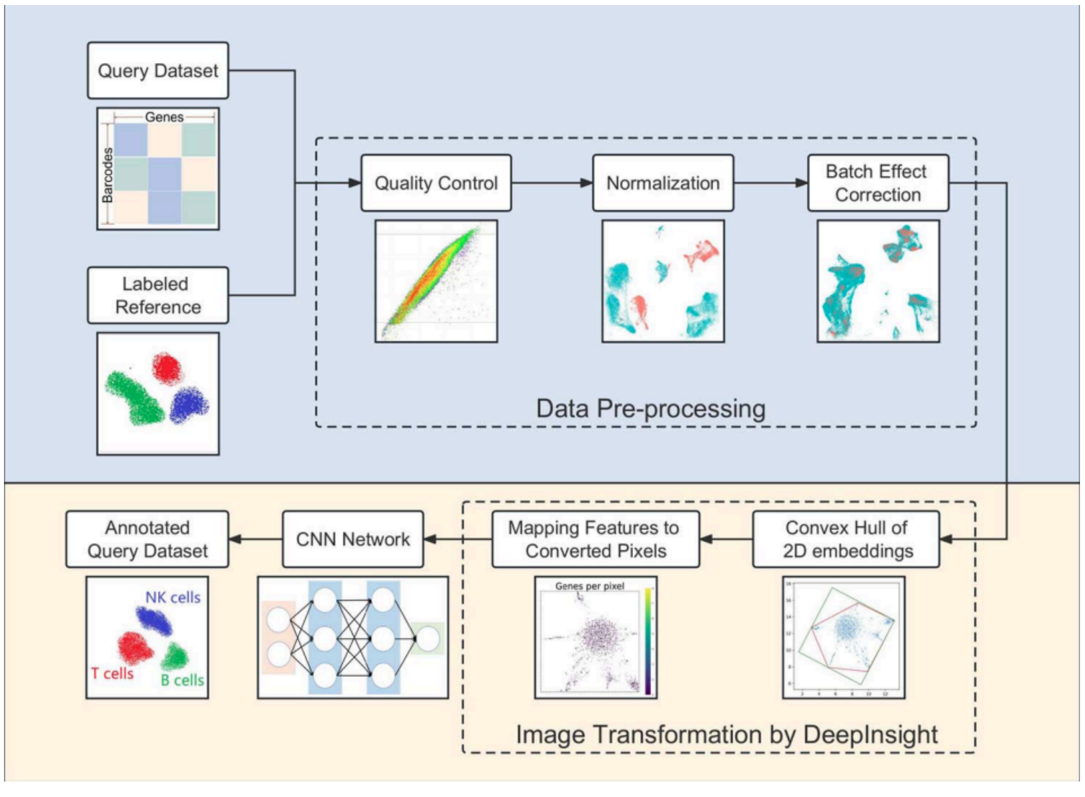
Cell Type Classification Using scDeepInsight on scRNA-seq Data from McKellar and Oprescu Datasets

Can Sahin
300185093
April 2025

Abstract

In this project, I used scDeepInsight, a supervised deep learning method, to classify cell types from single-cell gene expression data. I worked with two datasets, McKellar and Oprescu, both containing tens of thousands of annotated cells. The workflow involved preprocessing the data, transforming gene expression profiles into 2D images, and training a convolutional neural network (CNN) for classification. The best results were achieved when training and testing on the same dataset, with 94.7% accuracy on McKellar. However, accuracy dropped when the training and test datasets were preprocessed independently, and improved significantly when consistent preprocessing was applied, highlighting the importance of consistent preprocessing alignment, even in the absence of batch correction.



Introduction

Gene expression reflects how actively each gene is being used in a cell, shaping the cell's role and behavior. Scientists can measure this activity through RNA sequencing, which captures the levels of RNA produced by different genes.

Single-cell RNA sequencing (scRNA-seq) builds on this by capturing gene expression from individual cells, rather than averaging across a population. This makes it possible to see the unique characteristics of each cell and uncover the wide variety of cell types within a tissue.

Knowing which type of cell produced a given expression profile is key to understanding how tissues function, how they change over time, and how they respond to disease. For example, identifying certain immune cells in a tumor can help guide treatment decisions.

But classifying cell types from scRNA-seq data is not easy. The data is high-dimensional, with thousands of genes measured per cell, and some cell types are nearly indistinguishable based on their expression alone. Manual labeling is time-consuming, subjective, and not feasible for large datasets. Technical differences between experiments, known as batch effects, can also interfere with accurate classification.

To address these challenges, I used **scDeepInsight**, a Python-based deep learning pipeline. It transforms gene expression data into 2D images, allowing a convolutional neural network (CNN) to learn visual patterns and classify cell types. This approach brings the strengths of image-based learning into biological analysis and offers a new way to work with complex gene expression data.

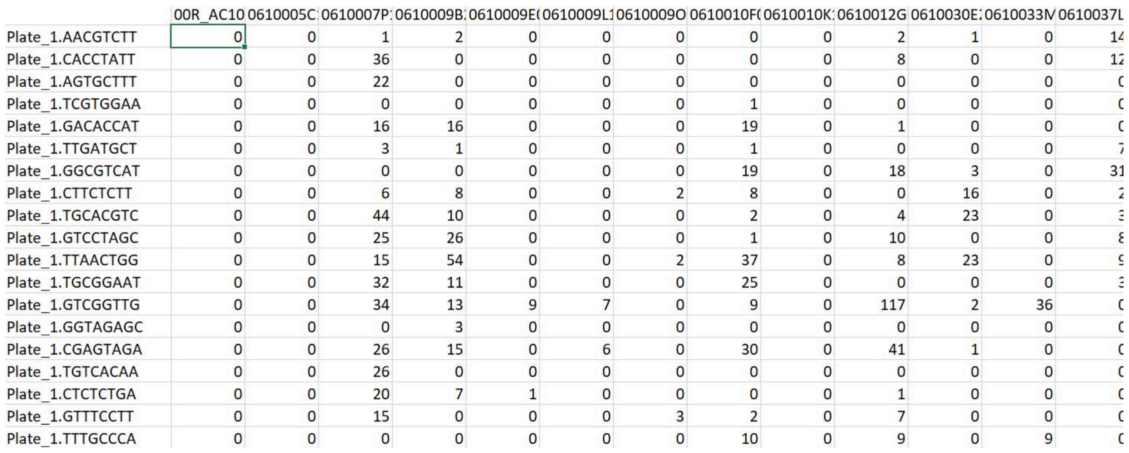
Background

A **Convolutional Neural Network (CNN)** is a type of deep learning model commonly used for image classification. CNNs are designed to detect spatial patterns in image data, such as edges, textures, or shapes, and build hierarchical representations that allow them to recognize complex visual features. Because of this, CNNs are well-suited for learning from the kinds of images produced during gene expression-to-image transformation in scDeepInsight.

scDeepInsight is an algorithm that converts non-image data, like gene expression values, into images, enabling CNNs to process them. The process starts by embedding genes into a 2D layout using dimensionality reduction methods like **t-SNE**. This layout remains consistent for all cells. Then, for each cell, its expression values are mapped onto this 2D grid to form an image where pixel brightness reflects gene activity. These images are then used to train a CNN for classification. The entire model is implemented in **Python using PyTorch**.

In this project, I worked with **two key data components**:

- The **expression matrix**, which contains the measured gene expression levels for each gene in every cell. (Below is an example for the expression matrix.)



	00R_AC10	0610005C	0610007P	0610009B	0610009E	0610009L1	0610009O	0610010F	0610010K	0610012G	0610030E	0610033M	0610037L
Plate_1.AACGTCTT	0	0	1	2	0	0	0	0	0	2	1	0	14
Plate_1.CACCTATT	0	0	36	0	0	0	0	0	0	8	0	0	12
Plate_1.AGTGCTTT	0	0	22	0	0	0	0	0	0	0	0	0	0
Plate_1.TCGTGGAA	0	0	0	0	0	0	0	1	0	0	0	0	0
Plate_1.GACACCAT	0	0	16	16	0	0	0	19	0	1	0	0	0
Plate_1.TTGATGCT	0	0	3	1	0	0	0	1	0	0	0	0	7
Plate_1.GGCGTCAT	0	0	0	0	0	0	0	19	0	18	3	0	31
Plate_1.CTTCTCTT	0	0	6	8	0	0	2	8	0	0	16	0	2
Plate_1.TGCACGTC	0	0	44	10	0	0	0	2	0	4	23	0	3
Plate_1.GTCCTAGC	0	0	25	26	0	0	0	1	0	10	0	0	8
Plate_1.TTAACTGG	0	0	15	54	0	0	2	37	0	8	23	0	9
Plate_1.TGCGGAAT	0	0	32	11	0	0	0	25	0	0	0	0	3
Plate_1.GTCGGTTG	0	0	34	13	9	7	0	9	0	117	2	36	0
Plate_1.GGTAGAGC	0	0	0	3	0	0	0	0	0	0	0	0	0
Plate_1.CGAGTAGA	0	0	26	15	0	6	0	30	0	41	1	0	0
Plate_1.TGTCACAA	0	0	26	0	0	0	0	0	0	0	0	0	0
Plate_1.CTCTCTGA	0	0	20	7	1	0	0	0	0	1	0	0	0
Plate_1.GTTTCCTT	0	0	15	0	0	0	3	2	0	7	0	0	0
Plate_1.TTTGCCCA	0	0	0	0	0	0	0	10	0	9	0	9	0

- The **metadata** includes labels like cell type and other annotations associated with each cell.

Methods

Datasets Used

The data used in this project came from a **Seurat object** that combined single-cell RNA-seq datasets from roughly 20 different studies. I did my experiments using two of those datasets

- **McKellar (Walter et al., bioRxiv, 2020)** – ~89,000 cells

- **Oprescu (Oprescu et al., iScience, 2020)** – ~47,000 cell

These are the included cell types in each dataset:

- **T cells** – coordinate immune responses and help eliminate infected or abnormal cells
- **B cells** – produce antibodies and help fight infection
- **FAPs (Fibro-Adipogenic Progenitors)** – contribute to tissue regeneration and fibrosis
- **MuSCs (Muscle Stem Cells)** – regenerate muscle tissue after injury
- **Myoblasts** – developing cells that eventually become muscle fibers
- **Myonuclei** – parts of muscle fibers that help control muscle functions
- **Neutrophils** – fast-acting immune cells that respond to infections
- **Monocytes** – immune cells that circulate in the blood and become macrophages or dendritic cells
- **Dendritic cells** – initiate immune responses by presenting antigens to T cells
- **Macrophages** – immune cells that digest germs, damaged cells, and debris
- **NK cells (Natural Killer cells)** – destroy virus-infected and tumor cells
- **Endothelial cells** – line the inside of blood vessels and help control what passes in and out
- **Smooth muscle cells** – help organs and blood vessels contract and function automatically
- **Tenocytes** – maintain and repair tendon tissue
- **Neural cells** – carry signals in the brain, spinal cord, and nerves

Using these, I performed four main experiments to measure how well the scDeepInsight model generalizes across datasets:

1. **McKellar → McKellar**: trained and tested using only McKellar data

2. **Oprescu → Oprescu**: trained and tested using only Oprescu data (1 test with 3000 HVG* and 1 test with 2000 HVG)

3. **McKellar → Oprescu**: trained on McKellar, tested on Oprescu

4. **Oprescu → McKellar**: trained on Oprescu, tested on McKellar

I did 2 separate tests for experiments 3 and 4 using aligned and unaligned test images.

*HVG = Highly Variable Genes

- *Unaligned test images: Test data was transformed using a preprocessing pipeline fitted on its corresponding training data, rather than the training data of the model.*
- *Aligned test images: Test data was transformed using the same fitted preprocessing as the training data (e.g., HVG selection, scaling, and layout).*

R to Python Conversion

The original datasets were provided in .RDA format as R objects. To use them in the Python-based scDeepInsight pipeline, I had to convert these R objects into compatible Python formats.

At first, I used localconverter() from the rpy2 package, which caused subtle issues in the metadata structure, leading to label mismatches during evaluation. After some debugging, I switched to directly using:

```
pandas2ri.rpy2py()
```

This produced a proper pandas.DataFrame with clean row and column formatting. I also converted the sparse expression matrix (dgCMatrix from R) into a SciPy sparse matrix for compatibility with AnnData.

The original code from the researchers splits a single dataset into 90% training, 5% validation, and 5% test, all within one reference.h5ad file. In my case, I created two separate files from the beginning: one for training (reference.h5ad) and one for testing (test.h5ad). This approach lets me work with completely distinct training and test sets drawn from different sources.

Preprocessing

After converting the R-based training and testing datasets to Python, I performed several preprocessing steps to prepare the gene expression data for image transformation:

- **Filtered out genes** with low expression
- **Selected** the top **3,000** highly variable genes
- **Normalized** the counts per cell
- **Log-transformed** the values (`log1p`)
- **Scaled** all values to the **[0, 1]** range using `MinMaxScaler`

Image Transformation

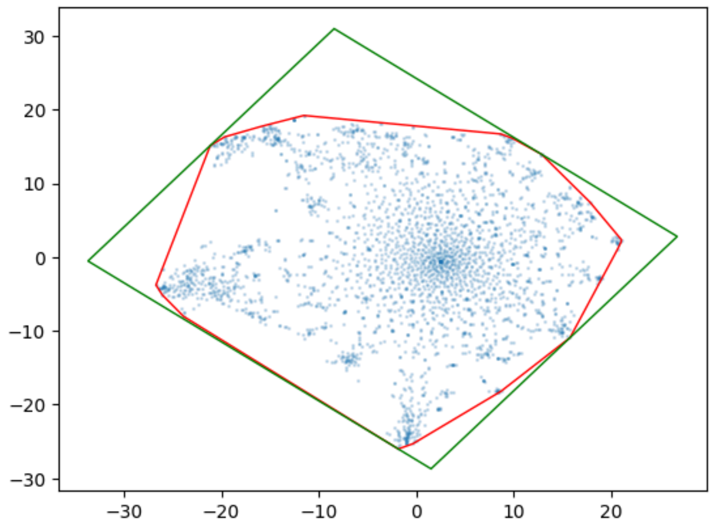
To prepare the gene expression data for CNN training, I used the ScDeepInsight image transformation pipeline. The steps were as follows:

Step 1: Train a t-SNE Layout on the Training Expression Matrix

I first trained a t-SNE layout using the training expression data, excluding labels. The t-SNE algorithm arranges genes in a 2D space based on similarity, producing a consistent spatial grid.

```
tsne = TSNE(n_components=2, perplexity=50, metric='euclidean', random_state=0,
n_jobs=-1, learning_rate="auto", init='pca')
it = ImageTransformer(feature_extractor=tsne, pixels=224)
it.fit(train_rna.iloc[:, :-1], plot=True)
```

Below is the t-SNE embedding that arranges genes in a 2D space. The green outline represents the full convex hull, while the red line shows the boundary where pixel mapping is performed.



