**Methods**

**Data Source and Description**

This research analyses high-dimensional genomics data derived from the Fly Atlas transcriptomics dataset. The dataset includes gene expression profiles from the head and body cells of flies at various ages, focusing on understanding the patterns of aging. The dataset comprises 289,981 cells and 15,992 genes, where each gene represents a dimension in the high-dimensional space. The transcriptomics data is stored in an AnnData object, with the main data matrix .X containing gene expression measurements, and additional metadata stored in .obs and .var DataFrames. The .obs DataFrame contains variables such as tissue type, sex, and age, while .var holds gene-specific information.

**Pipeline Overview**

The analysis was conducted using a custom, highly adaptable pipeline implemented in Python, designed specifically to preprocess data, train models, and conduct evaluations on high-dimensional genomics data, including the Fly Atlas transcriptomics dataset (head and body cells). This pipeline is modular and configurable, allowing for extensive adjustments through a configuration file (config.py). The flexibility of the pipeline ensures that it can be easily adapted to different research needs with minimal changes.

Key aspects of the analysis, such as the choice of encoding variable (target variable), the size of training and test sets, model parameters, and data splitting for validation and testing, can all be modified via the configuration file. Once changes are made in the configuration file, they are automatically applied throughout the pipeline, streamlining the process and reducing the need for manual adjustments. Additionally, the pipeline allows for the possibility of bypassing certain steps to expedite processing, provided those steps have been executed in previous runs. However, critical steps like model visualization and evaluation are mandatory and cannot be skipped. This design allows the same codebase to be reused for various analyses, ensuring consistency and efficiency across different projects.

**Exploratory Data Analysis (EDA)**

Initial exploratory data analysis (EDA) was performed to ensure the integrity and completeness of the dataset. Duplicate rows in the .obs metadata were checked and confirmed absent to ensure the uniqueness of each cell. The dataset was verified to contain no missing values in the gene expression data .X, a common characteristic of transcriptomics data where zero values indicate no gene expression.

**Data Preprocessing**

The preprocessing pipeline was developed to manage the complexities of transcriptomics data, ensuring it is prepared for downstream machine learning applications. The following sections describe each step of the pipeline, which is designed to ensure reproducibility, consistency, and data integrity.

The process begins with a one-time stratified split of the original transcriptomics dataset. This split creates two subsets: an evaluation subset of 5,000 cells reserved for model assessment and a training subset of 284,981 cells for model development and analysis. Stratification preserves the original class distribution across both subsets, maintaining the biological diversity inherent in the dataset. These subsets are saved for future use, and the original dataset is archived to prevent unintended use in later analyses.

Upon executing the pipeline, the first operation selectively handles specific categories within the training dataset, guided by configuration settings. For example, categories like ‘mix’ can be excluded to focus the analysis on more relevant cell types or experimental conditions. By default, the ‘mix’ category is excluded due to its irrelevance to the primary research question.

A key feature of the pipeline is its ability to filter and analyse data by specific cell types, including CMS neuron, epithelial cell, fat cell, glial cell, muscle cell, and sensory neuron. This flexibility allows for targeted analyses, enhancing the relevance and specificity of the findings. When a particular cell type is selected, the pipeline filters the dataset accordingly, enabling more focused analyses aligned with research objectives.

The pipeline also supports training models on one tissue type and testing on another (e.g., training on head tissue and testing on body tissue), which is crucial for understanding model generalization across different tissues. Additionally, the pipeline allows for training and testing based on sex, with options to focus on male, female, or all samples. This flexibility enables exploration of sex-specific differences in gene expression and model performance.

Furthermore, the pipeline allows the selection of a specific number of genes and cells, with constraints to ensure that selections remain within available data. Users can specify the number of genes (variables) and cells (samples) for analysis, allowing for tailored data subsets that align with study objectives. If fewer genes or cells are selected than available, the pipeline samples the specified number accordingly.

The pipeline also balances gene counts between different gene types, such as autosomal and sex-linked genes, based on user-defined configuration parameters. This ensures gene selection aligns with study needs. Additionally, the pipeline offers the option to shuffle gene order, which can be useful for certain analyses or testing model robustness.

Following gene selection, the pipeline applies masks and filters to refine the dataset further. Masks are created for sex-linked and autosomal genes and LncRNA genes which allows filters to include or exclude different gene types. If directed to remove unaccounted genes, the pipeline retains only those that have been identified.

