BioE 145/245 Final Report Spring 2021

Sina Ghandian
Collaborator: Kendall Kikkawa
{sghandian, kendall.kikkawa}@berkeley.edu

May 14, 2021

*Note: Sections marked with * were written **individually**.

1 Introduction

1.1 Context

We have discussed many applications of single cell RNA-seq and the computational challenges that arise from this complex data. In this project, we will work directly with scRNA-seq data in a real-world application. Peripheral blood mononuclear cells (PBMCs) are a group of different cell types in your blood, a subset of what we call white blood cells. They include many important parts of your immune system: T cells, B cells, natural killer cells, and so on. These cells are responsible for your innate and adaptive immune responses. When you get infected or vaccinated, they kick into high gear. In different situations, or with different diseases, the mix of different cell types shifts and so does the gene expression within one type of cell. PMBCs can be isolated very easily from patient blood samples, just by spinning the blood in a centrifuge that separates different cell types by weight. So, looking at these cells can be very informative.

1.2 Specific Approach*

In this experiment, we explore whether it is possible to classify different subtypes of PBMCs through machine learning methods as applied to a high-dimensional representation of our data. We begin by visualizing our data in lower dimensional space through PCA, t-SNE, or a combination of these techniques with an auto-encoded representation. Afterwards, we classify each point by training a number of statistical and neural network models and analyze the results.

2 Lower dimension representation of the cells

Starting from scRNA-seq data from PBMCs, we look at the cells in a lower-dimension space. We implement an autoencoder to find a latent space representation of the data.

2.1 Autoencoder implementation*

The autoencoder was constructed using the Keras Layer module, with each layer utilizing ReLU activation functions, as our problem is one of regression. We defined an input layer equivalent to the dimensionality of our feature matrix, then formulated a funnel-like architecture progressively decreasing

the hidden state size from one layer to the next until reaching a latent space with dimensionality of 32. We save this and a latent space of size 64 for downstream analysis. The decoding section of the architecture is then constructed as a reflection of the encoding scheme, so that the model's output can be compared to the input and so that it can be trained, optimizing for MSE loss. Our final MSE loss of 0.65 was achieved through regularization of two layers within the model, and after training for 500 epochs.

2.2 Visualization of lower dimension representations

We then compare two-dimensional representations of our data using t-SNE, PCA, and the latent space defined by the autoencoder. Because the proposed latent space outputs of our autoencoder are larger than two dimensions, we apply both PCA and t-SNE to the encoder outputs for comparable visual representations. For all cases of t-SNE embeddings, we use a perplexity of 100. Figure 1 and Figure 2 display the results of PCA and t-SNE applied to the initial dataset. Figure 3 and Figure 4 display the results of PCA and t-SNE applied to the latent space representation.

