsupervised training. In the task of bone suppression, we want the encoder part extract the information of origin image and then decode it into the same image without boney structure.

After reading several papers in this task, I decide to use the idea from Chest X-Ray Bone Suppression for Improving Classification of Tuberculosis-Consistent Findings and implement the ResNe-BS model as my base architecture. And customize the PCA decoder part into the network (Fig 2).

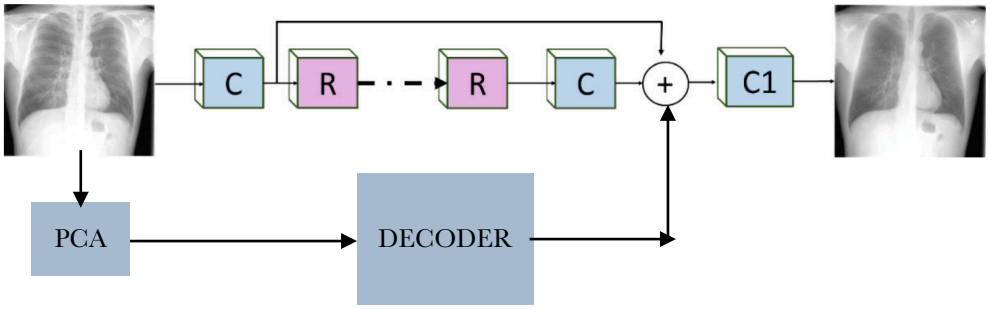


Fig 2

4.2. PCA Compression

PCA compression is a common way to compress image. By transforming the data to a new basis where the dimensions are low covariance and have high variance to reduce the dimensionality. Generally, we do the PCA in one dimension, ex. 1028*1028 to 1028*n_comp (Fig 3). However, to utilize the PCA result as the input of decoder, the dimension is still to hight . Therefore, I decide to transpose the first time PCA result array and redo the PCA in the other dimension, making it become a square matrix. In addition, observing the result of result of multi PCA, we can found that the matrix is close to diagonal matrix. Hence, I extract the diagonal part into a 1-dim array to present the main component as origin image, ex. 1028*n_comp to n_comp*n_comp (Fig 4). By doing so, we could transform an image from 1028*1028 to n_comp*1 and represent it.

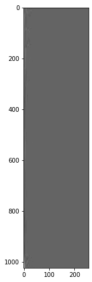


Fig 3

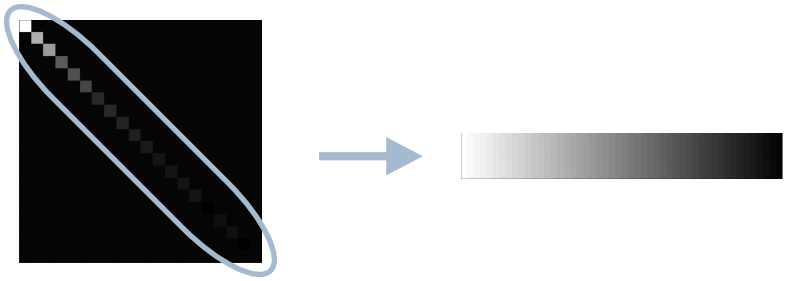


Fig 4

To testify the performance of this multi PCA, I reconstruct the image from the diagonal array and ensure it is workable (Fig 5 show the comparison of origin and reconstruction image). After verifying with it, we know that we could reconstruct the image from low dim array. The next question is, can we reconstruct the image but also suppress the boney structure at the same time? Due to the similarity between PCA and auto-encoder in compress the data, I take the diagonal array as the input of decoder, and combine the result of origin ResNet base auto-encoder to find out what will happen.