The raw transcriptomics data is stored in sparse matrix formats, which the pipeline converts to dense formats for compatibility with one-dimensional Convolutional Neural Networks (1D CNNs). For CNNs, the data is transformed from a two-dimensional to a three-dimensional format, ensuring compatibility with CNN architectures and preventing potential dimensionality errors during model training.

The training subset undergoes a further stratified split into training and testing subsets, preserving class distribution to ensure the test set remains representative of the overall dataset. This stratification is crucial for unbiased model evaluation during later stages of the analysis.

To handle categorical labels within the dataset, the pipeline uses a label encoding process. Labels in the training data are encoded using the fit\_transform method, while the transform method is applied to the test labels. This consistency in label encoding is vital for effective processing of categorical variables by the machine learning models.

The pipeline includes the capability to standardize and normalize the data using a StandardScaler.

An optional feature involves selecting highly variable genes, which can focus the analysis on the most informative features. When enabled, this process normalizes the data, applies a logarithmic transformation, and identifies the top 5,000 most variable genes, ensuring that subsequent analyses concentrate on genes exhibiting the most significant variation.

The pipeline also offers mechanisms for handling batch effects, common in large-scale transcriptomics studies. Batch effect correction is performed as needed, with features selected from batch-corrected data to mitigate systematic biases. This step is essential for improving the accuracy and reliability of the machine learning models.

Upon completing preprocessing, the processed data and label encoder are saved separately for future use to prevent data leakage, thus maintaining model integrity. The pipeline also includes functionality to load previously saved pre-processed data and associated label encoders in subsequent runs, ensuring consistency and reproducibility across different analyses.

For evaluating pre-trained models, the pipeline applies the same preprocessing steps to the evaluation dataset as were applied to the training data. This includes using the same label encoder to transform the evaluation data labels, ensuring uniformity in the model’s input and preventing discrepancies during evaluation.

**Model Architecture and Training**

The Convolutional Neural Network (CNN) architecture used in this study is designed to process high-dimensional transcriptomics data efficiently, capturing complex patterns within gene expression profiles. The model is built using a sequence of convolutional blocks followed by fully connected layers, each tailored to extract and refine features from the input data. Below is a breakdown of the model’s architecture:

**Convolutional Block 1:**

This initial block is crucial for the early stages of feature extraction from the input data. It consists of the following layers:

* **Conv1D Layer**: The first layer in the block, configured with a kernel size of 3, a stride of 1, and ‘same’ padding. This layer applies 32 filters to the input data, extracting local features across the gene expression profiles.
* **Batch Normalization**: This layer standardizes the outputs of the Conv1D layer, stabilizing and speeding up the training process by reducing internal covariate shifts.
* **ReLU Activation Function**: The ReLU (Rectified Linear Unit) introduces non-linearity to the model, enabling it to learn and represent complex patterns within the data.
* **Max Pooling Layer**: The max pooling layer, with a pool size of 2 and a stride of 2, reduces the spatial dimensionality of the output from the Conv1D layer, effectively down sampling the feature map and retaining the most significant features.

**Convolutional Block 2:**

This block mirrors the structure of the first block, continuing the process of feature extraction and refinement:

* **Conv1D Layer**: Like Block 1, this layer applies 64 filters, using the same kernel size, stride, and padding. The increase in the number of filters allows the model to capture more complex features from the data.
* **Batch Normalization**: Again, this layer normalizes the output, contributing to the stability and efficiency of the training process.
* **ReLU Activation Function**: The non-linearity introduced by ReLU helps the model to handle the more complex features being extracted.
* **Max Pooling Layer**: This pooling layer further reduces the dimensionality, ensuring that the model remains computationally efficient while retaining key features.

**Convolutional Block 3:**

The third convolutional block is designed to retain more feature information by excluding the pooling layer:

* **Conv1D Layer**: This layer applies 128 filters, with the same kernel size, stride, and padding as the previous layers. This increase in filters allows the model to capture even more detailed and complex patterns.
* **Batch Normalization**: As with the previous blocks, batch normalization is applied to stabilize the output.
* **ReLU Activation Function**: The ReLU activation continues to introduce non-linearity, preparing the feature maps for the fully connected layers that follow.
* Notably, this block does not include a max pooling layer, allowing the model to maintain the spatial resolution of the feature maps, which can enhance the model’s performance in the later stages.