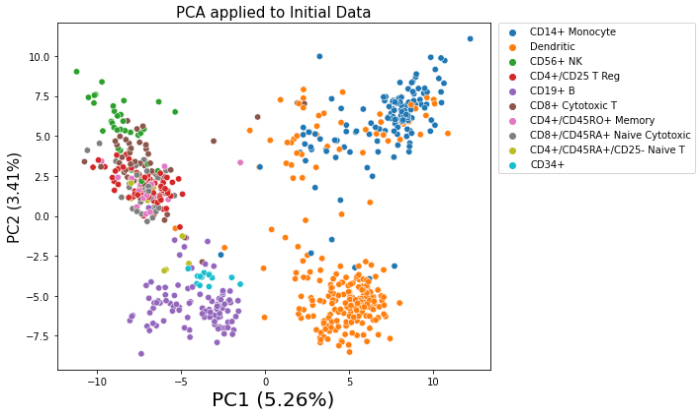

Figure 1: PCA applied to original dataset

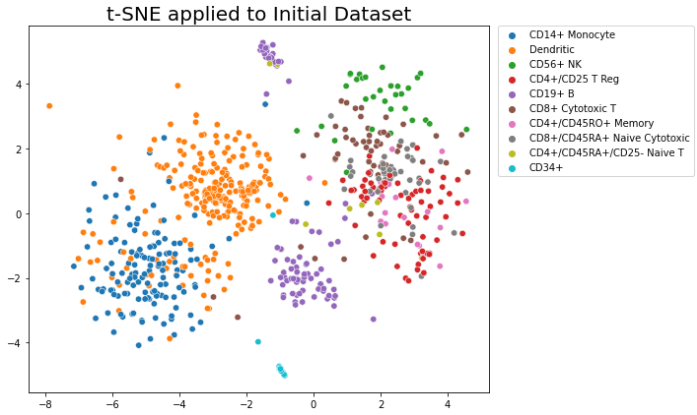

Figure 2: t-SNE applied to original dataset

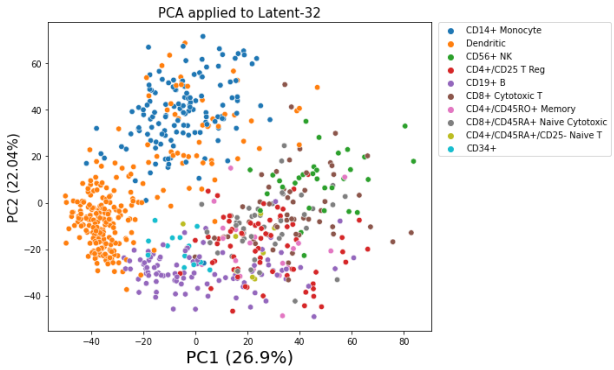

Figure 3: PCA applied to Latent-32 Space

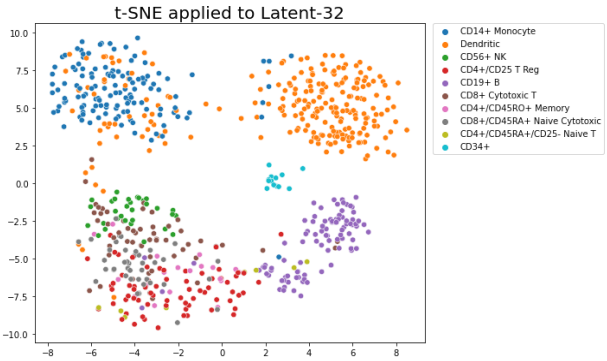

Figure 4: t-SNE applied to Latent-32 Space

2.3 Comparison and Analysis of Dimensionality Reduction Techniques*

By our analysis, it seems that t-SNE on our latent space of size 32 performed best. Comparing PCA and t-SNE on our initial data, we see that both primarily struggle with separating CD4 and CD8 subtypes. t-SNE is better able to separate CD56 cells in both the initial and latent space. PCA in the latent space performs about the same as PCA in the initial space, if not worse due to additional noise added by the encoded scheme. Finally, t-SNE in the latent space performs at least as well as it did in the high dimensional space, but can additionally group together CD19 subgroups and separate out the CD4 and CD8 subtypes more effectively than PCA. One could draw a less complex, albeit still nonlinear, decision boundary in this space to capture most groups without misprediction. Therefore, as a clustering and visualization technique, t-SNE with the latent-32 representation performs the best.

3 Classification

In this section, we implement a classifier to classify unlabeled cells as one of the cell types labeled in the original dataset.

3.1 Descriptions of approaches/models surveyed for this task*

We chose to approach this classification task by comparing a number of models, cross-validating each in order to choose best hyperparameters via the *Optuna* library, and comparing their validation accuracies. We followed a standard 80:20 train:test split and validated using a subset of the training data. The models we tried were: logistic regression, support vector machine, random forest, FFNN, and adaboost applied to both decision trees and logistic regression. In order to train the statistical models, we flatten our one-hot encoded data by taking the argmax of each row, resulting in a single vector input. In addition, we understood that there are few sample points with respect to the dimensionality of the matrix, so we chose to train our neural network with duplicated training points by oversampling the training set.