Step 2: Save the transformer

This trained layout was saved to a file so the exact mapping could be reused for validation and test data.

```
with open("img_transformer_pre.obj", "wb") as f:
    pickle.dump(it, f)
```

Step 3: Convert training data into images

To avoid memory issues, the training data was split into four equal parts. Each chunk was transformed into grayscale images using the trained transformer, and then concatenated into one large array.

```
rna_list = np.array_split(train_rna, 4)

img_1 = (it.transform(rna_list[0].drop(columns=["celltype_l2"]).values) * 255).astype(np.uint8)
img_2 = (it.transform(rna_list[1].drop(columns=["celltype_l2"]).values) * 255).astype(np.uint8)
img_3 = (it.transform(rna_list[2].drop(columns=["celltype_l2"]).values) * 255).astype(np.uint8)
img_4 = (it.transform(rna_list[3].drop(columns=["celltype_l2"]).values) * 255).astype(np.uint8)

train_img = np.concatenate((img_1, img_2, img_3, img_4), axis=0)
```

Step 4: Saving Image Arrays

Once the training and validation images were generated, I saved them in NumPy `.npy` format. This allows for efficient loading during model training without having to repeat the transformation process.

Step 5: Generating and Saving Test Images

To complete the dataset preparation, I also generated images from the test data using the same trained transformer. This ensures that all input images (train, validation, and test) share a consistent spatial mapping of genes.

Step 6: Output Verification

To verify that the transformation process worked as intended, I visualized a few of the resulting training images from both datasets. These grayscale images represent the expression profile of a single cell, where each white dot corresponds to a gene with high expression intensity at that pixel location.

Sample training image from Mckeller

Sample training image from Oprescu

Model Training

The training pipeline was adapted from the official scDeepInsight implementation, which provided the baseline structure and model code. I made several modifications to tailor it to my local environment and experimental goals.

I used an **EfficientNet-B3** architecture implemented with PyTorch, along with a training loop that incorporated label encoding, label smoothing, adaptive learning rate scheduling, and early stopping. The model was trained on an Apple M1 Pro GPU using the **MPS backend**, which required adjustments such as reducing the batch size and total epochs.

Training Pipeline Overview:

- **Architecture:** EfficientNet-B3 with a modified output layer
- **Loss Function:** Cross-Entropy with Label Smoothing ($\varepsilon = 0.1$)
- **Optimizer:** NAdam (learning rate = 3e-4)
- **Scheduler:** ReduceLROnPlateau
- **Early Stopping:** Patience = 30 epochs
- **Training Epochs:** Capped at 30 (vs. 100 in the original code)
- **Batch Size:** 32 (reduced for compatibility with MPS)
- **Crash Recovery:** Added a checkpointing method to save the model after each epoch
- **Training Duration:** Approximately 6 hours per experiment
- **Label Handling:** Used `LabelEncoder` from scikit-learn, saved and reused consistently across experiments

The model was trained separately for each experimental setup (e.g., McKellar → McKellar, Oprescu → McKellar), using the corresponding training and validation image arrays for each case.

Evaluation

I used a combination of standard classification metrics and visualizations to assess the performance of the trained models. These evaluation tools were applied consistently across all four experimental setups.

- **Classification metrics** included:
    - Overall Accuracy
    - Precision
    - Recall
    - F1-score
- **Plots and visualizations**:
    - Confusion matrices to examine per-class classification performance
    - Bar charts comparing predicted vs. true label distributions
    - Precision-Recall (PR) curves to evaluate performance across thresholds
    - Accuracy and loss curves across epochs to monitor training behavior

---

Results

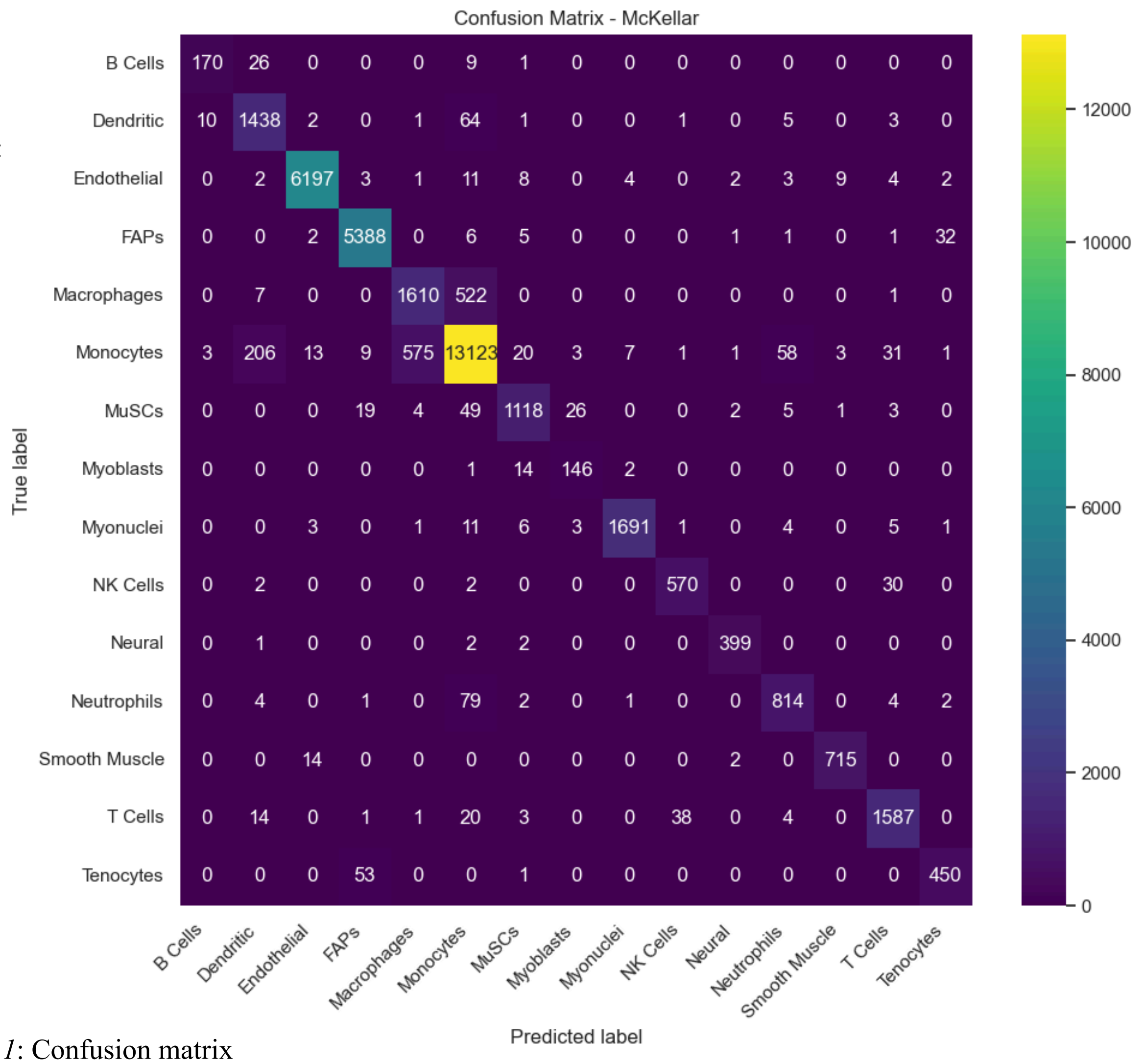
The CNN model was evaluated across four experimental setups, each involving a different combination of training and test datasets. Below are the results for each setup, including classification metrics and visualizations that support performance analysis.

Experiment 1: McKellar → McKellar

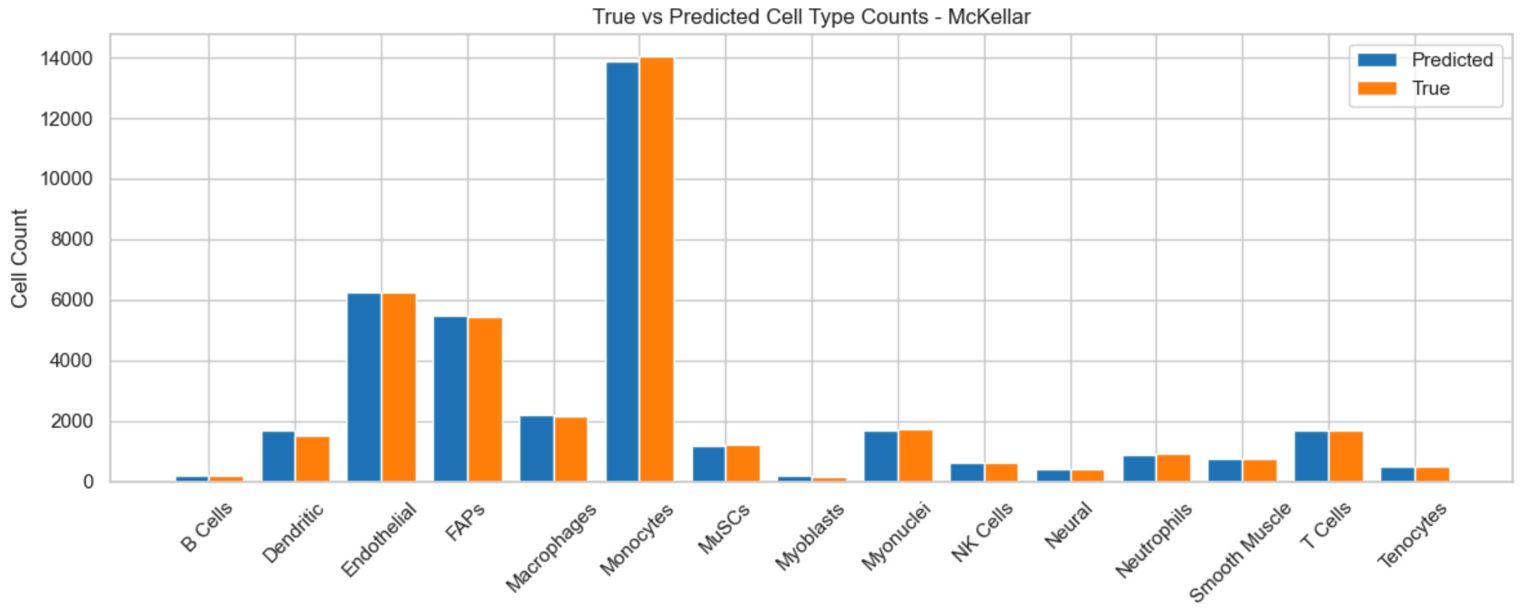
- **Accuracy**: **94.7%**
- **Precision**: 0.9306
- **Recall**: 0.9252
- **F1-Score**: 0.9273

Observation: The model performed best when trained and tested on the McKellar dataset, likely due to its larger sample size and more balanced class distribution.

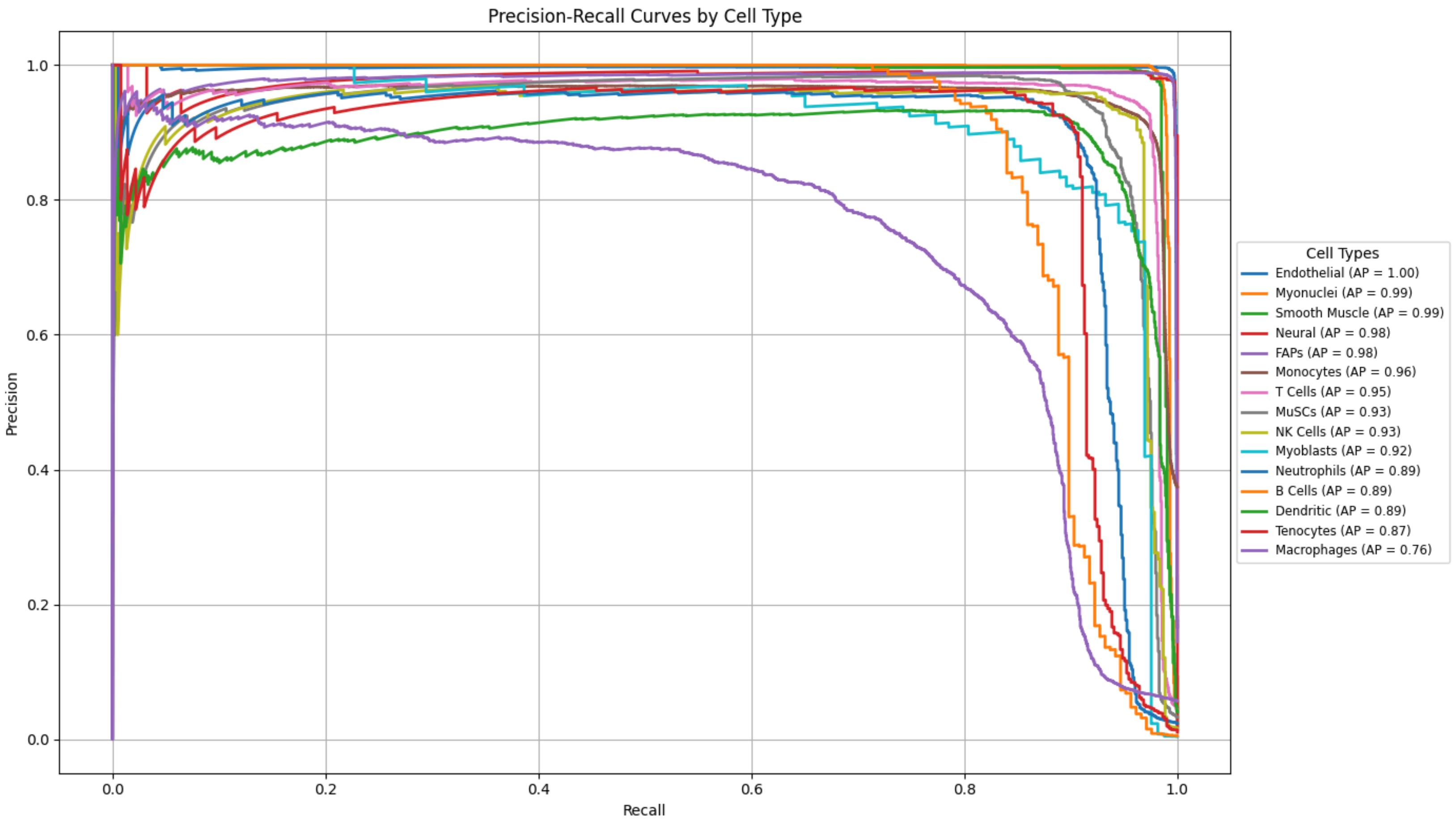
Visualizations:



- *Figure 1*: Confusion matrix

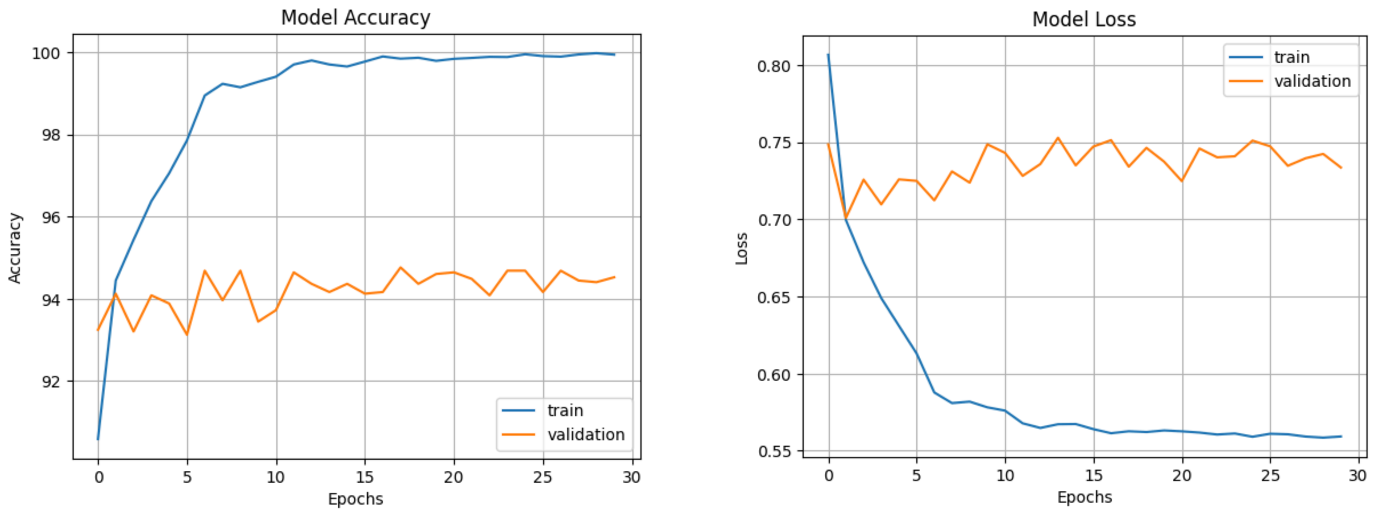


- *Figure 2*: Bar chart comparing predicted vs. true label counts



- *Figure 3*: Precision-Recall curve

The model shows consistently high precision and recall across nearly all cell types, with AP scores above 0.95 for many classes, including Endothelial, Myonuclei, and Smooth Muscle. The only notable exception is Macrophages (AP = 0.76), suggesting some overlap or ambiguity in that class.



