2023
S1

# DATA3888 - Research on Classification of Frozen Mouse Brain Cell Images Using Deep Learning Models

IMAGE 10

SID: 500273300, 500462616, 500177868, 510014087, 510407902, 490065813

# Table of Contents



*Words count 2784 – From Executive Summary to 6. Shiny App Deployment*

IMAGE 10

# Executive Summary

This report presents a deep learning project classifying mouse brain cell images from the "Fresh Frozen Mouse Brain for Xenium Explorer Demo" dataset using two models: ResNet152 V2 and Vision Transformer (ViT-B/32).

Our findings indicated that processed image data performed better on models than raw data, and optimizing the ViT model and adding a separate test dataset improved model performance. The stratified splitting method proved to be more rational than the independent method for handling our dataset, and by reducing the number of clusters, we enhanced our model's ability to distinguish between different groups.

Although ResNet150 V2 achieved the highest accuracy (42.21%) with an independent splitting method on seven clusters, we recommend the ViT-B/32 model with a stratified splitting method for its superior generalizability (17.91% accuracy) across 28 clusters.

The project encountered limitations, including computational constraints and data quality issues, suggesting future opportunities for improvement. In the future, we aim to apply our model to other animal cell images and minimize data leakage for better model evaluation.

The report highlights the ViT-B/32 model with stratified splitting as the optimal choice due to its greater generalizability, which provides valuable insights into classifying fresh images of mouse brain cells. It demonstrates the potential of deep learning models in cell image classification and contributes to ongoing efforts in neuroscience to understand the intricacies of the brain better. Ultimately, the methodology and results of our project could potentially inform future research and treatments in neuroscience.

# 1. Background

This section describes the motivation, data sets, data formats and the data processing for this project.

## 1.1 Motivation

For the cell images of the fresh mouse brain, we aim to find the model best classifies images based on a large number of clusters. Based on two core research questions:

1. How to improve the accuracy of the model?
2. How to make our models better generalized?

Develop an interactive communication tool - Shiny app, to show our experiment, in which we compare our models and create various visualizations to show our findings.

## 1.2 Data Sets

The dataset in focus is acquired from the "Fresh Frozen Mouse Brain for Xenium Explorer Demo" dataset, retrieved from the 10x Genomics website. This comprehensive dataset generates various data types, including cell morphology images, cell boundaries, and RNA abundances (gene expression), all classified into 28 distinct clusters. The tiny subset of this comprehensive dataset is specifically utilized for the current analysis.

## 1.3 Data Formats

1. The cell morphology data is in the form of raw image data.
2. Cell boundary information is provided as a .csv file, containing vertices outlining each cell's boundary.
3. RNA abundances or gene expression data is grouped into 28 clusters.

## 1.4 Data Processing

The processing of this dataset is undertaken primarily using the R programming language, with a specific focus on the cell morphology image data and cell segmentation data. The different phases of the data processing pipeline include:

1. **Reading and Conversion:** Raw cell morphology images are read and converted into a tif format for better compatibility. Similarly, the cell boundaries and cluster data are read into the R environment.
2. **Normalization and Extraction:** The intensities of these images are scaled or normalized according to the 99th percentile of the intensity distribution. Subsequently, boundary vertices and cluster values are extracted for each cell in the dataset.
3. **Image Organization:** The images of cells are organized by their respective cluster and saved in distinct folders as png files.
4. **Processed Functions:** This phase involves two key functions in R: 'get_inside' and 'mask_resize.' These functions filter and rescale cell boundaries, identify interior and exterior cell pixels, create a new image of the interior pixels, mask the original image, and resize it to the required dimensions.
5. **Image Resizing and Masking:** In this phase, the 'cluster_images' function in R resizes the cluster images to a standard size of 224x224 pixels. The code then reads cell boundaries, generates clean image directories, calculates cell centroids, and applies the 'get_inside' and 'mask_resize' functions to each image. The resultant cleaned, masked, and resized images are saved in the respective clean cluster folders.