3.2 Results*

Out of the box, we found sklearn's logistic regression to perform quite well out of the chosen models at a validation accuracy of 0.878. After utilizing Optuna to tune the statistical models' hyperparameters, these are the results we achieved. We did not utilize Optuna for the neural network. Only the FFNN with duplicated data could achieve a higher accuracy than the logistic regression.

Model	**Accuracy**
Logistic Regression	87.86
Support Vector Machine	87.14
Random Forest	82.14
AdaBoost Logistic Regression	82.86

Feedforward Neural Network	87.86
Feedforward Neural Network with 2x data	**88.57**

Figure 5 and Figure 6 display the training loss and accuracy of our final network, respectively.

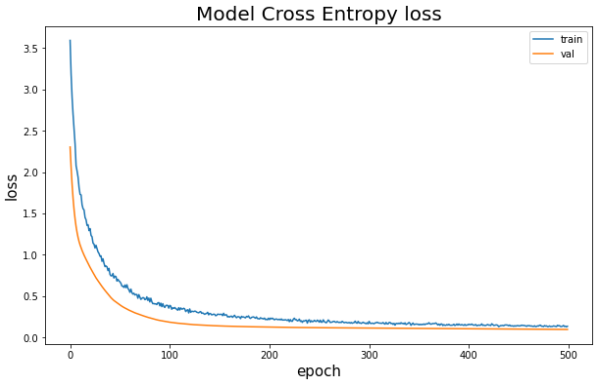

Figure 5: Epochs vs Loss for feed-forward network

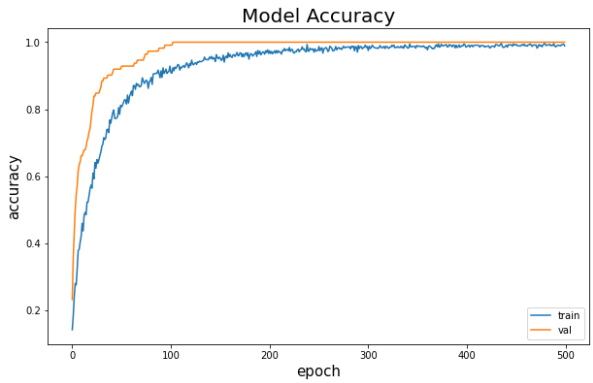

Figure 6: Epochs vs Accuracy for feed-forward network

3.3 Analysis*

A major limitation of our dataset was the scarcity of training examples. Since there were only ~500 training points available after splitting our data, we suspected that increasing the number of points would allow a more complex model to fit the data more accurately. While there are techniques for generating synthetic data such as SMOTE, we ultimately opted for a simple duplication of training points and evaluated the results. The FFNN that utilized this data achieved the highest accuracy, and thus we're inclined to believe that a neural network would have likely achieved the best performance if more data were available. Class imbalance also played a role in our models' performance, as seen below.

