Task 1

Task 1a

Training on binary labels:

To do the binary training in the train and test method we had to modify the labels from a given multi-label target to a binary-label target. To do this we decided to count a sample as positive (1) if at least one disease was present, at least one of the labels was set to 1, and as negative otherwise. Then we concatenated the modified tensor with the original one to get the final result and set it to require a gradient so it comfort with the original tensor and to be able to do backpropagation in the training phase. The rest of the train and test code is equivalent to the previous methods we implemented and used in the last exercises. We believe that the accuracy is not calculated appropriately. To support this argument we calculated the true positive rate of the modified training samples and came to the conclusion that there is a strong class imbalance present in the given dataset. The accuracy is a performance measure known to be highly sensitive to data imbalance, as the model predicting the majority class will lead to a high accuracy. Because of the nature of the dataset we would suggest to consider a Balanced Accuracy or F1 Score.

Instead of using a fixed number of steps for training a deep learning model, it would be more effective to employ adaptive training techniques that adjust the training process dynamically based on the model's performance. Through not training for a set amount of steps and deploying dynamic techniques like “Early Stopping” and “Learning Rate Scheduler” we reduces the risk of overfitting. To select the best model using “smart techniques” we could monitor multiple metrics and after finishing the training select the model checkpoint that performs best across all metrics. We should always monitor the training and validation loss and halt the training when the validation loss starts to increase again or at the end select the model checkpoint with the lowest validation loss.

Task 1b

Adjust the Net class to return embeddings:

This task left us wondering at which point to best extract the embeddings. We decided on returning the embeddings before the input is fed into the linear layers. However, we also researched the problem and found other approaches returning the embeddings before the last Convolution layer. We are to this day not sure which approach is the best because the class separation visualized in the next part did not change with different embedding-return-location within the model.

Apply PCA and t-SNE to the embeddings:

For this part we just used the `sklearn.decomposition and sklearn.manifold` library to apply PCA and t-SNE respectively. We also transformed our test_labels into binary target to be able to plot the

two dimensionality reduction functions. We are not sure that the results are correct as the model does not seem to separate the classes, healthy/un-healthy, correctly.
Our hypothesis is that due to the limited data we trained the chest model on and the low accuracy, the model did not yet learn a valid class separation.

Task 1c

Initial Training with Limited Labels (300 samples):
The accuracy range between 0.45 and 0.5 suggests that the model might be struggling to learn meaningful patterns from a small labeled dataset. With such a limited amount of labeled data, the model may not have enough information to generalize well to unseen examples.
Autoencoder Training:
The autoencoder's good performance, as indicated by the Structural Similarity Index (SSIM) scores of about 0.88-0.90, suggests that the autoencoder is effective at capturing and reconstructing features within the dataset. Autoencoders are known for learning robust and compact representations of data, which can be beneficial for feature extraction.
Transfer of Autoencoder Weights:
Transferring weights from the well-trained autoencoder to the classification model is a common technique to leverage the learned representations. However, the improvement in accuracy from 0.5 to 0.55 after transferring weights suggests a moderate enhancement. It's worth noting that the improvement might be influenced by factors such as architecture compatibility and the quality of the learned features.
Possible Explanations for Limited Improvement:
The original model may still be limited by the small labeled dataset, and the autoencoder may not provide sufficient additional information to overcome this limitation. The architecture of the original model might not be fully compatible with the features learned by the autoencoder, leading to suboptimal integration of the pre-trained weights. Therefore, to experiment with different architectures for both the initial model and the autoencoder could help finding a better combination that complements each other.

In conclusion, while the results indicate some improvement, there may be room for further optimization and experimentation. It's crucial to iterate on the model, consider alternative architectures, and potentially explore additional techniques to enhance the performance of this semi-supervised learning approach.

Task 2

Transfer learning is a technique in machine learning where a model trained on one task is used to help solve a different but related task. Our aim is to increase the success rate of the model on the limited PneumoniaMNIST dataset by using Transfer learning.

2a Transfer Learning - Freezing Networks

In this step, after training the chest model, with a for loop, we transferred the state dictionary of the net class except for linear layers or classification layers. Then we made all parameters not require grading except for the newly initialized linear layers and classification layer.