Thus, the processed dataset contains 36,553 cell images resized and masked to a uniform dimension, each associated with a cluster label, resulting from grouping gene expression data into 28 distinct clusters. Fig.1 shows an example of how the image data was generated and processed.

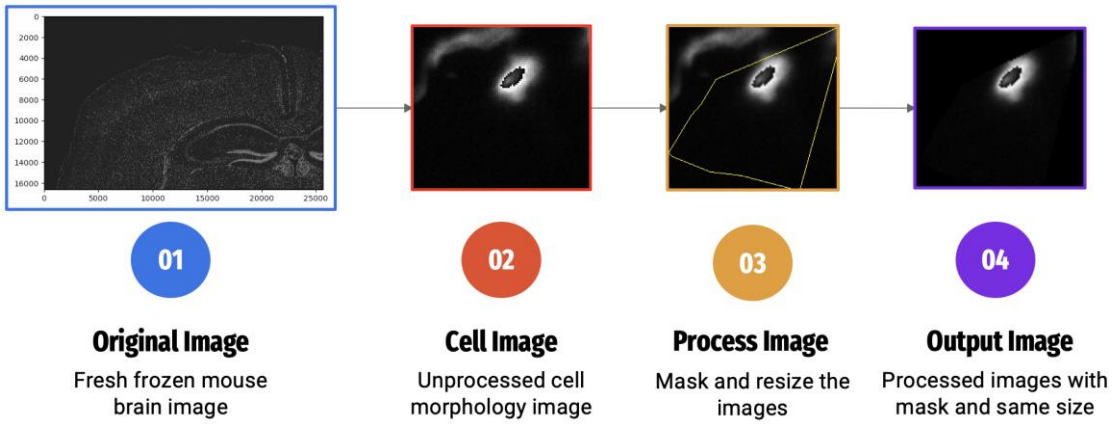


*Fig. 1. Process Char of Image Data Generated and Processed*

## 1.5 Complete Experimental Procedure

This section delineates the comprehensive experimental process, encompassing preparation, methodology implementation, analysis, conclusion formulation, and deployment of our findings.