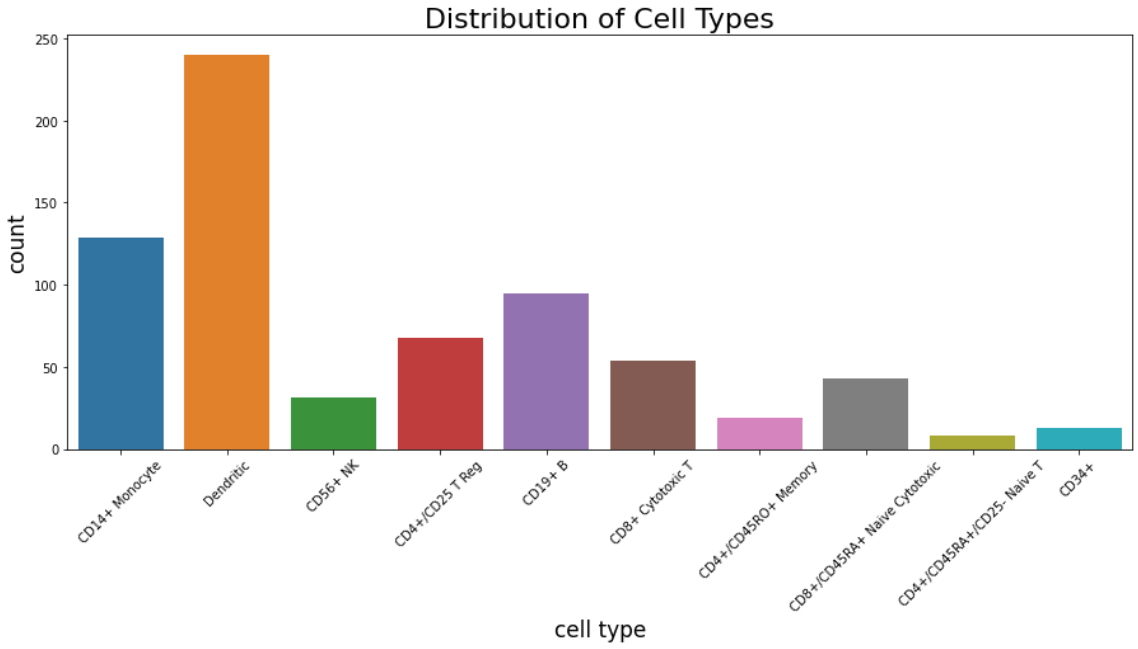

Figure 7: Cell Type Distribution

Some classes had only a few training instances overall, and so most models generally discarded them from its predictions for the sake of generalization. Rare classes such as CD4+/CD45RA+/CD25-Naive T did not include even a single correct classification.

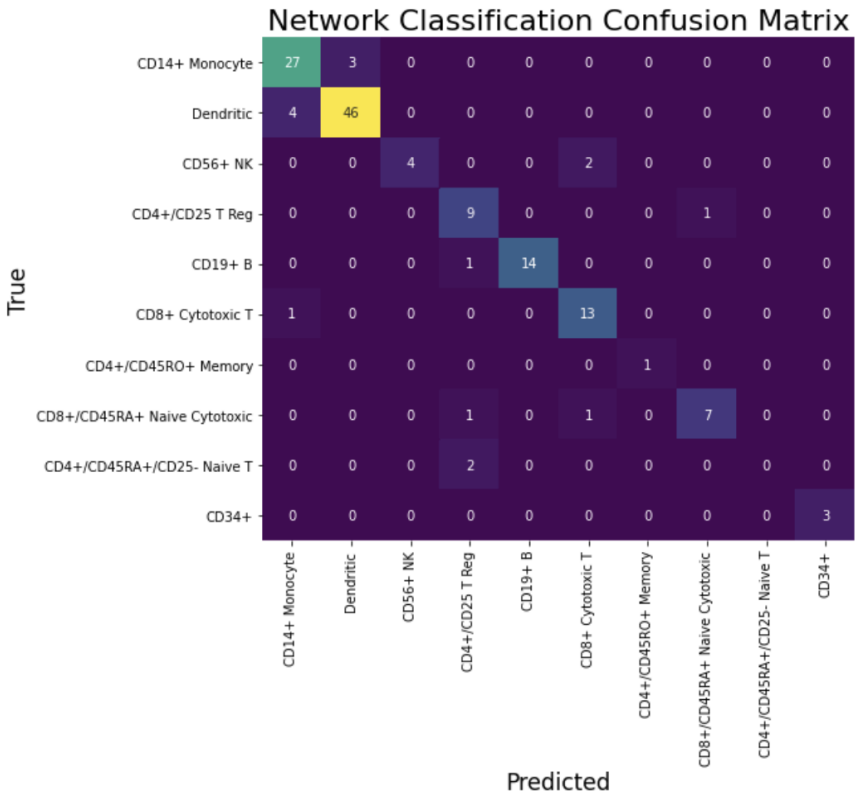

Figure 8: FFNN Confusion Matrix - Test Set

4 Conclusion

By our analysis, classifying these cell types according to the provided features is not only possible, but able to be done with relatively high accuracy. Performance gains will primarily be seen by collecting additional data, then training a neural network.

5 Appendix

The code for Part 1 (Lower dimension representation of the cells) can be found in this notebook
The code for part 2 (Classification) can be found in this notebook