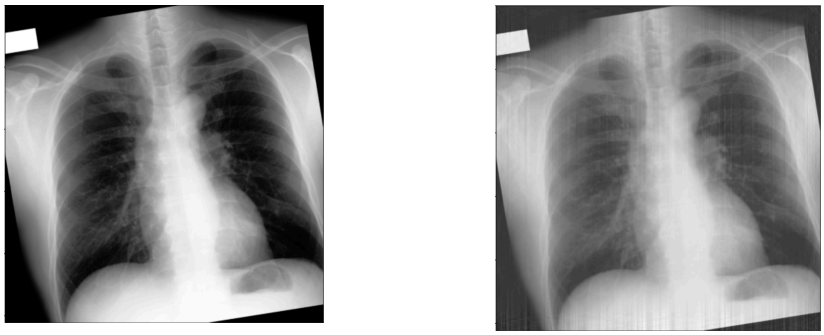


Fig 5. The comparison of original image (left) and reconstruct image (right), number of component in this example is 256 (n_comp=256)

4.3. MS-SSIM And MAE Loss

MAE (Mean Absolute Error) loss is a common metric used to evaluate the performance of a model in regression tasks, such as predicting a continuous numerical value. It measures the average absolute difference between the predicted values and the ground truth values. It is is a simple and easy-to-understand metric, but it can be sensitive to outliers, meaning that a few large errors can have a significant impact on the overall loss, and in sometime, the distance of MAE loss is very different from human intuitive feeling. Therefore, it introduce another loss function to deal with this problem.

MS-SSIM (Multi-Scale Structural Similarity) loss is a metric that measures the structural similarity between two images. Its loss compares the structural similarity between two images at multiple scales, taking into account the luminance, contrast, and structure of the images. The MS-SSIM loss is calculated by comparing the structural similarity of the two images at each scale and then taking the mean of these values. The MS-SSIM loss is generally considered to be a more reliable and robust metric for evaluating the quality of generated images compared to simpler metrics like the MAE loss. In the task, it use a combined loss function that benefits from both MAE and MS-SSIM [1], which $\Omega = 0.84$.

$$\text{Loss}_{\text{Combined}} = (\Omega \times \text{MS} - \text{SSIM}) + ((1 - \Omega) \times \text{MAE}) \quad (1)$$

5. Contribution

5.1. Compress Image Into Smaller Scale With Multi PCA

When compressing the image, most of the method that using PCA doing one time manipulation of it. In the case mention above, it successfully compress the image from 1028*1028 to 1028*256, which is 25% of the original size. By doing the multiple times of PCA and extract its diagonal part, we could ultimately turn the image from 1028*1028 to 256*1, which is only 0.02% of its original size, and still could get a good reconstruction from this only 256-dim array (Fig 4).

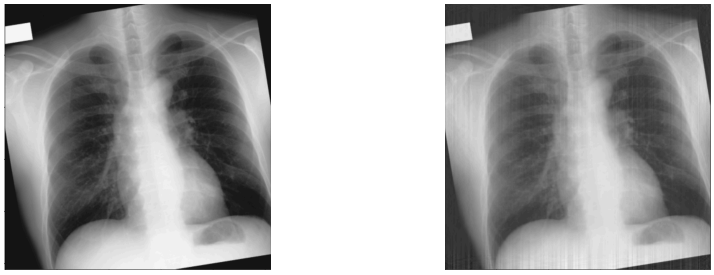


Fig 4. Reconstruction from one time pea (left, 25% of origin size) and reconstruction from multi PCA (right, 0.02% of origin size)

5.2. PCA Base Decoder

In the beginning, I tried to generate the bone suppression image directly from the PCA result with decoder. But after several experiments, I found it doesn't perform well (Fig 5,6) . I guessed there are two possible reasons of the result. The first is that it is kind of like a generative network, and this kind of network usually require more training time for its high complexity. However, due to the limit of hardware resource, I didn't have time and GPU to train that much iteration and test my thought. The second reason is that it's actually doesn't

work because PCA mainly extract the main component of the image, and the result of it is not detailed enough to capture the subtile place like bone structure or lung fiber. Hence, having the goal to generate a bone suppression image is too complicated for it. After the failure, I started thinking that if the PCA mainly get the main component of image, combined it and the origin autoencoder might be a good way to try.