**Flattening Layer:**

Following the convolutional blocks, the output, which is still in a multi-dimensional form, is flattened into a one-dimensional array. This flattening process converts the spatially structured data into a format that can be fed into the fully connected layers, facilitating the final stages of pattern recognition and decision-making by the model.

**Fully Connected Layers:**

The flattened output is then processed through a series of fully connected (dense) layers:

* **Dense Layer 1**: This layer consists of 128 units, each performing a non-linear transformation of the input using a ReLU activation function. This layer begins the process of synthesizing the high-level features extracted by the convolutional layers.
* **Dropout Layer**: To prevent overfitting, a dropout layer with a rate of 0.5 is applied, randomly setting 50% of the input units to zero during training. This regularization technique helps the model generalize better to new, unseen data.
* **Dense Layer 2**: The second dense layer contains 64 units, further refining the feature representation. It also uses a ReLU activation function, allowing the model to learn more intricate patterns in the data.
* **Dropout Layer**: Another dropout operation is applied after this layer, again with a rate of 0.5, to ensure the model’s robustness.

**Output Layer:**

The final layer of the model is a dense layer that uses a SoftMax activation function. The number of units in this layer corresponds to the number of classes in the classification task. The SoftMax function converts the outputs into probabilities, with each unit representing the probability of the input data belonging to a particular class.

The model was compiled using the Adam optimizer with a learning rate of 0.001, categorical cross-entropy as the loss function, and evaluation metrics including accuracy and Area Under the Curve (AUC).

**Custom Checkpointing and Training Configuration**

To enhance the training process, the model training pipeline incorporates a custom checkpointing class called CustomModelCheckpoint. This class extends TensorFlow’s ModelCheckpoint callback and adds additional functionalities, including:

* Saving the model weights only if the current epoch results in the lowest validation loss observed so far, preventing the saving of suboptimal models.
* Storing other important aspects of the training process, such as the label encoder, model history, explainer, reference data, and scalers, to ensure that all necessary components are saved alongside the model.
* This approach ensures that the best-performing model is available for evaluation and that the model’s environment can be recreated precisely for future runs or additional analyses.

**Training Procedure**

The training process was configured as follows:

**Validation and Test Splitting**: The dataset was divided into training, validation, and test sets with proportions of 80%, 10%, and 10%, respectively. The split was performed with a fixed random seed for reproducibility.

**Training Parameters**:

* **Epochs**: The model was trained for up to 15 epochs, with early stopping applied if the validation loss did not improve for 5 consecutive epochs.
* **Batch Size**: A batch size of 250 was used, balancing the need for efficient computation with the stability of gradient estimates.
* **Early Stopping**: Early stopping was implemented with a patience of 5 epochs, which helps prevent overfitting by halting training when the validation performance plateaus.
* **Data Augmentation and GPU Acceleration**: Data augmentation techniques were applied during training to enhance the model’s generalization capability. Additionally, GPU acceleration was utilized to speed up the training process, particularly important given the high dimensionality and large size of the dataset.

**Baseline Models**

For comparative purposes, several baseline models were also trained using the same dataset and preprocessing pipeline:

1. **Multi-Layer Perceptron (MLP)**: A fully connected neural network that serves as a strong baseline model. The MLP includes several hidden layers with ReLU activation functions and dropout for regularization. This model is simpler than the CNN but still capable of capturing non-linear patterns in the data.

2. **Logistic Regression**: A linear model serving as a simple baseline. Despite its simplicity, logistic regression is often effective for high-dimensional data and provides a benchmark for more complex models.

3. **Random Forest**: An ensemble model that builds multiple decision trees and aggregates their predictions. Random forests are known for their robustness and ability to handle noisy data.

4. **Support Vector Machine (SVM)**: A powerful classifier that finds the hyperplane which best separates the classes in the feature space. SVMs are particularly useful in cases with high-dimensional data and are often used as a baseline in transcriptomics studies.

These models provided benchmarks against which the performance of the CNN was compared.

**Model Evaluation and Interpretation**

Post-training, the pipeline triggers a visualize function that generates visualizations of the model’s performance and conducts a comprehensive evaluation using metrics such as F1 score, recall, precision, and AUC. Model interpretability was further enhanced using SHAP (SHapley Additive exPlanations) to identify the contribution of each gene to the model’s predictions, providing insights into the biological relevance of the features.