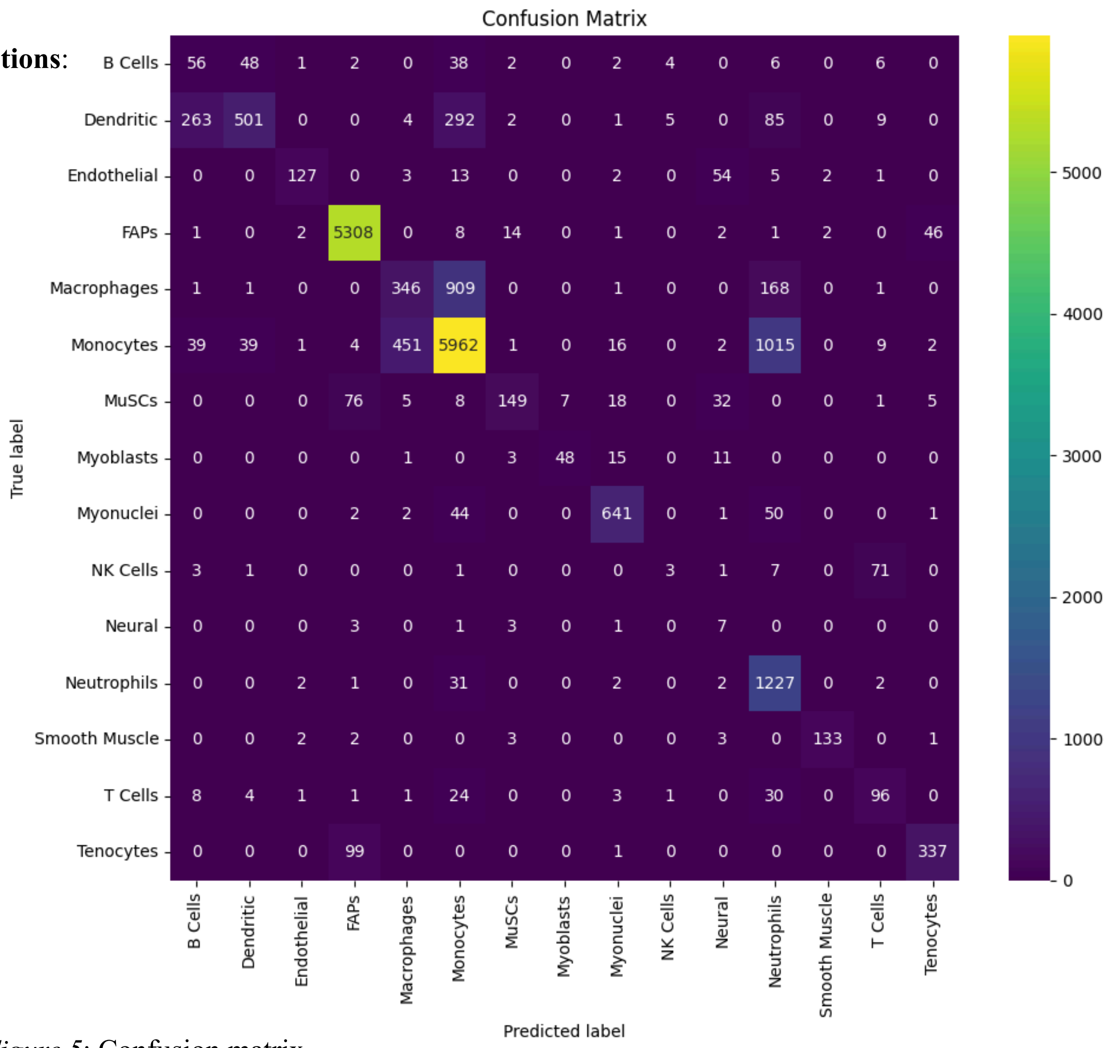
- *Figure 4*: Training curves (accuracy and loss over epochs)

Experiment 2A: Oprescu → Oprescu (3000 Highly Variable Genes {HVG}, 30 Epochs)

- **Accuracy**: **78.1%**
- **Precision**: 0.8027
- **Recall**: 0.7812
- **F1-Score**: 0.7782

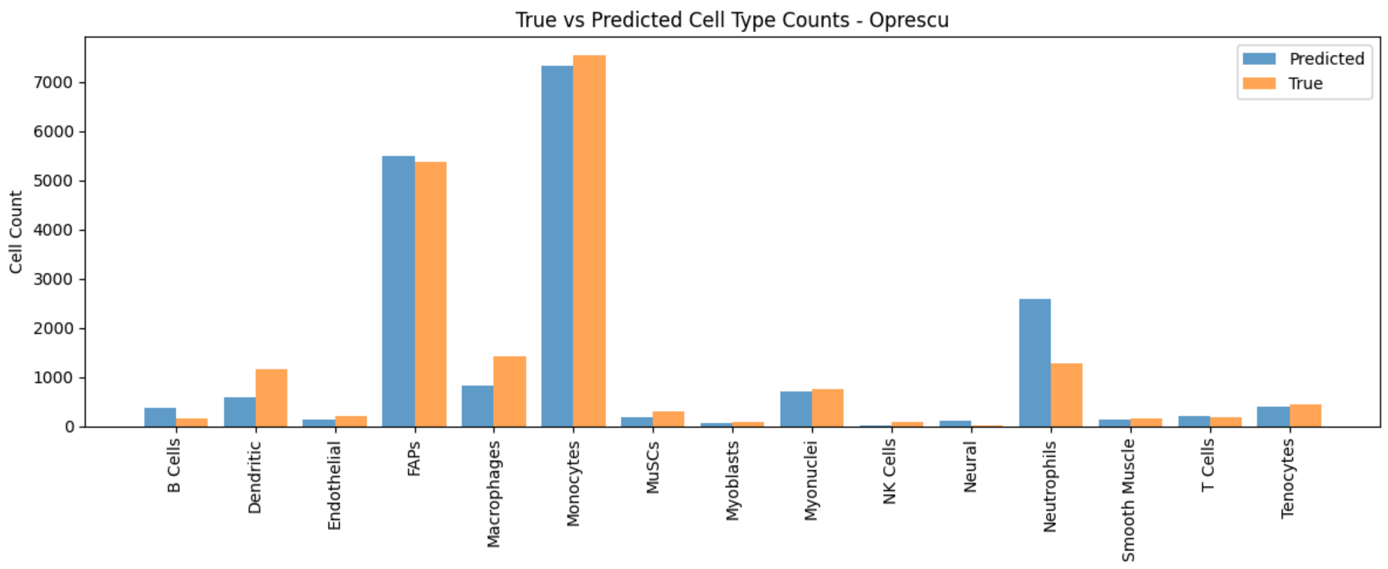
Observation: Performance was solid, though lower than on McKellar. This may reflect fewer samples and greater label imbalance.

Visualizations:

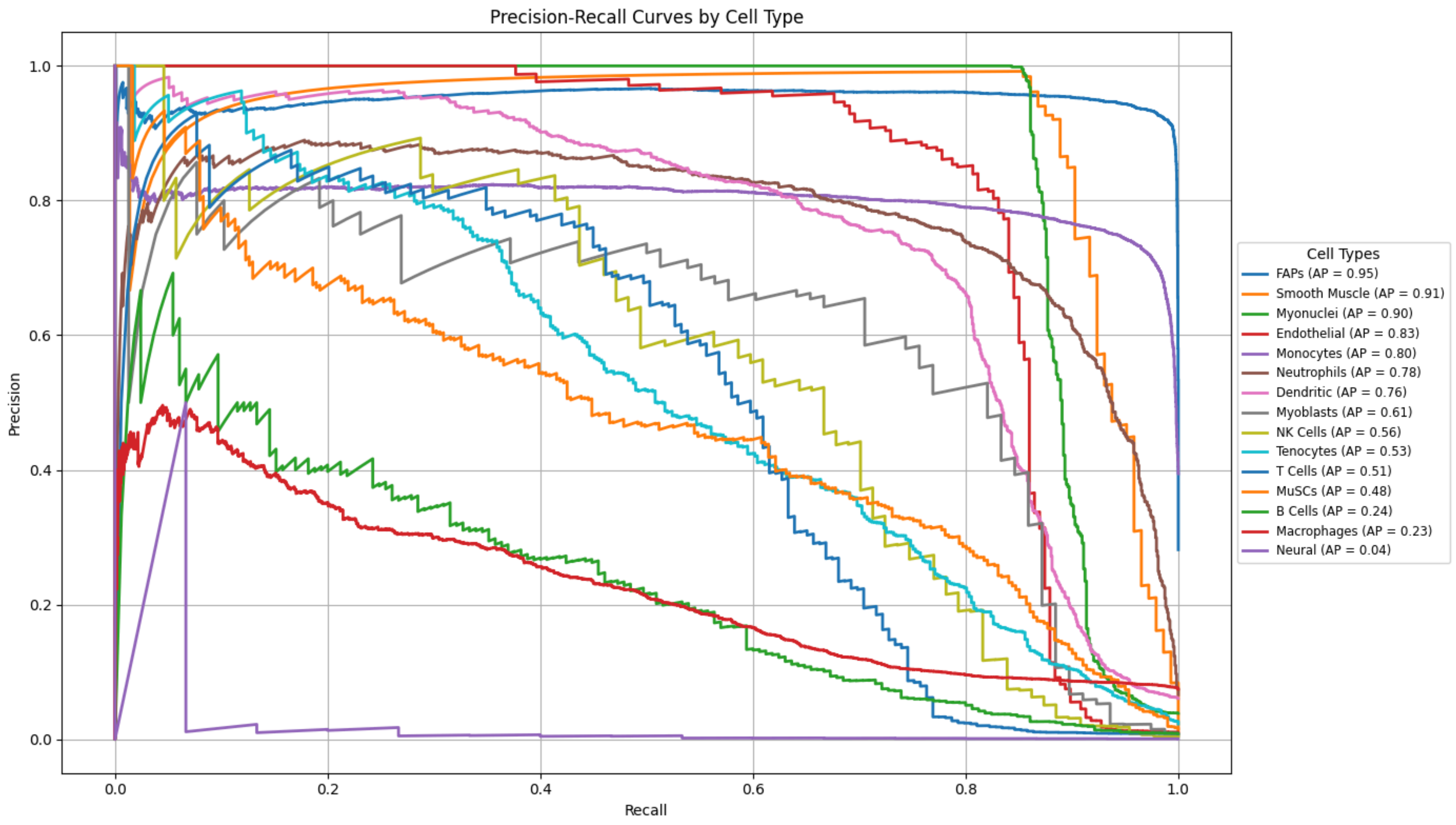


True label \ Predicted label	B Cells	Dendritic	Endothelial	FAPs	Macrophages	Monocytes	MuSCs	Myoblasts	Myonuclei	NK Cells	Neural	Neutrophils	Smooth Muscle	T Cells	Tenocytes
B Cells	56	48	1	2	0	38	2	0	2	4	0	6	0	6	0
Dendritic	263	501	0	0	4	292	2	0	1	5	0	85	0	9	0
Endothelial	0	0	127	0	3	13	0	0	2	0	54	5	2	1	0
FAPs	1	0	2	5308	0	8	14	0	1	0	2	1	2	0	46
Macrophages	1	1	0	0	346	909	0	0	1	0	0	168	0	1	0
Monocytes	39	39	1	4	451	5962	1	0	16	0	2	1015	0	9	2
MuSCs	0	0	0	76	5	8	149	7	18	0	32	0	0	1	5
Myoblasts	0	0	0	0	1	0	3	48	15	0	11	0	0	0	0
Myonuclei	0	0	0	2	2	44	0	0	641	0	1	50	0	0	1
NK Cells	3	1	0	0	0	1	0	0	0	3	1	7	0	71	0
Neural	0	0	0	3	0	1	3	0	1	0	7	0	0	0	0
Neutrophils	0	0	2	1	0	31	0	0	2	0	2	1227	0	2	0
Smooth Muscle	0	0	2	2	0	0	3	0	0	0	3	0	133	0	1
T Cells	8	4	1	1	1	24	0	0	3	1	0	30	0	96	0
Tenocytes	0	0	0	99	0	0	0	0	1	0	0	0	0	0	337