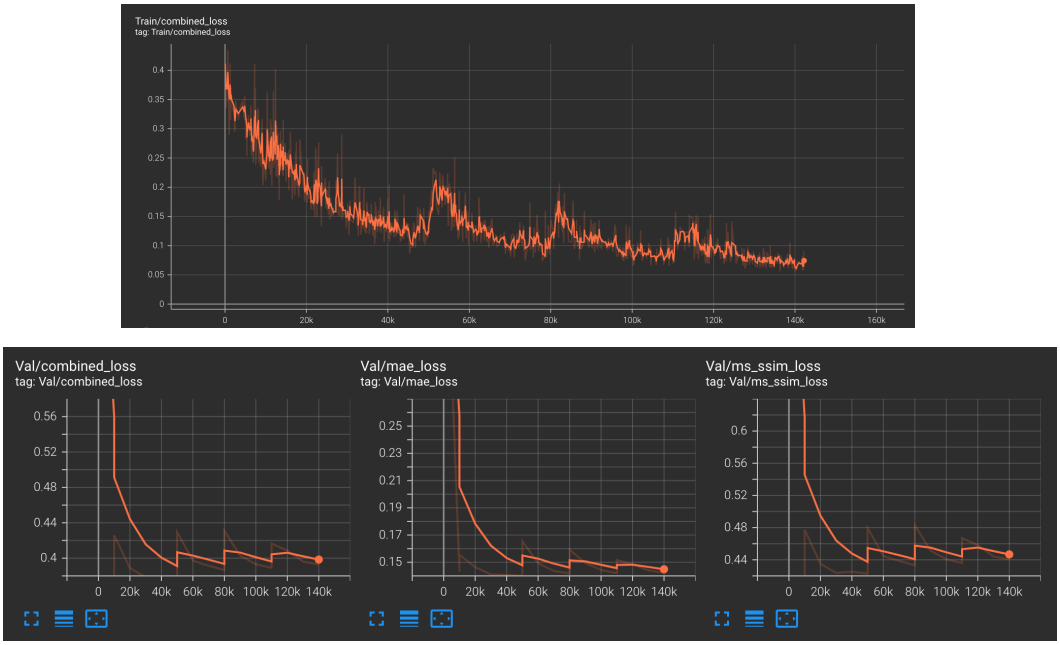


Fig 5,6 the training loss and validation loss of purely PCA with decoder.

5.3. Autoencoder with PCA Decoder

As mentioned in 4.1. above, after failure in purely PCA decoder, I decide to combined them together for these reason. First, due to focusing on the main component of the image, PCA could reconstruct the rough image of the goal. Therefore, when we combine them, we could expect the PCA part focus on the same part, which is most region in the image, and let the autoencdoer can focus on the boney structure part. From the results of experiment, we could get a well suppressed image and find each model structure devotes to a simpler subproblem (Fig 6), getting some improvement on the task.

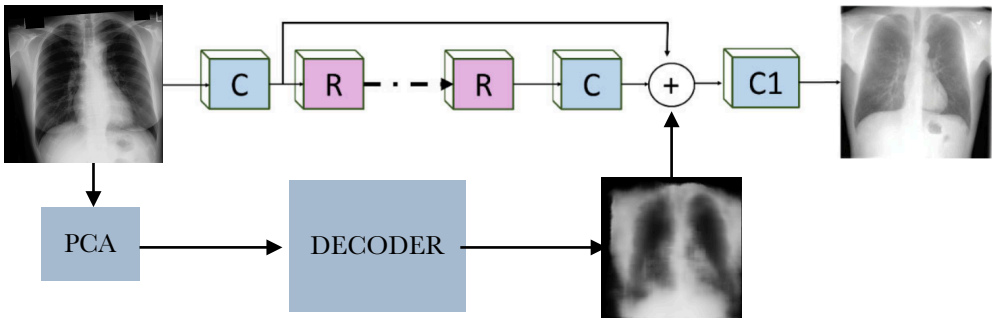


Fig 6. The illustration of output layer, hidden layer compared to original image when evaluation

As the realistic metric score, I compared the new structure with its origin autoencoder using the combined loss mentioned in 4.3. All the parameters such as adaptive learning rate strategy or random seed remain the same except for the different architecture adding the PCA decoder part. Since the resource limitation make me couldn't have same training iteration within two experiments. I capture the first 26000 iterations to make the comparison (Fig 7). We can found the loss of new model drop a little faster than the other one.