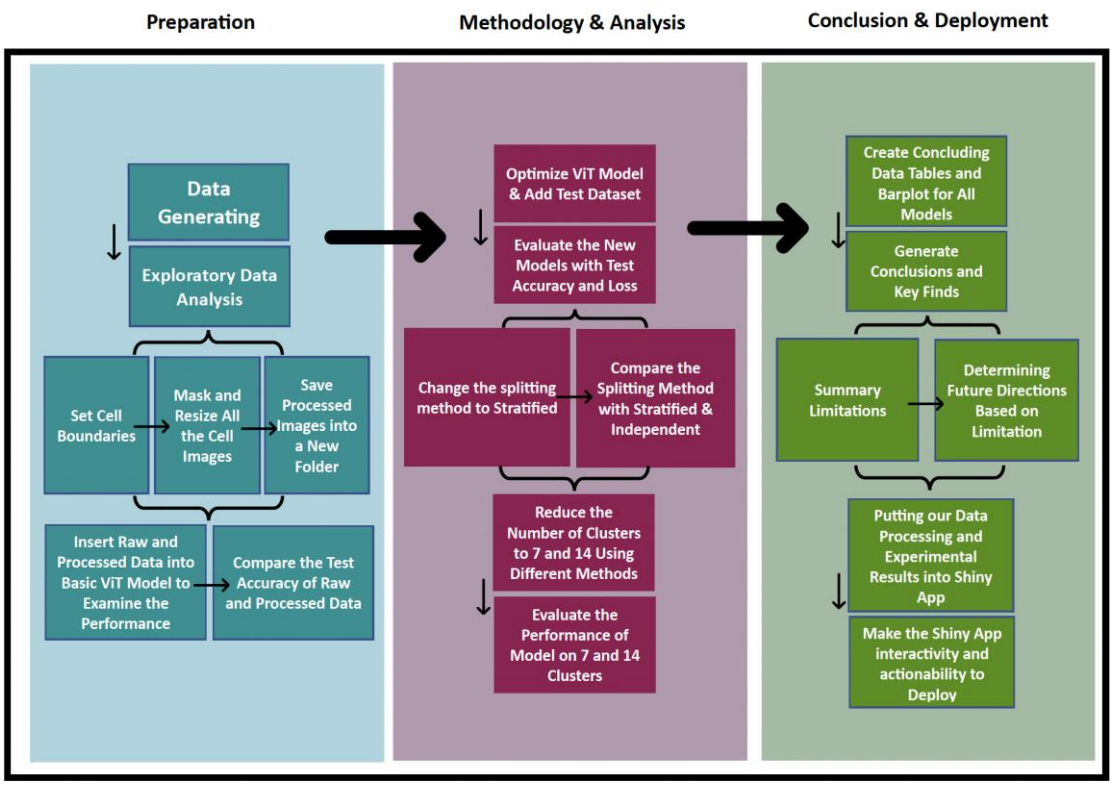


*Fig. 2. Complete Experimental Procedure*

# 2. Methodology

This section describes the experimental environment, deep learning models and experiment procedure.

## 2.1 Experimental Environment

All experiments were conducted on the Matpool platform with a computer configuration of CPU: 12× Xeon Platinum 8260, GPU: NVIDIA GeForce RTX 3090.

## 2.2 Developing Deep Learning Models

### 2.2.1 ResNet Model

The ResNet152 V2 model is a pre-trained variant optimized for transfer learning and fine-tuned on our dataset. It leverages the pre-trained weights from the ImageNet dataset and focuses on fine-tuning the top 10 layers, repurposing them for our specific image classification task. The architecture is supplemented with a classification head comprising a global average pooling layer and two dense layers with ReLU and softmax activations designed to classify images into 28 classes. The model uses the categorical cross-entropy loss function, SGD optimizer, with a learning rate of 0.01 and momentum of 0.9. The choice of the ResNet152 V2 model, the deepest ResNet model, balances computational resources and time while achieving high-quality performance, particularly for our complex dataset comprising numerous gene images and clusters.

### 2.2.2 ViT Model

We use the pre-trained ViT-B/32 model, renowned for its high performance in computer vision tasks. It incorporates 12 Transformer layers and treats image patches as sequences for processing. Compiled with an Adam optimizer and a categorical cross-entropy loss function, label and one-hot encoding methods are applied to class labels, and image data is normalized. Its selection is informed by high computational costs and time demands of training models from scratch, especially on our large dataset. With its pre-training on a large dataset allowing it to learn a broad range of features, the deep ViT-B/32 model can generalize better and converge

faster, making it advantageous for our high-dimensional data, such as our 28 clusters, where a shallow model might not capture the complexity sufficiently.

## 2.3 Experimental Procedure

This section describes how we do the experiment base on our core research question.

### 2.3.1 Use the Basic Model to Examine the Performance of Processed Image Data

In the initial phase of our experiment, we employed a basic Vision Transformer (ViT) model, precisely the ViT-B/32 configuration, to evaluate the performance of image data in both raw and processed forms. The model was set up with an image size of 224x224. The ViT-B/32 model was compiled with an Adam optimizer and a categorical cross-entropy loss function. We then performed a train-test split to 80:20 on the entire dataset, converting labels into a categorical one-hot encoding format and normalizing the image data.

Regarding model training, we ran the model for 20 epochs with a batch size of 128 and used the test dataset for validation. The model's performance was then evaluated based on its accuracy on the test data.