- *Figure 5*: Confusion matrix

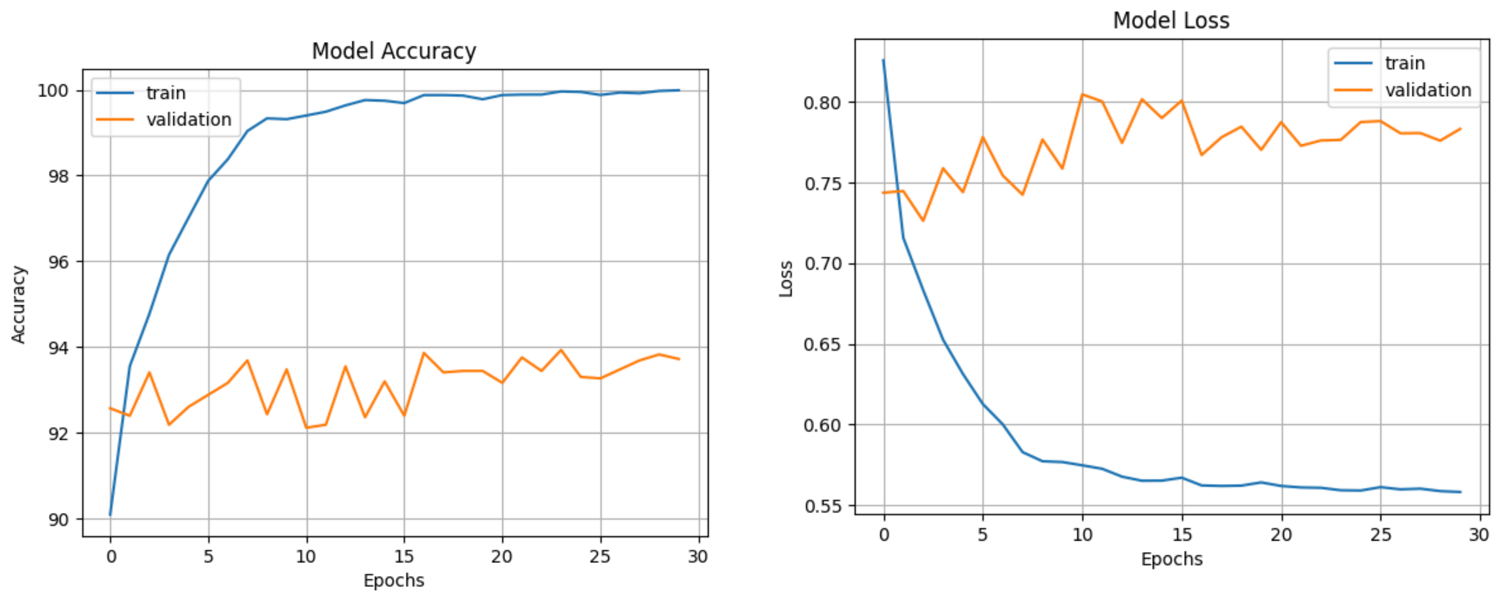


- *Figure 6*: Bar chart comparing predicted vs. true label counts



- *Figure 7*: Precision-Recall curve

The PR curves show strong performance for dominant cell types like FAPs, Smooth Muscle, and Myonuclei. In contrast, lower AP scores for B Cells, Macrophages, and Neural cells reflect challenges tied to class imbalance and potential overfitting due to noise when selecting 3,000 HVGs in a smaller dataset.



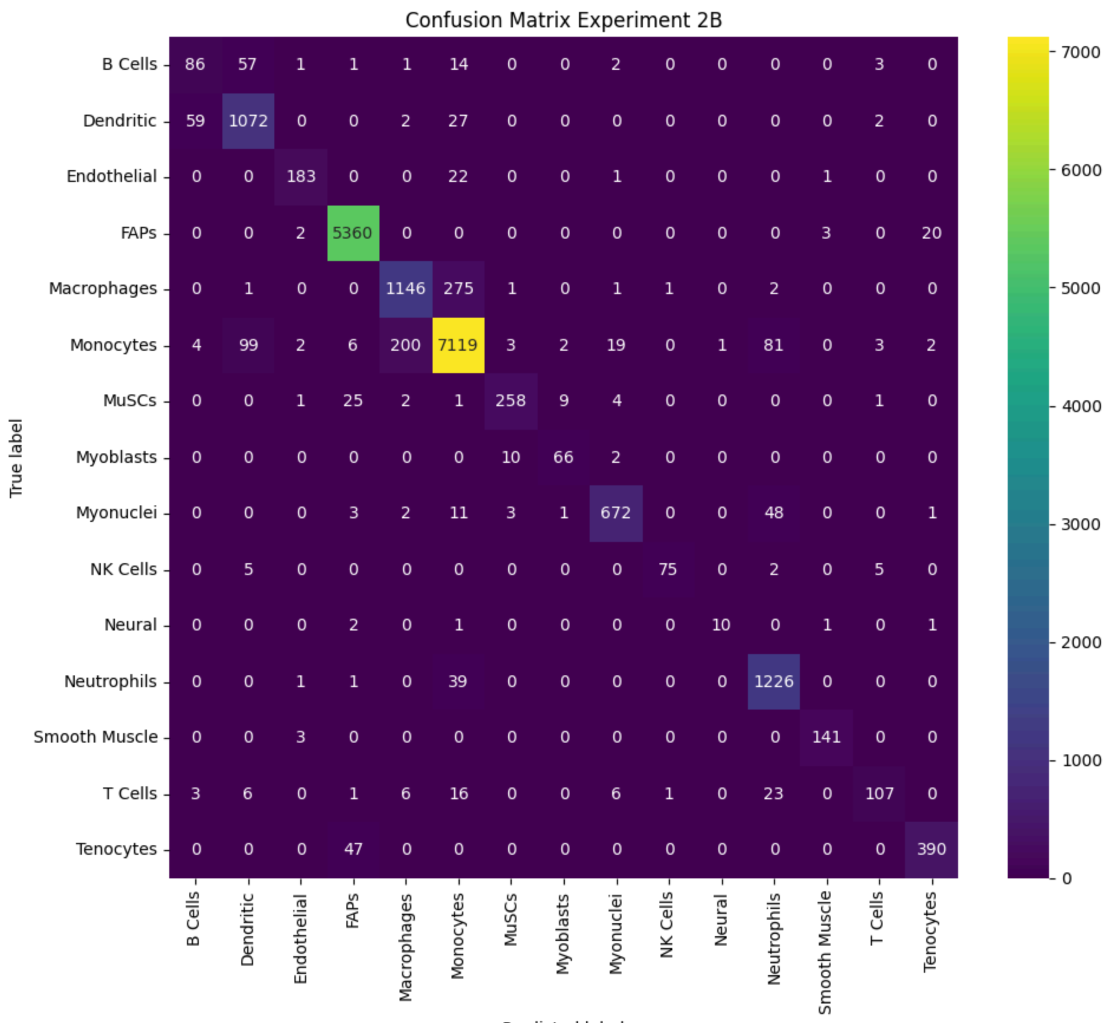
- *Figure 8*: Training curves (accuracy and loss over epochs)

Experiment 2B: Oprescu → Oprescu (2000 HVG, 10 Epochs)

- **Accuracy**: **93.6%**
- **Precision**: 0.9363
- **Recall**: 0.9365
- **F1-Score**: 0.9358

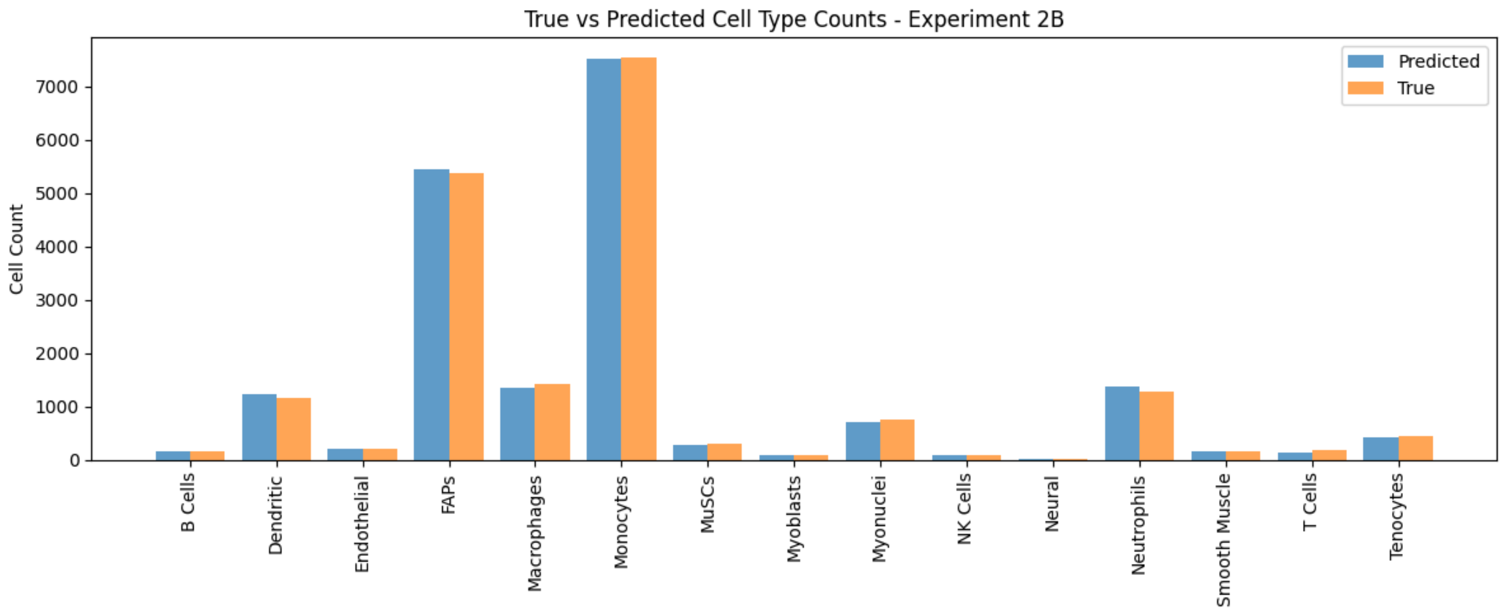
Observation: With 2,000 HVGs and a 10-epoch limit due to time constraints, the model achieved 93.6% accuracy on Oprescu, a significant improvement over the earlier 78.1%. This suggests that reducing the number of genes helped control noise and improve within-dataset classification on a smaller, more variable dataset.

Visualizations:

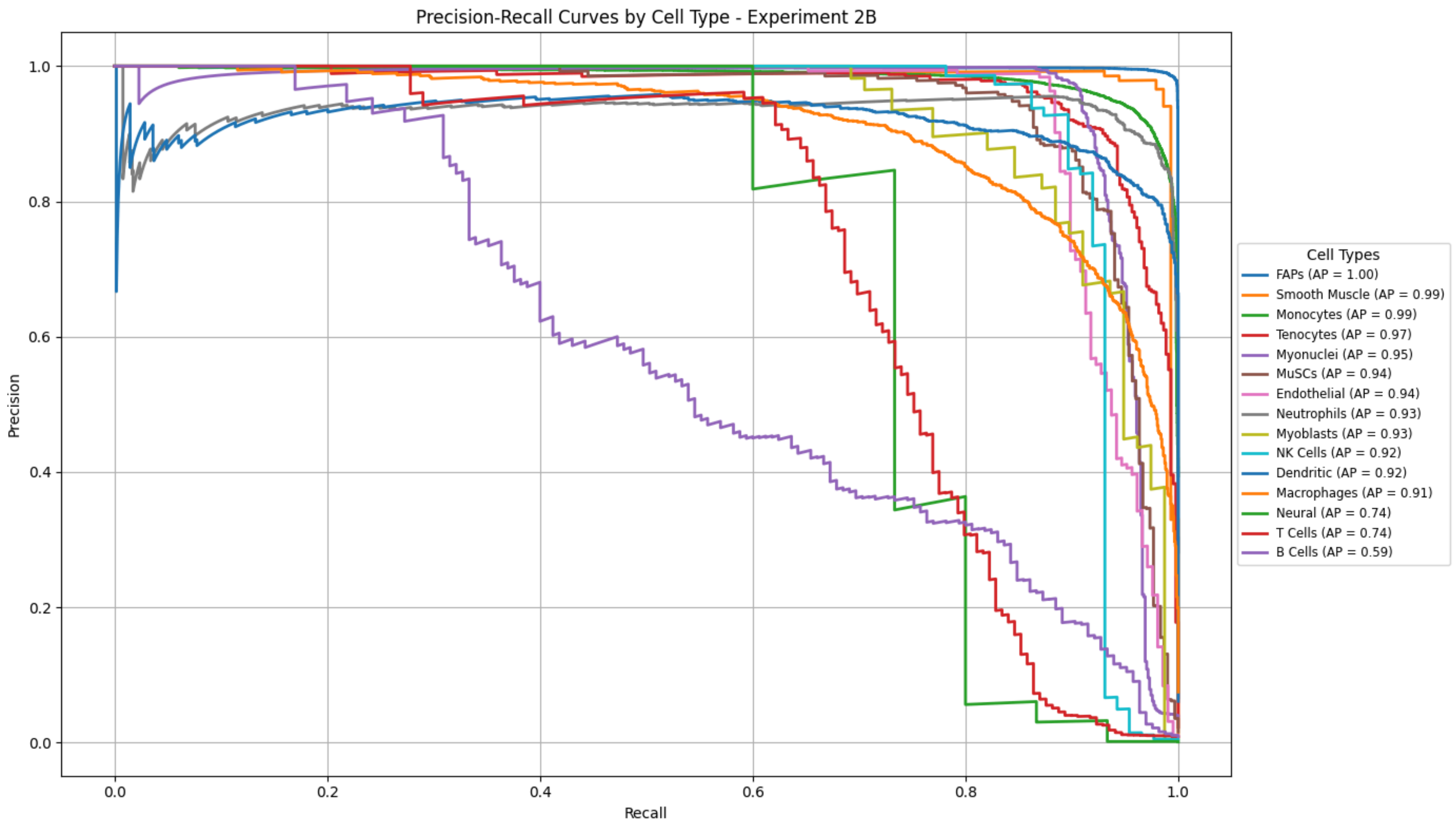


True label \ Predicted label	B Cells	Dendritic	Endothelial	FAPs	Macrophages	Monocytes	MuSCs	Myoblasts	Myonuclei	NK Cells	Neural	Neutrophils	Smooth Muscle	T Cells	Tenocytes
B Cells	86	57	1	1	1	14	0	0	2	0	0	0	0	3	0
Dendritic	59	1072	0	0	2	27	0	0	0	0	0	0	0	2	0
Endothelial	0	0	183	0	0	22	0	0	1	0	0	0	1	0	0
FAPs	0	0	2	5360	0	0	0	0	0	0	0	0	3	0	20
Macrophages	0	1	0	0	1146	275	1	0	1	1	0	2	0	0	0
Monocytes	4	99	2	6	200	7119	3	2	19	0	1	81	0	3	2
MuSCs	0	0	1	25	2	1	258	9	4	0	0	0	0	1	0
Myoblasts	0	0	0	0	0	0	10	66	2	0	0	0	0	0	0
Myonuclei	0	0	0	3	2	11	3	1	672	0	0	48	0	0	1
NK Cells	0	5	0	0	0	0	0	0	0	75	0	2	0	5	0
Neural	0	0	0	2	0	1	0	0	0	0	10	0	1	0	1
Neutrophils	0	0	1	1	0	39	0	0	0	0	0	1226	0	0	0
Smooth Muscle	0	0	3	0	0	0	0	0	0	0	0	0	141	0	0
T Cells	3	6	0	1	6	16	0	0	6	1	0	23	0	107	0
Tenocytes	0	0	0	47	0	0	0	0	0	0	0	0	0	0	390