- Freezing the encoder and analyzing the latent space allows researchers to understand how well the model is able to separate different classes or conditions. It helps verify that the transferred features are useful for the new task and that the model is learning a meaningful representation.
- Freezing layers is often done to retain the knowledge learned in pre-training. Lower layers in a neural network tend to capture general features, while higher layers may be more task-specific. By freezing lower layers, you keep these general features intact.

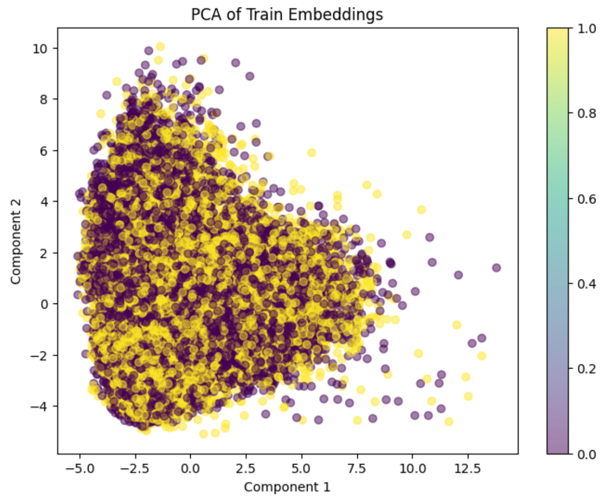
2b Transfer Learning - Trainable Networks

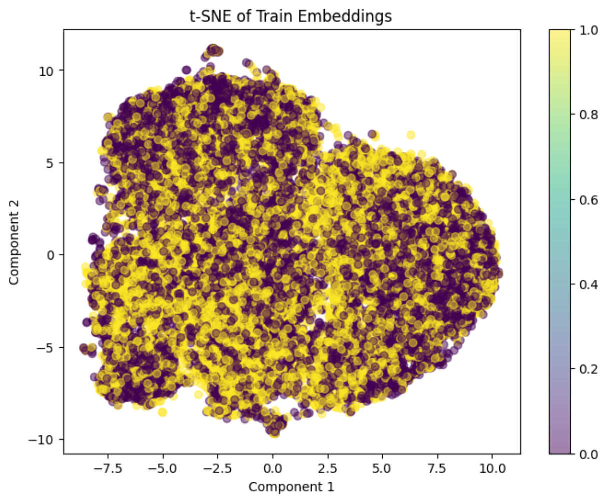
In this step, we transferred all layers from the chest model to the pneumonia model and kept all layers trainable.

- Leaving the weights trainable worked better, because allowing all layers to be trainable enables the model to adjust not only the fully connected head but also the features learned in the encoder layers to better suit the specifics of the pneumonia detection task. However, sometimes this can lead to overfitting, so this is not true all the time.

2c Latent Space Musings

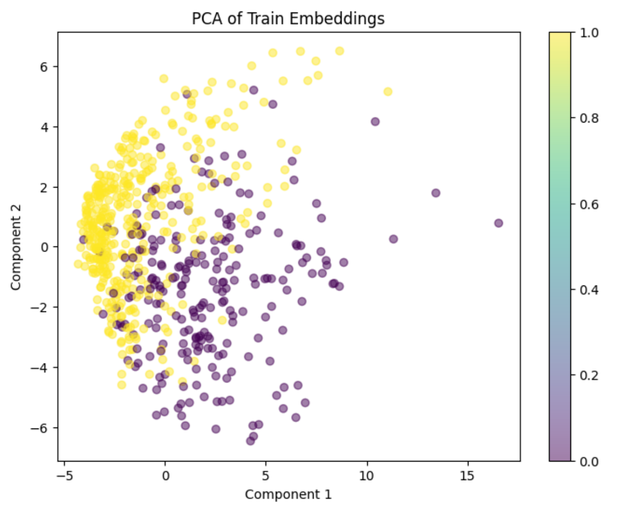
Task 1 Latent Space Figures:

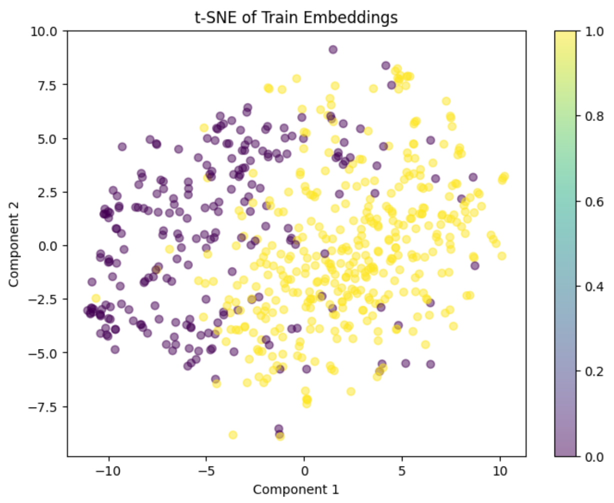




Task 1 was challenging, and the net class was not capable of separating between the 2 classes; it is very obvious in the plots of the latent space and was also verified from the accuracy obtained 50% which was totally chance level.

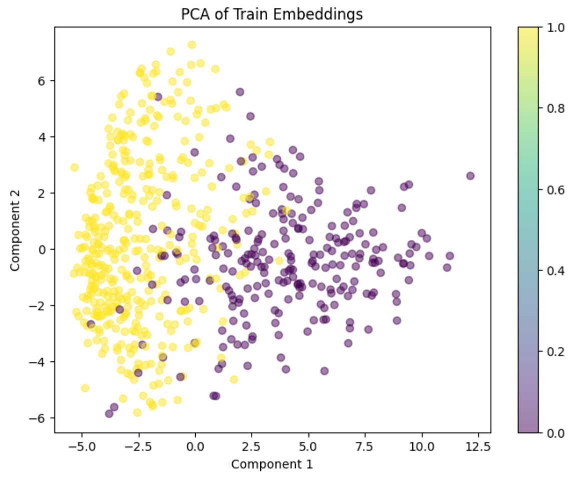
Task 2a Latent space figures:

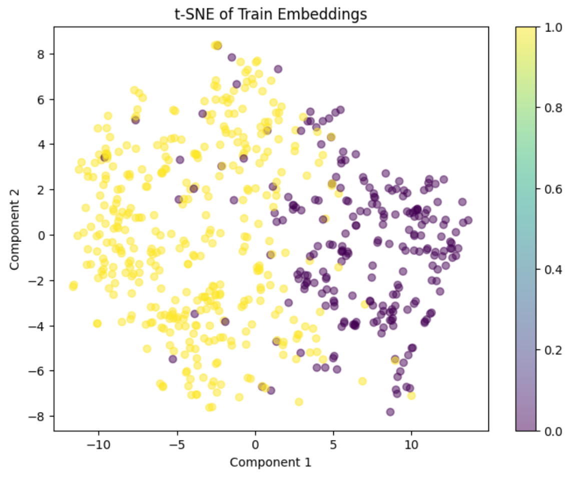




The latent space looks separable to an extent, still, some embeddings are challenging, and the data points still overlap in some areas. However, it looks like the trained encoder and its weights on the chest model were capable of encoding the PneumoniaMNIST dataset without retraining them on the data and that the transferred features were actually useful.

Task 2b Latent space figures:





The latent space for task 2b looks better than task 2a; the embeddings are more separable, which means the task has become easier and clearer and that the model is trained better for the pneumonia task.

Bonus

Yes, this is a common problem, as the newly initialized non-trained head can heavily affect the pre-trained layers so that the weights of these layers change significantly because of the large error caused by the new head, causing what we call forgetting the pre-trained features.
This can be solved with multiple solutions:

1. **Linear probing and then finetuning**: Keep the encoder frozen while training the head only and after achieving a convergent solution with the linear head. Unfreeze the encoder then finetune the whole model.
2. **Gradually unfreezing:** This is very similar to the approach above, except that instead of finetuning the whole encoder in one shot, gradually or progressively unfreeze layers from the encoder and then add these layers to the finetuning pipeline.
3. **Layer-wise learning rate:** Use different learning rates for different layers. Lower learning rates are often used for layers closer to the input, and higher rates for layers closer to the output. This can help stabilize training and prevent forgetting the pre-trained features.