### 2.3.2 Optimize ViT Model & Add Test Dataset

In the next phase of our experiment, we took additional measures to optimize the Vision Transformer (ViT) model and introduced a separate test dataset for enhanced validation. We continued utilizing the processed images but incorporated an ImageDataGenerator for data augmentation, including rotation, width and height shift, shear transformation, zoom, and horizontal flip. These manipulations aimed to develop a more resilient model capable of generalizing effectively by learning from various transformations of the same data.

We further introduced a separate test dataset that offered a more rigorous evaluation of the model's performance on unseen data, thus providing a more realistic measure of its effectiveness. All the training, validation, and test datasets were subsequently normalized with a rate of 60:20:20. An ExponentialDecay learning rate scheduler was introduced for the Adam optimizer, which gradually adjusted the learning rate over time, permitting a rapid start followed by a slow deceleration of learning.

We amplified the number of training epochs to 50, maintaining a batch size of 128. The model was trained using the augmented training data and the validation dataset. Furthermore, we introduced a callback for early stopping based on the validation loss, with patience set for five epochs, which meant that the training would be halted if the validation loss did not improve for five successive epochs. Moreover, the early stopping callback restored the model weights from the epoch that produced the best validation loss.

### 2.3.3 Change the Independent to Stratified Split Method

The data stratification was undertaken to mitigate the issue of imbalanced distribution within the image dataset, a problem discovered during the Exploratory Data Analysis phase. Independent splitting datasets, where the complete dataset is divided randomly into a training and testing set, could fail to represent the entire dataset adequately and miss numerous labels. This could lead to poor generalization and unreliable performance estimates [1]. Given our dataset's 28 clusters with highly varied image counts, we decided that stratified splitting was more appropriate than independent splitting. This is particularly important in cases where our data is unbalanced. Thus, we split each of the 28 clusters into training, validation, and testing sets. This method ensured that each training, validation, and test set had a similar proportion of images from each cluster, allowing the model's performance to be evaluated on a representative sample of the entire dataset rather than a skewed subset.

### 2.3.4 Reduce the Number of Clusters

To improve the performance of our model, we identified that some of our image clusters were quite similar, potentially impeding the model's capacity to accurately distinguish between them. To tackle this issue, we employed two strategies to reduce the number of clusters, with the aim

of enhancing model performance by concentrating on the most distinguishable clusters. The first strategy involved selecting seven clusters (2, 4, 5, 7, 9, 10, and 13) based on their distinctness as visualized using UMAP (Uniform Manifold Approximation and Projection), a dimensionality reduction technique. The second approach involved the use of the confusion matrix from our ViT model, which was trained to classify the original 28 clusters. By observing several columns with only zeros, we concluded that images from those clusters might not have been included in the validation and testing sets, and thus these clusters were excluded, leaving us with a refined group of 14 clusters. The UMAP visualization and confusion matrix can be found in Appendix A.

# 3. Results

The results section contains accuracy and loss metrics from multiple model configurations. Our experiments focused on three primary dimensions: the type of model used (ViT or ResNet), the splitting method (independent or stratified), and the number of clusters (7, 14, or 28). Each model configuration's test accuracy and loss are detailed in the following table and barplot (Fig.2).

| Model | Test Accuracy | Test Loss |
|---|---|---|
| Vit Independent 7 | 37.49% | 1.59235 |
| Vit Independent 14 | 22.47% | 2.25526 |
| Vit Independent 28 | 18.22% | 2.84296 |
| Vit Stratified 7 | 36.47% | 1.64247 |
| Vit Stratified 14 | 22.01% | 2.25338 |
| Vit Stratified 28 | 17.91% | 2.85318 |
| RestNet Independent 7 | 42.21% | 1.49551 |
| RestNet Independent 14 | 25.96% | 2.14797 |
| RestNet Independent 28 | 18.96% | 2.79140 |
| ResNet Stratified 7 | 38.28% | 1.62378 |
| ResNet Stratified 14 | 19.76% | 2.35292 |
| ResNet Stratified 28 | 15.95% | 2.95156 |