- *Figure 9*: Confusion matrix

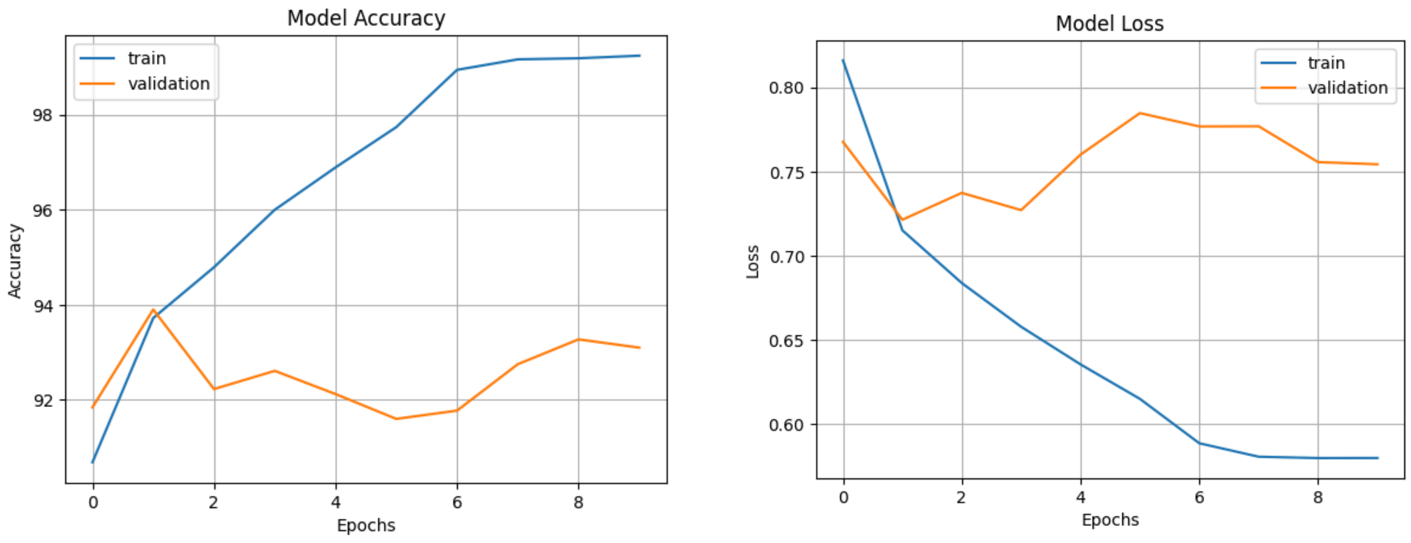


- *Figure 10*: Bar chart



- *Figure 11*: Precision-Recall curve

Most cell types achieved high average precision (AP), with several exceeding 0.95. While T Cells and B Cells showed lower scores, the overall performance reflects improved precision-recall dynamics after reducing HVGs to 2,000.



- *Figure 12*: Training curves

Experiment 3A: McKellar → Oprescu (Unaligned Test Images, 30 Epochs)

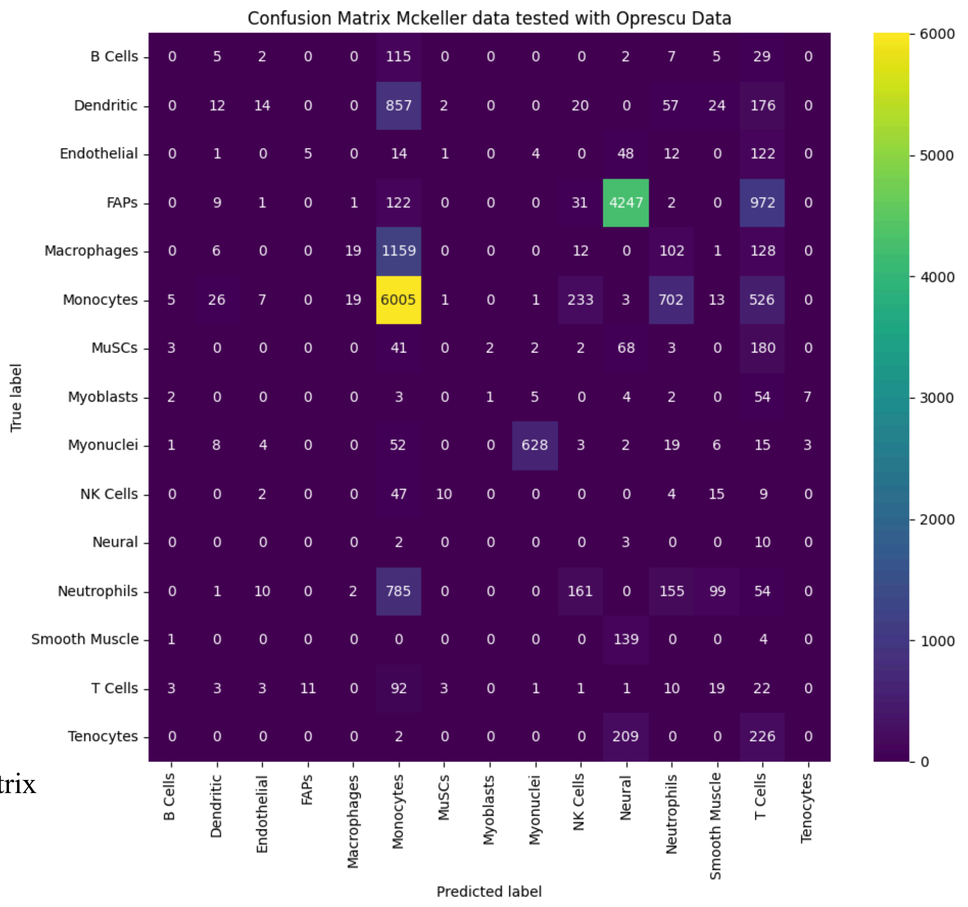
- **Accuracy**: **35.8%**
- **Precision**: 0.3485
- **Recall**: 0.3579
- **F1-Score**: 0.3286

Description:
The model was trained on McKellar and tested on Oprescu. However, the test image arrays for Oprescu were generated using a transformation pipeline fitted on **Oprescu's own training data**, including separate HVG selection, scaling, and t-SNE layout.

Observation:
This mismatch between training and test preprocessing led to poor generalization, with the model achieving only **35.8% accuracy**, despite shared cell types. The results suggest that inconsistent preprocessing pipelines across datasets can significantly impair model performance.

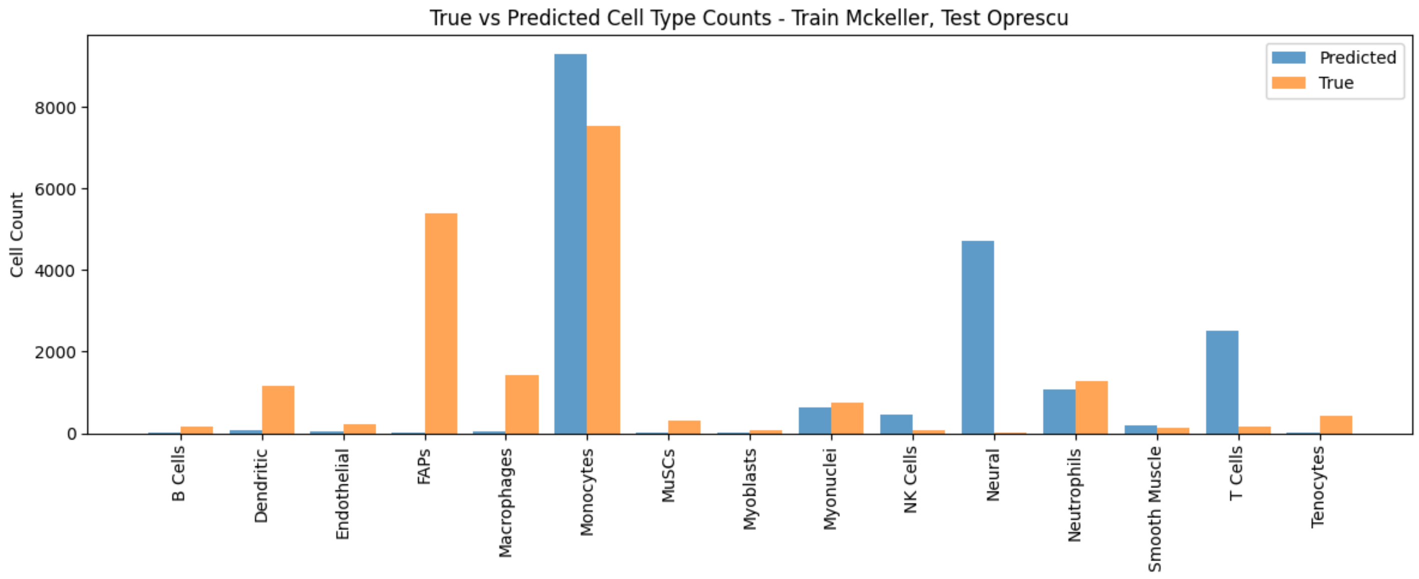
Visualizations:

Note: A PR curve was not generated for Experiment 3A, as I transitioned to the aligned setup (Experiment 3B) shortly after initial testing. Given time constraints, I was unable to revisit and re-run Experiment 3A.

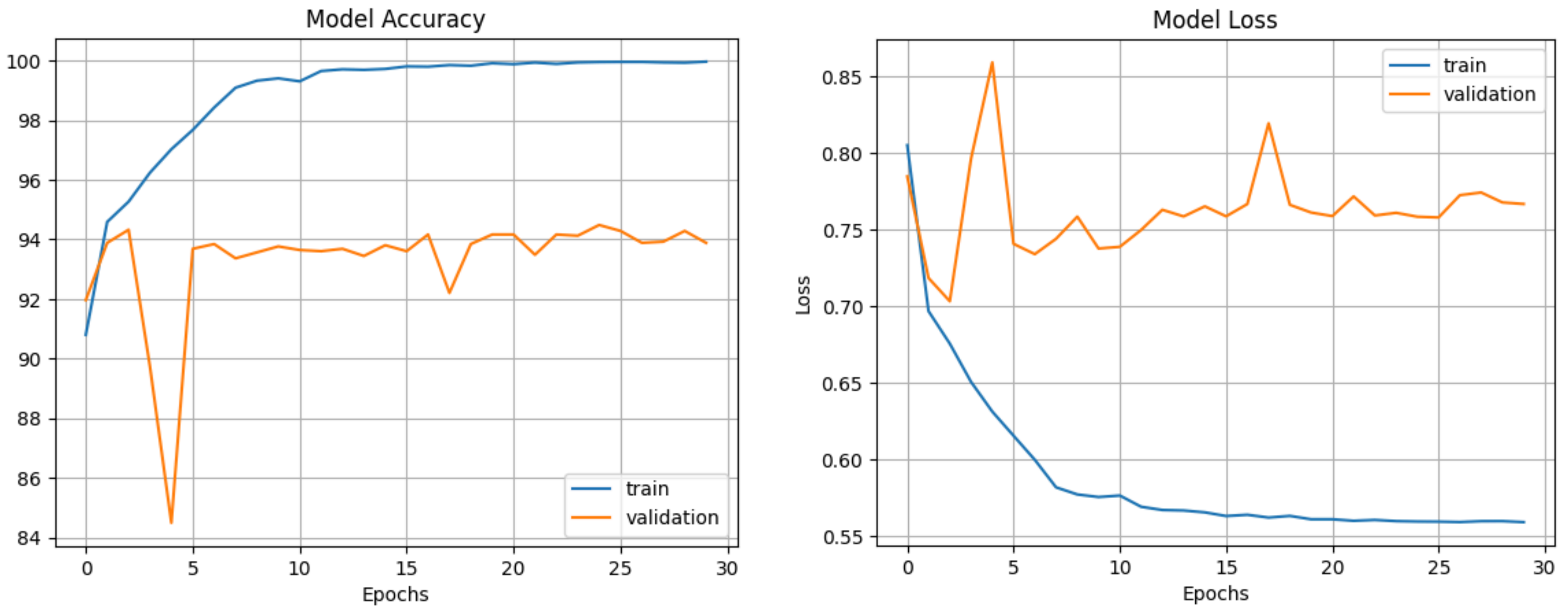


True label \ Predicted label	B Cells	Dendritic	Endothelial	FAPs	Macrophages	Monocytes	MuSCs	Myoblasts	Myonuclei	NK Cells	Neural	Neutrophils	Smooth Muscle	T Cells	Tenocytes
B Cells	0	5	2	0	0	115	0	0	0	0	2	7	5	29	0
Dendritic	0	12	14	0	0	857	2	0	0	20	0	57	24	176	0
Endothelial	0	1	0	5	0	14	1	0	4	0	48	12	0	122	0
FAPs	0	9	1	0	1	122	0	0	0	31	4247	2	0	972	0
Macrophages	0	6	0	0	19	1159	0	0	0	12	0	102	1	128	0
Monocytes	5	26	7	0	19	6005	1	0	1	233	3	702	13	526	0
MuSCs	3	0	0	0	0	41	0	2	2	2	68	3	0	180	0
Myoblasts	2	0	0	0	0	3	0	1	5	0	4	2	0	54	7
Myonuclei	1	8	4	0	0	52	0	0	628	3	2	19	6	15	3
NK Cells	0	0	2	0	0	47	10	0	0	0	0	4	15	9	0
Neural	0	0	0	0	0	2	0	0	0	0	3	0	0	10	0
Neutrophils	0	1	10	0	2	785	0	0	0	161	0	155	99	54	0
Smooth Muscle	1	0	0	0	0	0	0	0	0	0	139	0	0	4	0
T Cells	3	3	3	11	0	92	3	0	1	1	1	10	19	22	0
Tenocytes	0	0	0	0	0	2	0	0	0	0	209	0	0	226	0

- *Figure 13*: Confusion matrix



- *Figure 14*: Bar chart



- *Figure 15*: Training curves

Experiment 3B: McKellar → Oprescu (Aligned Test Images, 10 Epochs)

- **Accuracy**: **91.3%**

Macro average:

- **Precision**: 0.8902
- **Recall**: 0.8256
- **F1-score**: 0.8380

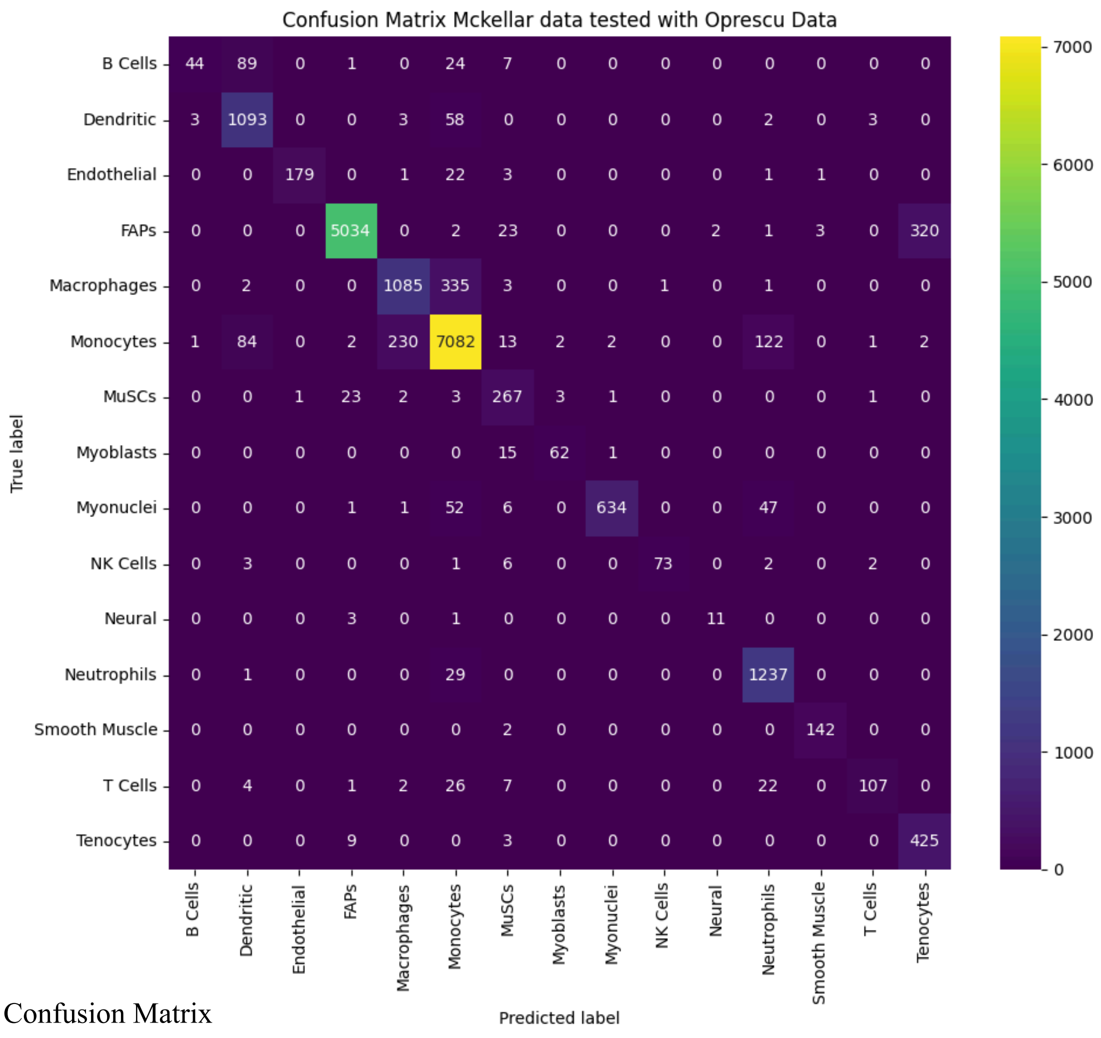
Weighted average :

- **Precision**: 0.9219
- **Recall**: 0.9137
- **F1-score**: 0.9137

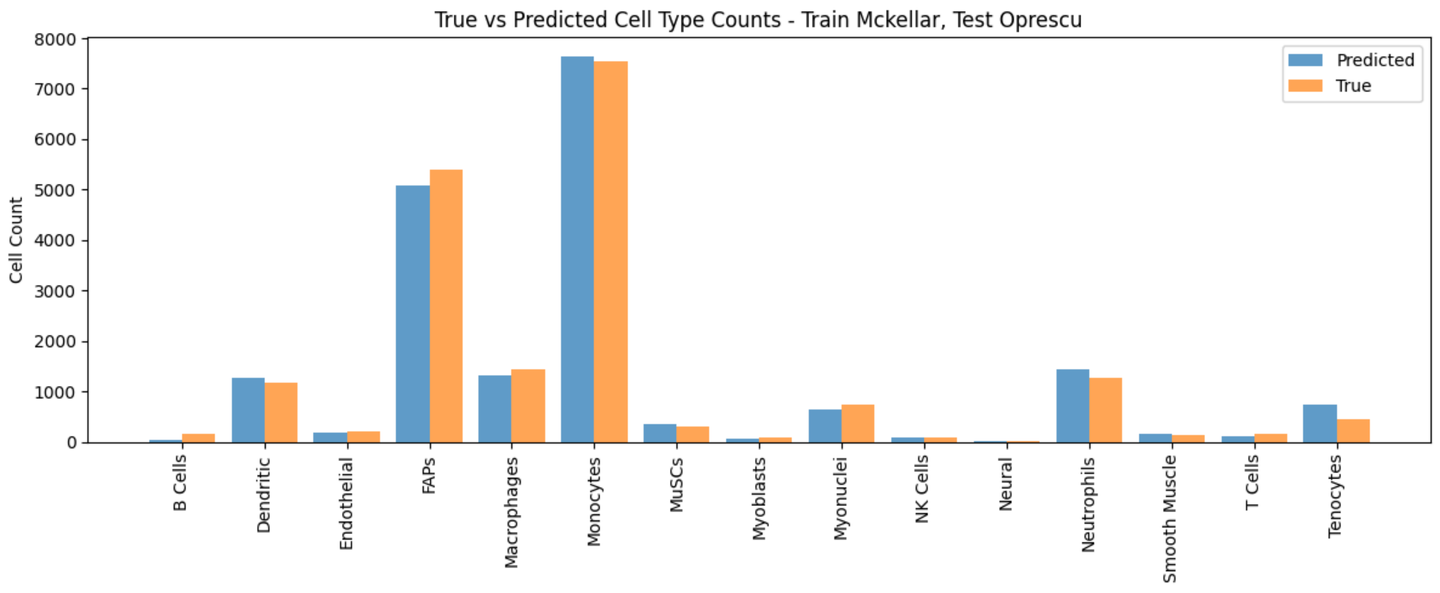
Description:
The model was trained on McKellar and tested on Oprescu. This time, the Oprescu test image arrays were generated using the same transformation pipeline fitted on the McKellar training data, including identical HVG selection, MinMax scaling, and t-SNE layout.

Observation:
Aligning the test data preprocessing with the training set led to a dramatic improvement in generalization. The model achieved **91.3% accuracy**, even though it was trained for only 10 epochs due to time limitations. This result highlights the importance of consistent preprocessing when evaluating cross-dataset performance.

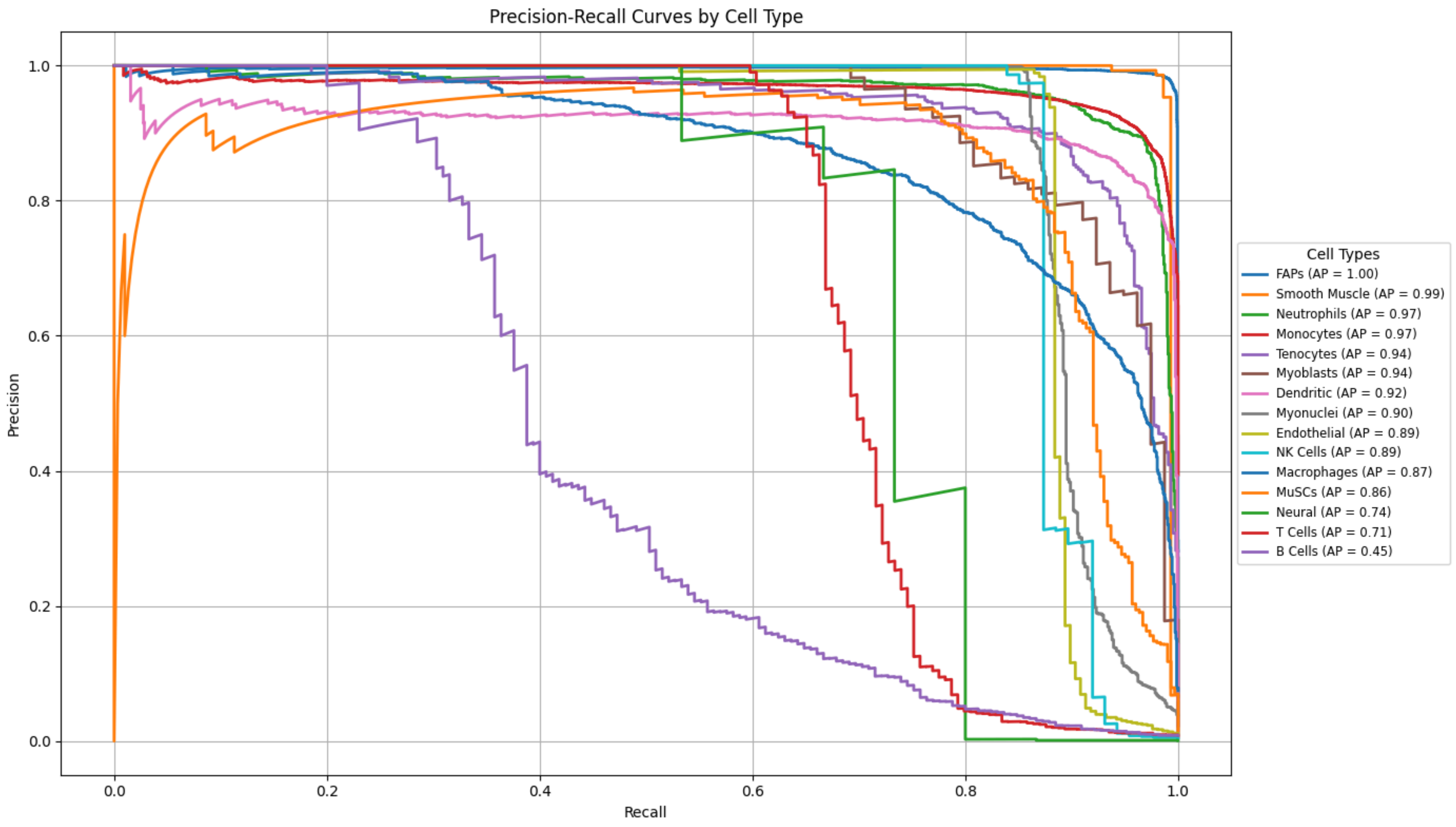
Visualizations:



- *Figure 16*: Confusion Matrix

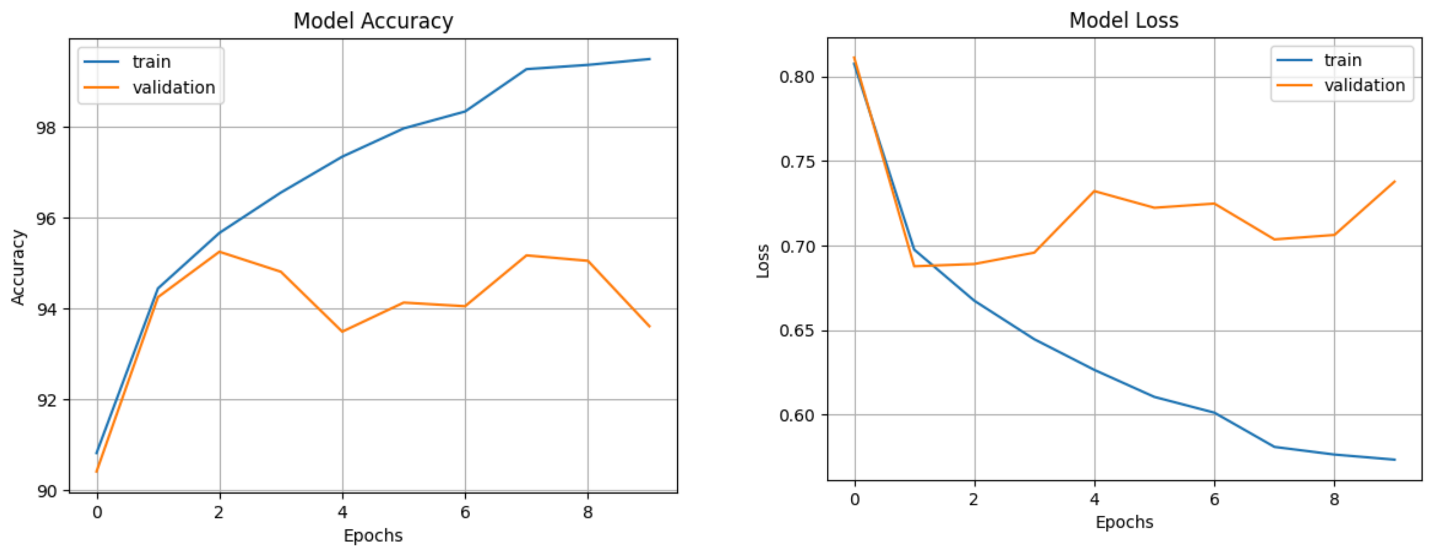


- *Figure 17*: *Bar chart*



- *Figure 18*: Precision-Recall curve

Most cell types achieved high average precision (AP), with several exceeding 0.90. Although FAPs reached an AP of 1.00, the confusion matrix reveals a few misclassifications, indicating that the model ranked FAPs highly but still made occasional errors at the prediction threshold. Overall, the PR curves confirm strong generalization when test preprocessing was aligned with the training set.