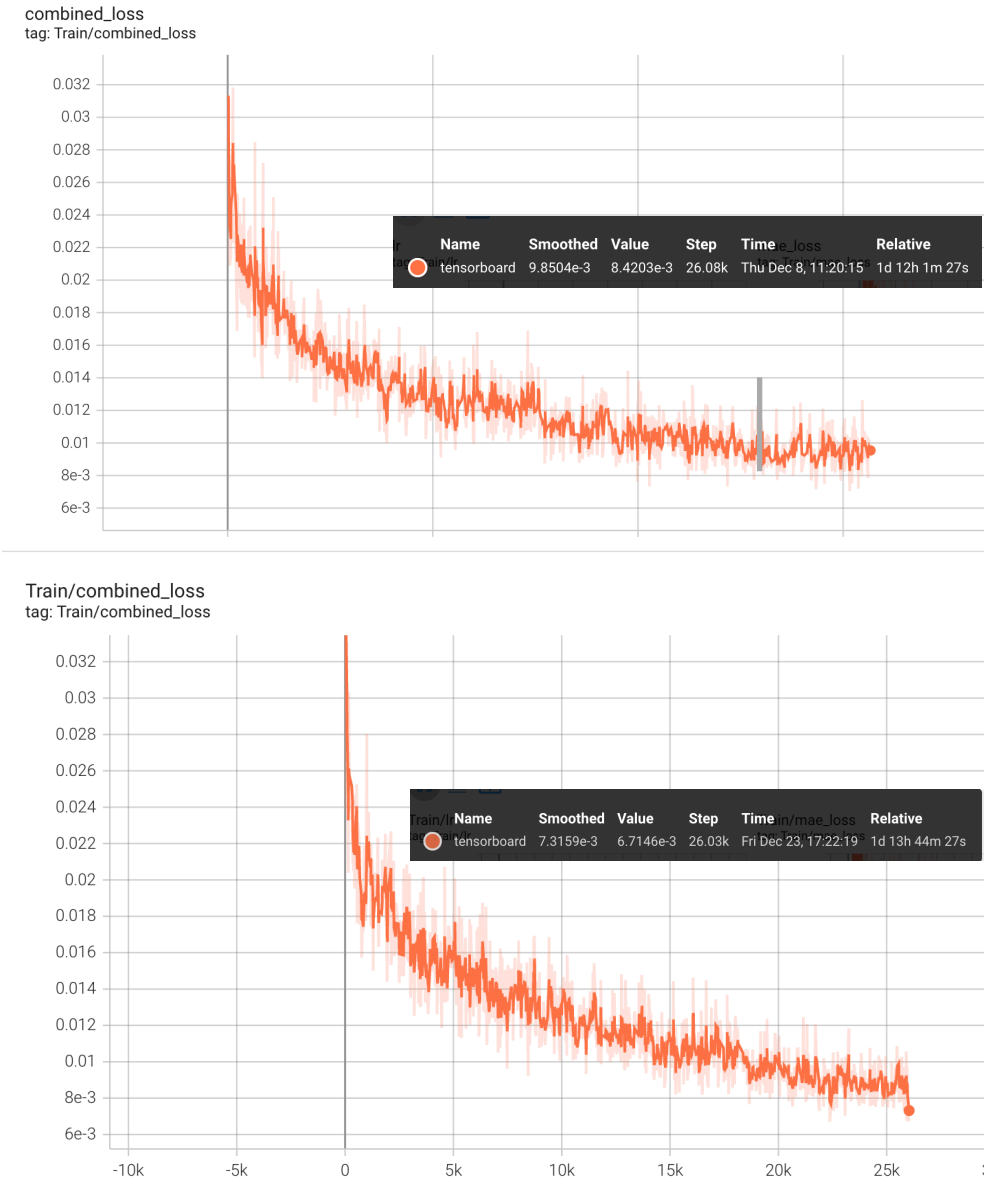


Fig 7. Original autoencoder (up) compare to res_pca version(down)

Extract the loss at both 26k iterations

6. Appendix

6.1. Python notebooks for others easier to reproduce this experiment