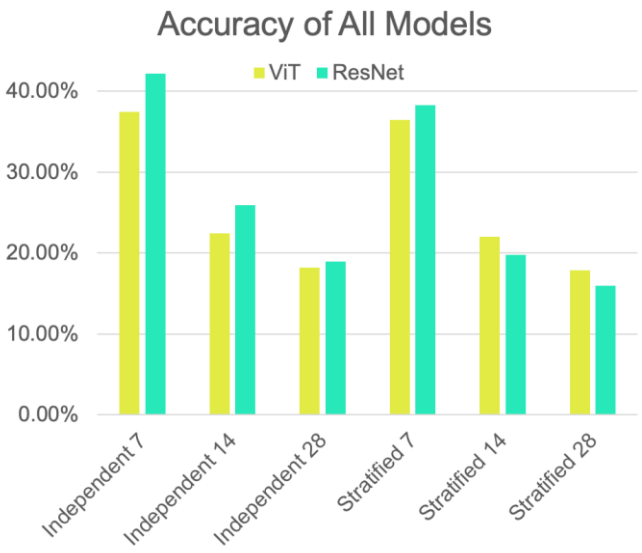


*Fig. 3. Concluding Data Tables and Barplot for All Models*

# 4. Discussion

This section discusses the questions for our experimental procedure based on the results.

## 4.1 Does Processed Data Perform Better on Models?

We compare the performance of raw and processed image datasets by running a basic ViT model on each dataset, which yielded a test accuracy of 0.14982911944389343 for the raw dataset and 0.161810964345932 for the processed dataset. The processed dataset showed a slight improvement. This led to the decision to utilize the processed image dataset for subsequent experiments in the study. The accuracy plots of the model mentioned above are shown in Appendix B, which show that test accuracy of the processed dataset is higher than that of the raw dataset.

## 4.2 Does Optimising the ViT model & Adding Test Dataset Perform Better?

Substant benefits were reaped in optimizing the ViT model and incorporating a separate test dataset. Refining the ViT model and adding the test dataset facilitated a more precise evaluation and validation of the model's performance, yielding improved results and insights.

Upon completion of training, the best model was loaded and assessed on the independent test dataset. The model's accuracy on this test set showed significant improvement, attaining a value of 0.18219122290611267. This marked a promising advancement from our prior experiments. The progression of accuracy throughout the training phase is displayed in Fig.4, which shows a significant enhancement on the test accuracy.