- *Figure 19*: Training curves (accuracy and loss over epochs)

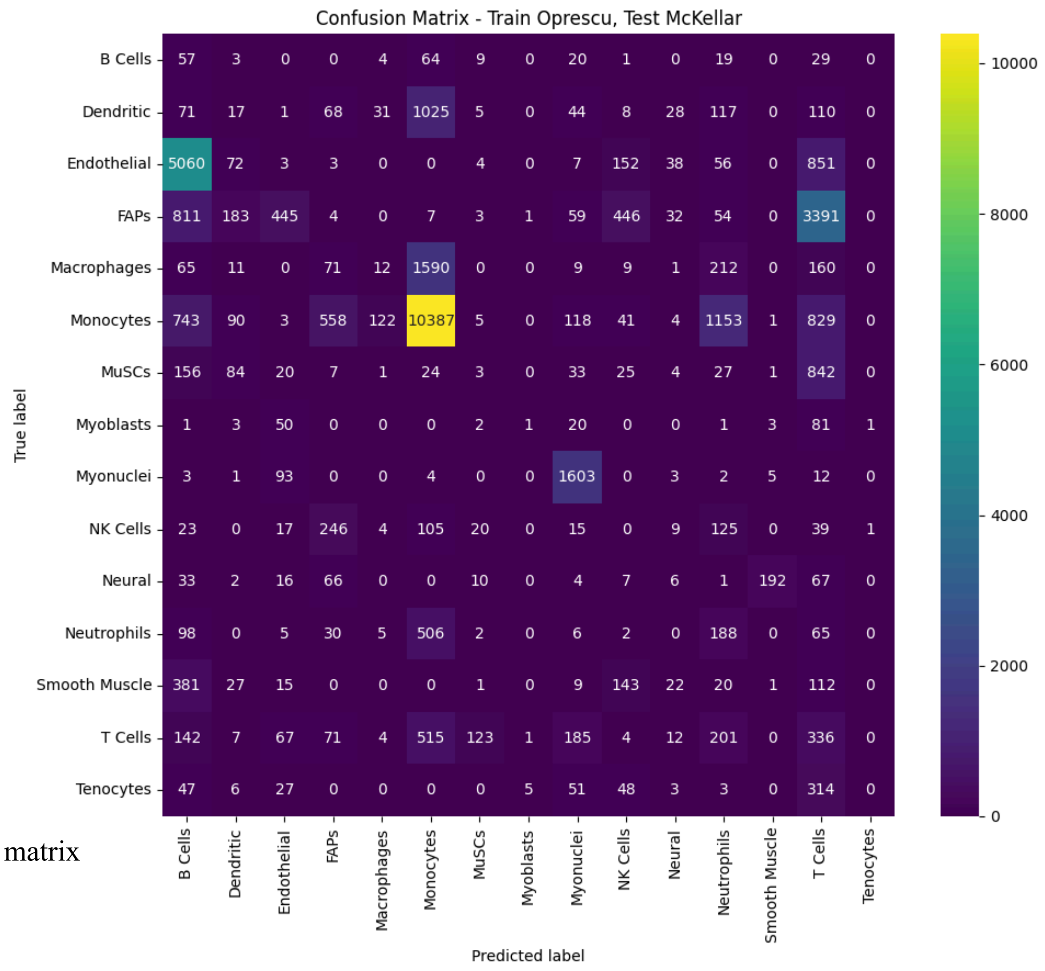
Experiment 4A: Oprescu → McKellar (Unaligned Test Images, 30 Epochs)

- **Accuracy**: **33.6%**
- **Precision**: 0.3191
- **Recall**: 0.3361
- **F1-Score**: 0.3211

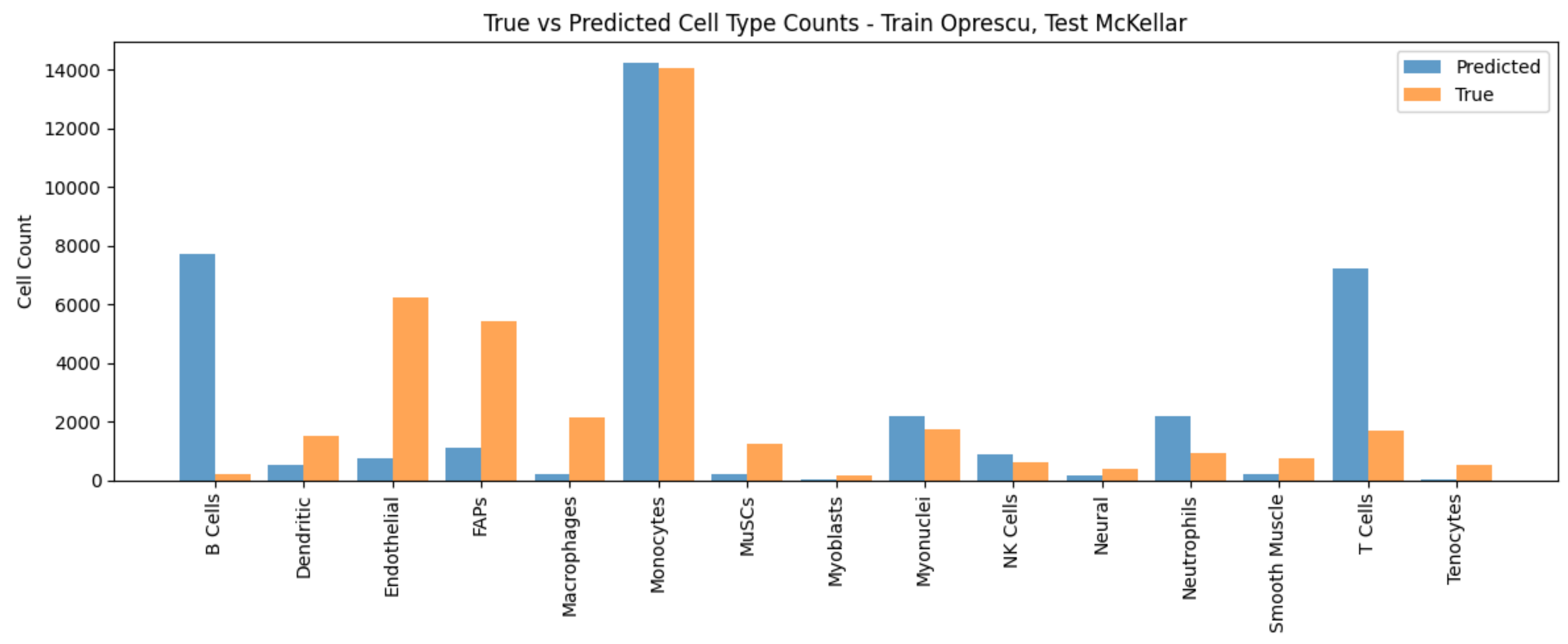
Description:
The model was trained on Oprescu and tested on McKellar. The McKellar test images were generated using transformations fitted on McKellar's own training data, separate from Oprescu's pipeline.

Observation:
The mismatch between preprocessing steps led to poor generalization, with many cell types misclassified. This result mirrors Experiment 3A and shows that inconsistent preprocessing hurts cross-dataset performance.

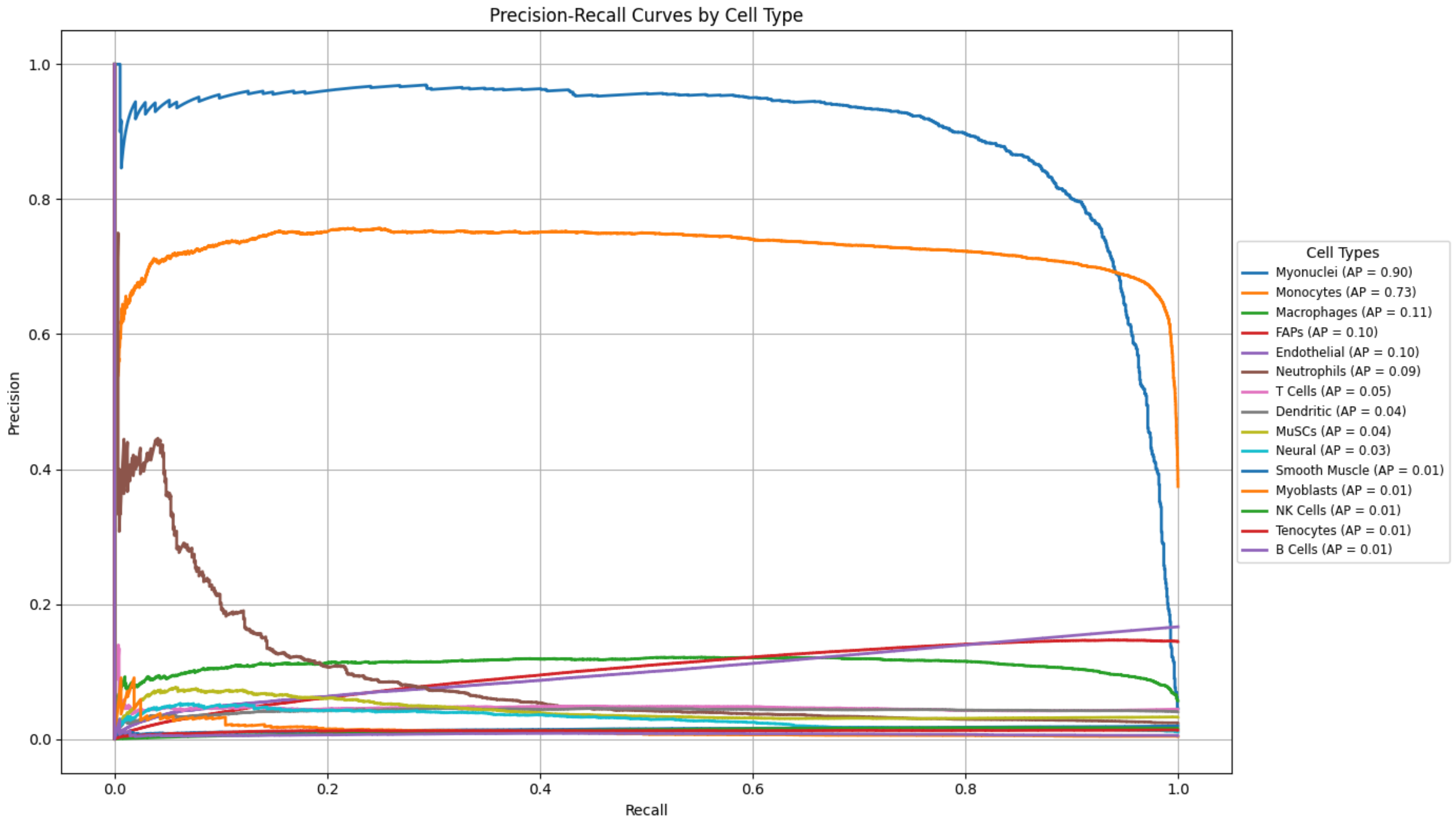
Visualizations:



- *Figure 20*: Confusion matrix

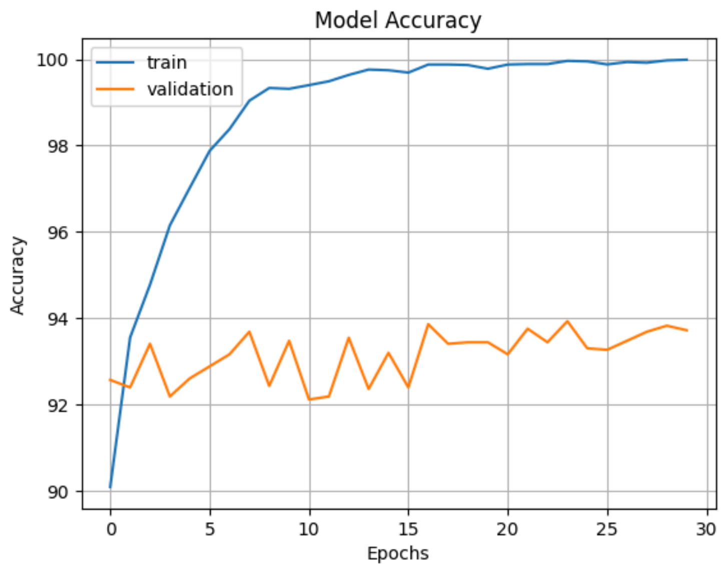


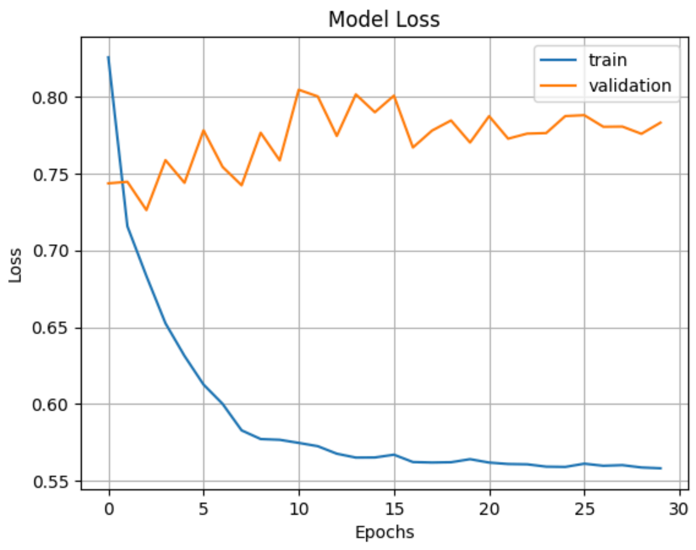
- *Figure 21*: Bar chart



- *Figure 22*: Precision-Recall curve

The model shows high classification performance for Myonuclei and Monocytes, but struggles with most other cell types, which demonstrate low precision and recall.





- *Figure 23*: Training curves

Experiment 4B: Oprescu → McKellar (Aligned Test Images, 30 Epochs)

- **Accuracy**: **92.2%**

Macro average:

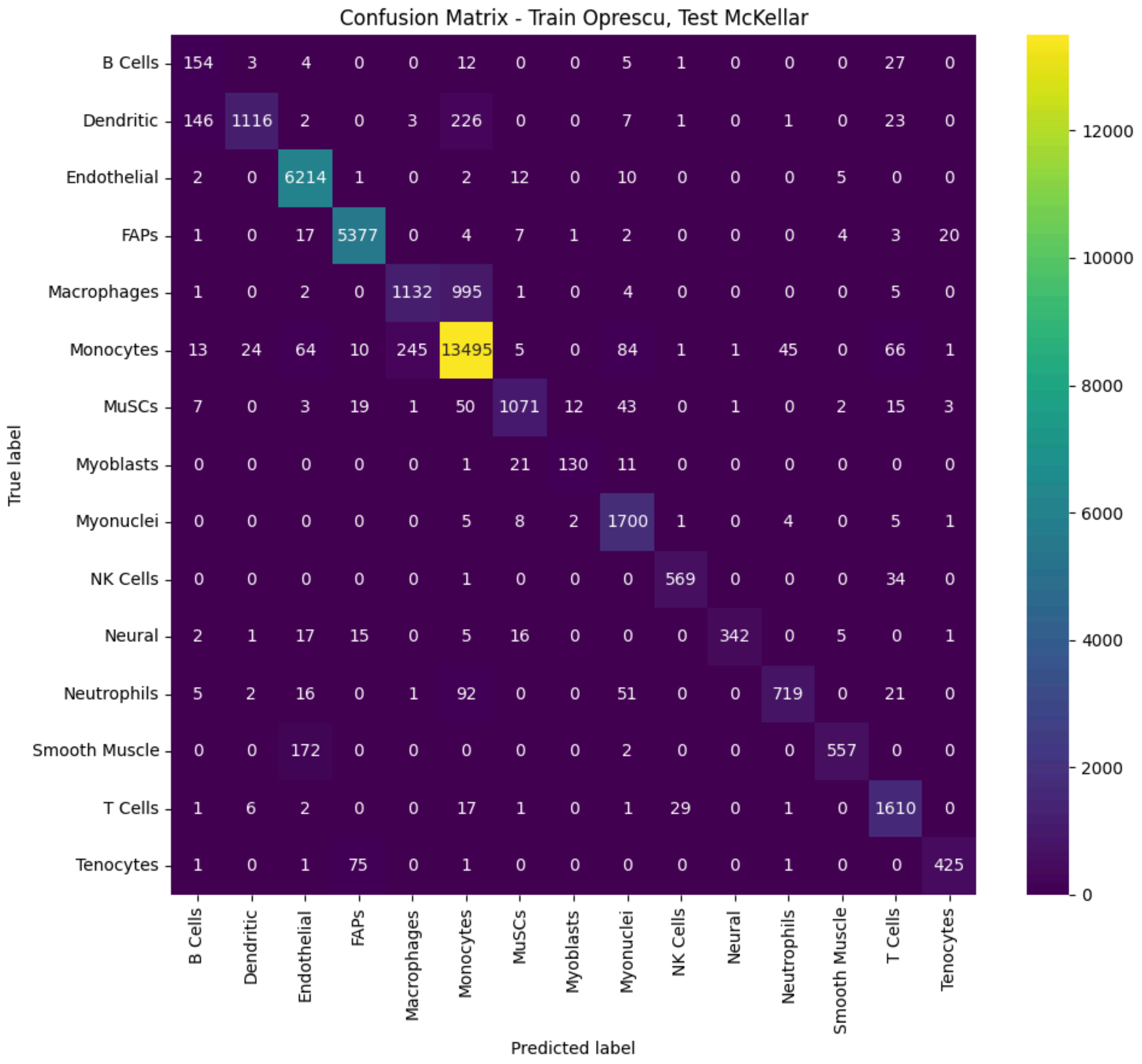
- **Precision**: 0.899
- **Recall**: 0.851
- **F1-score**: 0.867