https://github.com/Jonathan-Chin328/bone_suppression/blob/main/final.ipynb

6.2. Validation loss

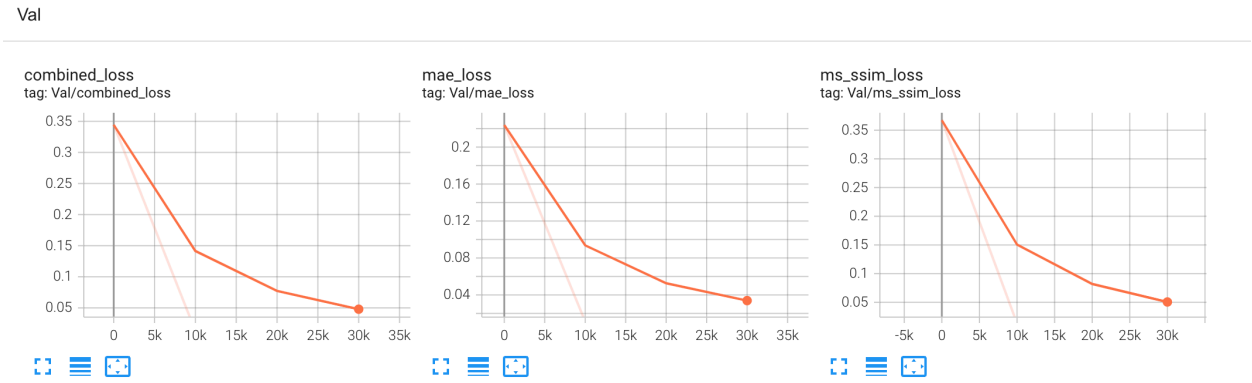


Fig 7. Val loss for original Autoencoder

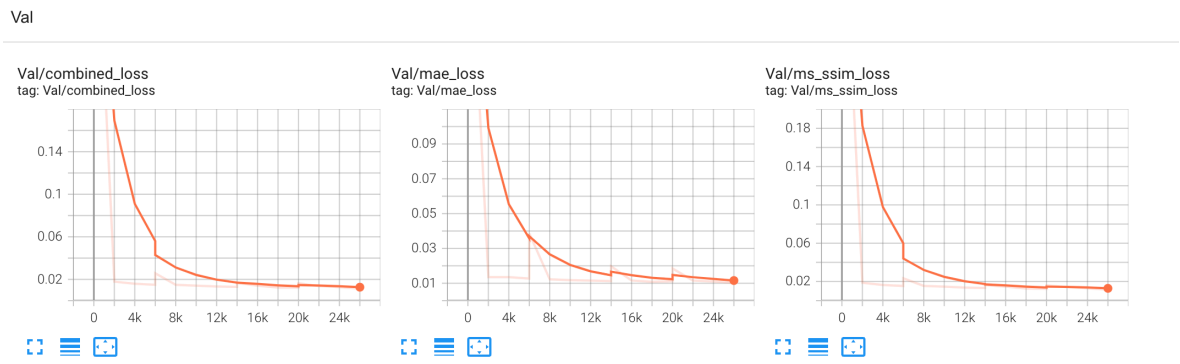


Fig 8. Val loss for Autoencoder with PCA