Weighted average :

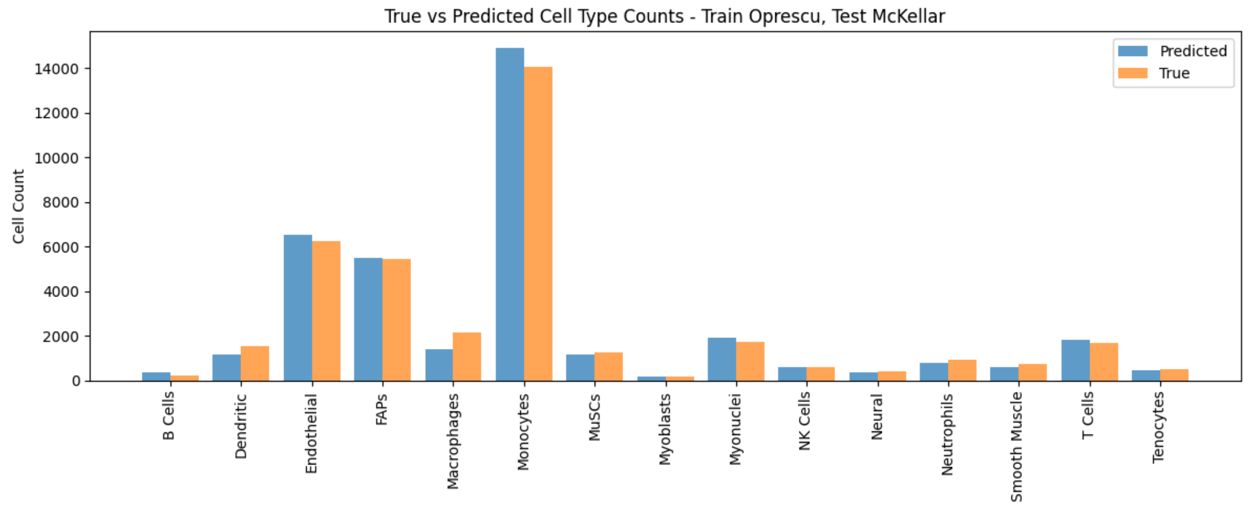
- **Precision**: 0.923
- **Recall**: 0.922
- **F1-score**: 0.919

Description:
The model was trained on Oprescu and tested on McKellar. Test image arrays for McKellar were generated using the same preprocessing pipeline fitted on the Oprescu training data, including HVG selection, scaling, and t-SNE layout.

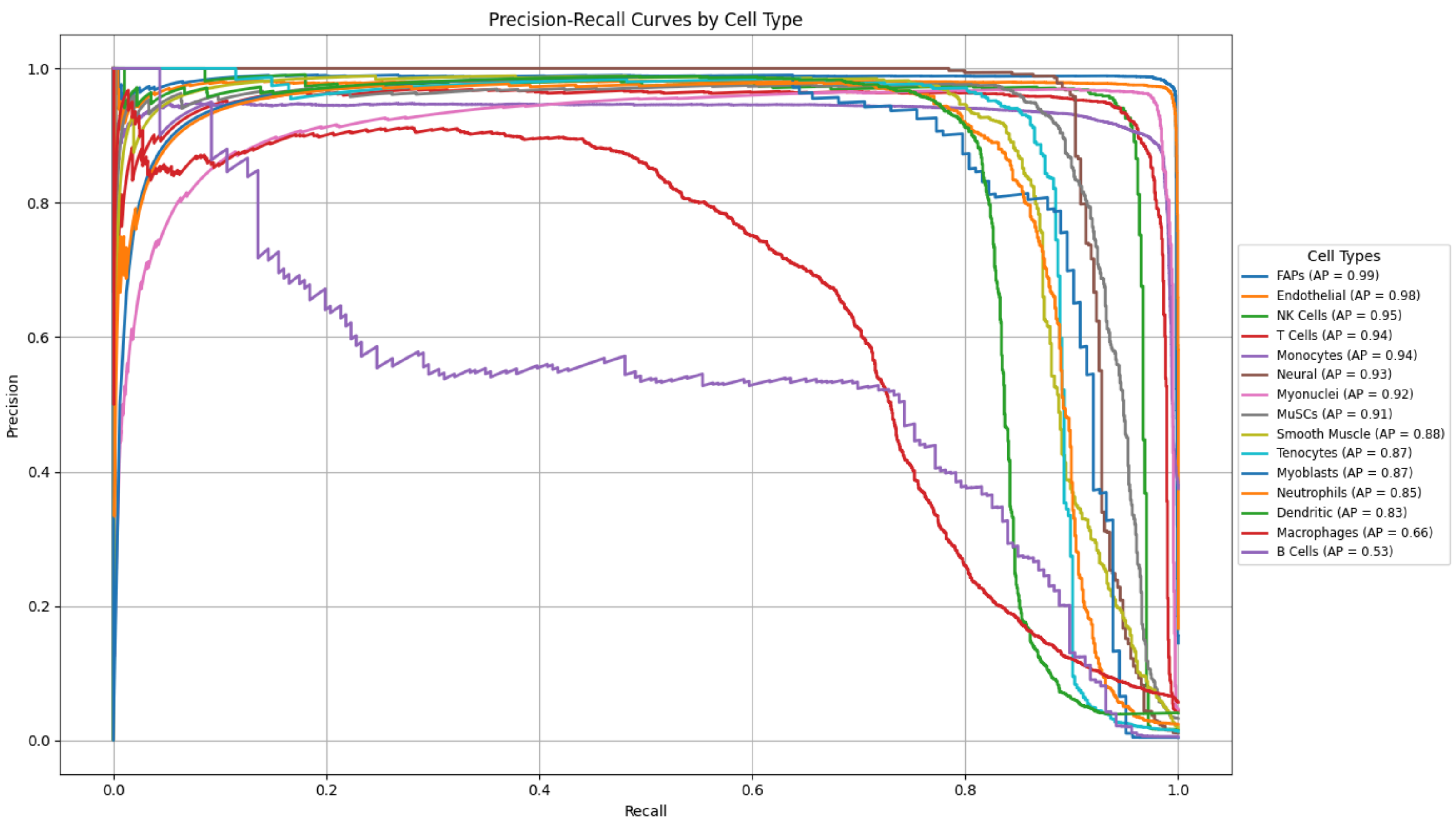
Visualizations:



● *Figure 24*: Confusion Matrix

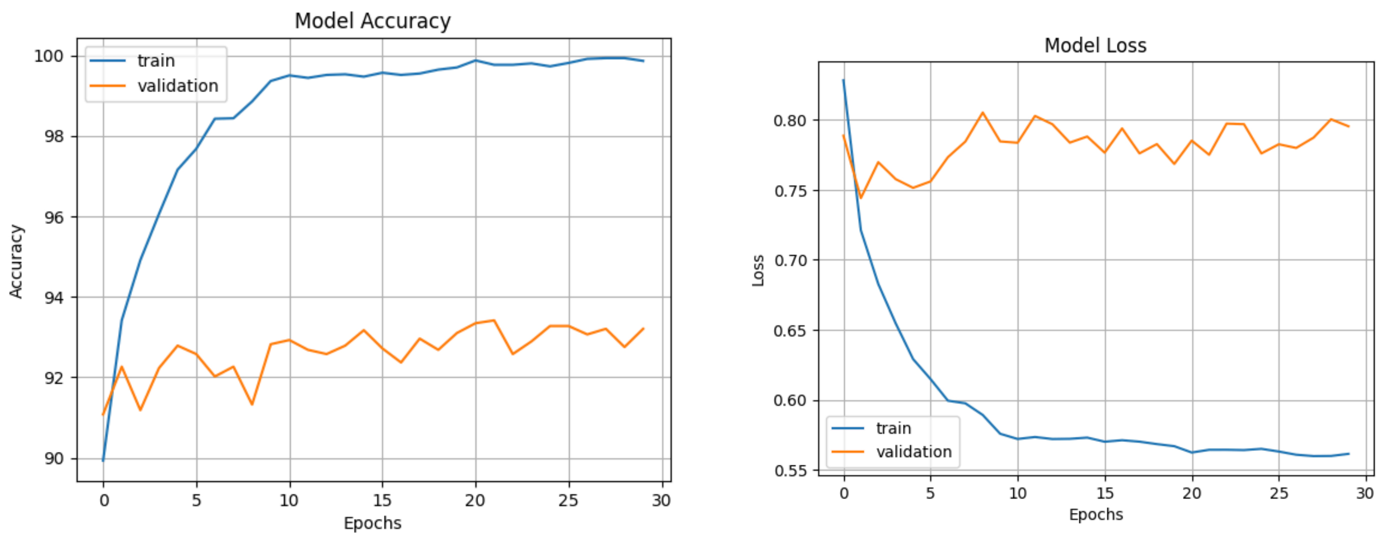


● *Figure 25*: Bar chart



- *Figure 26*: Precision-Recall curve

Most cell types achieved strong average precision, with over half exceeding 0.90. A few classes like Macrophages and B Cells showed lower AP, indicating some difficulty distinguishing them. Overall, the PR curves reflect solid generalization



- *Figure 27*: Training curves

All Results Combined

Experiment	Training Dataset	Test Dataset	Image Alignment	# HVGs	Epochs	Accuracy (%)	Notes
1	McKellar	McKellar	Aligned	3000	30	**94.7**	Best within-dataset result
2A	Oprescu	Oprescu	Aligned	3000	30	**78.1**	Lower performance, potential noise from 3000 HVGs
2B	Oprescu	Oprescu	Aligned	2000	10	**93.6**	Improved results by reducing HVGs despite fewer epochs
3A	McKellar	Oprescu	*Unaligned*	3000	30	**35.8**	Mismatched preprocessing led to poor generalization
3B	McKellar	Oprescu	Aligned	3000	10	**91.3**	Aligned preprocessing resulted in high cross-dataset accuracy
4A	Oprescu	McKellar	*Unaligned*	3000	30	**33.6**	Similar mismatch as 3A
4B	Oprescu	McKellar	Aligned	3000	30	**92.2**	Similar alignment as 3B enabled strong generalization

Discussion

The scDeepInsight model showed strong performance when trained and tested on the same dataset, particularly with McKellar, where it reached **94.7% accuracy**. This high performance can be explained by several factors:

- **Balanced Cell Types**: The McKellar dataset included a more balanced distribution of cell types, giving the model consistent examples to learn from across all classes.

- **Larger Sample Size**: McKellar also had more cells in the training set, which helped the CNN generalize better during training.

In smaller datasets like Oprescu, selecting too many HVGs (e.g., 3,000) may introduce noise and reduce performance, particularly for underrepresented or similar cell types. In Experiment 2B, reducing the HVG count to 2,000 improved accuracy significantly, even though the model was only trained for 10 epochs due to time constraints. While additional training might have further improved results, this outcome highlights how careful feature selection alone can boost generalization in limited settings.

Cross-dataset experiments revealed that model performance depends heavily on how the test data is prepared. In the earlier setup, I created the test image arrays for Oprescu by fitting the transformation steps (such as t-SNE layout and scaling) on Oprescu itself, then evaluated the model trained on McKellar. This mismatch led to poor performance, with only 35.8% accuracy. In contrast, when I used the exact same preprocessing components fitted on McKellar to transform the Oprescu test data, accuracy improved dramatically to 91.0%.

I chose not to apply batch correction in this project to evaluate how well scDeepInsight performs when trained and tested on datasets from different studies without aligning them. Since the pipeline I followed was based entirely in Python, and scDeepInsight already performed well on within-dataset tasks, I prioritized maintaining a consistent and reproducible workflow.

Although batch correction might improve generalization across datasets, exploring that would involve a different preprocessing pipeline in R, which I plan to investigate in future work.

Limitations

No Batch Correction: Cross-dataset performance may have improved with batch alignment. However, this step was excluded to isolate the model's raw generalization ability.

Label Mismatch (Fixed): Initially, a label encoder mismatch caused low accuracy scores during evaluation, but this was later fixed by properly loading and applying the original label encoder used during training.

Computational Constraints: Unlike the original scDeepInsight experiments, which were run on high-end GPUs (Quadro RTX 8000) and a Xeon Gold CPU, this project was trained on a more limited Apple M1 Pro. The reduced computational power required using smaller batch sizes and fewer epochs, which may have affected the model's ability to learn more complex or less common cell types.

Conclusion

This project evaluated scDeepInsight as a deep learning framework for cell type classification using single-cell gene expression data. The model consistently performed well in within-dataset experiments, confirming the effectiveness of converting expression profiles into image representations for CNN-based classification.

Cross-dataset evaluation showed that performance depended not only on which datasets were used, but also on how the preprocessing was applied. In one setup, the transformation steps (such as t-SNE layout and scaling) were fitted using the test dataset's training data before generating test images, resulting in a mismatch that led to low accuracy. However, when those steps were fitted on the training data and then applied to the test data from a different dataset, performance improved significantly. This demonstrates that consistent preprocessing, anchored to the training data, plays a key role in achieving reliable cross-dataset performance.

Next Steps

- Apply batch correction to improve cross-dataset alignment.
- Extend experiments to more diverse datasets to further test the model's robustness.
- Train for more epochs with early stopping to better capture performance potential.
- Fine-tune the model on a small portion of the target dataset to improve cross-dataset generalization.

Code Availability

All code used for preprocessing, image transformation, model training, and evaluation was adapted from the official scDeepInsight GitHub repositories:

- https://github.com/shangruJia/scDeepInsight
- https://github.com/shangruJia/scDeepInsight-additional

Custom scripts were written to convert R-based datasets (.RDA) into .h5ad format for compatibility with the Python-based scDeepInsight pipeline. Additional modifications were made to support Apple M1 hardware, adjust training parameters, and generate evaluation metrics including PR curves, confusion matrices, and bar charts.

References

1. **Jia, et al.** (2023). *scDeepInsight: a supervised cell-type identification method for scRNA-seq data with deep learning.* https://doi.org/10.1093/bib/bbad266

2. **McKellar Dataset**: Walter et al., bioRxiv (2020)

3. **Oprescu Dataset**: Oprescu et al., iScience (2020)

4. **Python Libraries:**
    - scanpy, anndata, scikit-learn, PyTorch, torchvision, efficientnet_pytorch, matplotlib, seaborn, pandas, numpy, pickle, tqdm, rpy2, os

Acknowledgment

I used AI tools to assist me with report editing and general guidance throughout the project. All analysis and final decisions were completed independently by me.