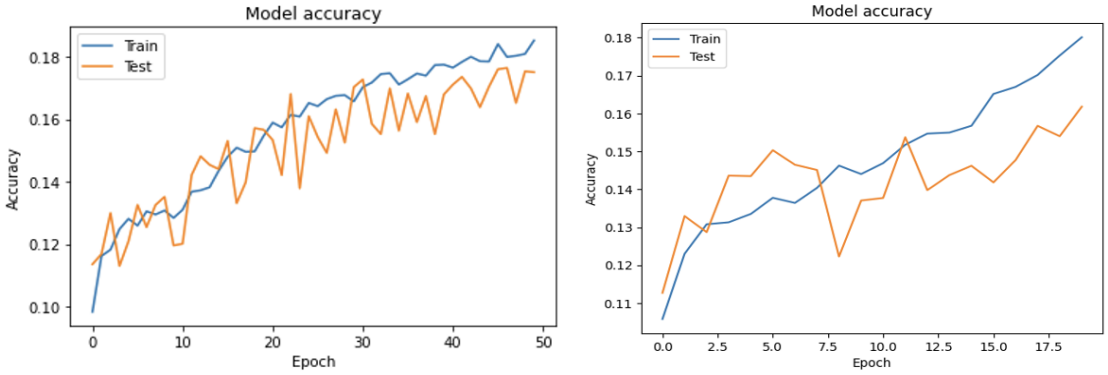


*Fig. 4. Accuracy Plot of Optimized ViT Model (Left) & Basic ViT Model (Right)*

## 4.3 Which Splitting Method is More Reasonable: Independent or Stratified?

As Fig.1 in the Results section shows, despite slightly lower testing accuracy, the stratified method provides a more accurate representation of the model's performance. The independent splitting method may show a higher testing accuracy. However, it carries a risk of overfitting to specific clusters or classes, which could result in poor generalization to new data. On the other hand, the stratified splitting method prevents overfitting by ensuring that all clusters are represented in the training, validation, and test datasets. It gives a more realistic measure of how the model might perform on unseen data. Given these factors, especially the imbalanced nature of our data, we have decided to continue using the stratified method.

## 4.4 Are the Cluster Reduction Procedures Effective?

The effectiveness of the implemented cluster reduction procedures is reflected in our results. We designed these strategies to concentrate on the most distinct clusters, addressing potential issues resulting from high similarity among specific clusters. Consequently, there was a noticeable improvement in the test accuracy of our models. Those trained on the refined group of 14 clusters achieved accuracy rates between 19.76% and 25.96%, while the models trained on the carefully selected 7 clusters displayed even better performance, with accuracy rates ranging from 36.47% to 42.21%. Hence, by focusing on the more distinct clusters, we successfully improved our model's ability to differentiate between groups. The confusion matrix of the stratified splitting ViT model on 7 clusters shows a better generalization than that of 14 clusters, which can be found in Appendix C.

# 5. Conclusion

This section describes the key finds, conclusion, limitations, and future direction.

## 5.1 Key Findings

1. The application of the data processing method outlined in section 2.4 has demonstrated enhanced model performance compared to using raw data.
2. Adding a test dataset and optimizing the ViT model have significantly contributed to more precise evaluations and improved model performance.
3. The stratified splitting method has proven to be a more rational choice for handling our particular dataset.
4. By focusing on the more distinct clusters (reducing the number of clusters), we have effectively enhanced our model's ability to distinguish between different groups.

## 5.2 Conclusion

The highest accuracy of 42.21% was achieved by the ResNet150 V2 model utilizing the independent splitting method on 7 clusters. However, our study's insights suggest the ViT-B/32 model using a stratified splitting method as the optimal candidate for classifying 28 clusters. This approach exhibits superior generalizability, making it the most reasonable choice in our project's context.

## 5.3 Limitation & Future Direction

The following limitations warrant future improvement:

1. **Computational Constraints:** Our project was restricted by available computational resources. With increased CPU and GPU power, we could employ methods like cross-validation or repeated cross-validation and utilize deeper models on dataset of full coronal section rather than tiny subset.
2. **Data Quality:** There is room for enhancing data quality, particularly by ensuring a balanced distribution of images across each cluster.
3. **Scope of Application:** Our research is confined to frozen mouse brain cell images. Expanding the model's applicability to other animals' cell images could be an avenue for future exploration.
4. **Potential Data Leakage:** As we only had a single dataset, we split it into training, validation, and testing sets. This arrangement could potentially lead to data leakage, impacting the objective evaluation of the model's generalization ability.

# 6. Shiny App Deployment

The shiny app provides visualizations of our experiment process and a comparison of our models. The EDA page demonstrates the resizing and cleaning of images by selecting different clusters. The main page shows our 12 models that can be selected based on different conditions on the left side. Users can then examine the model's performance on the right side in greater detail. The Summary page compares the final accuracy of the models generated by 12 methods. This feature is helpful for users as it allows them to compare the performance of each model. Additionally, all graphs in our shiny app are interactive, providing users with detailed data exploration capabilities. To access the completed shiny app, please click the link: https://github.sydney.edu.au/RPOR2575/DATA3888. Below is the shiny app introductory chart (Fig.5).

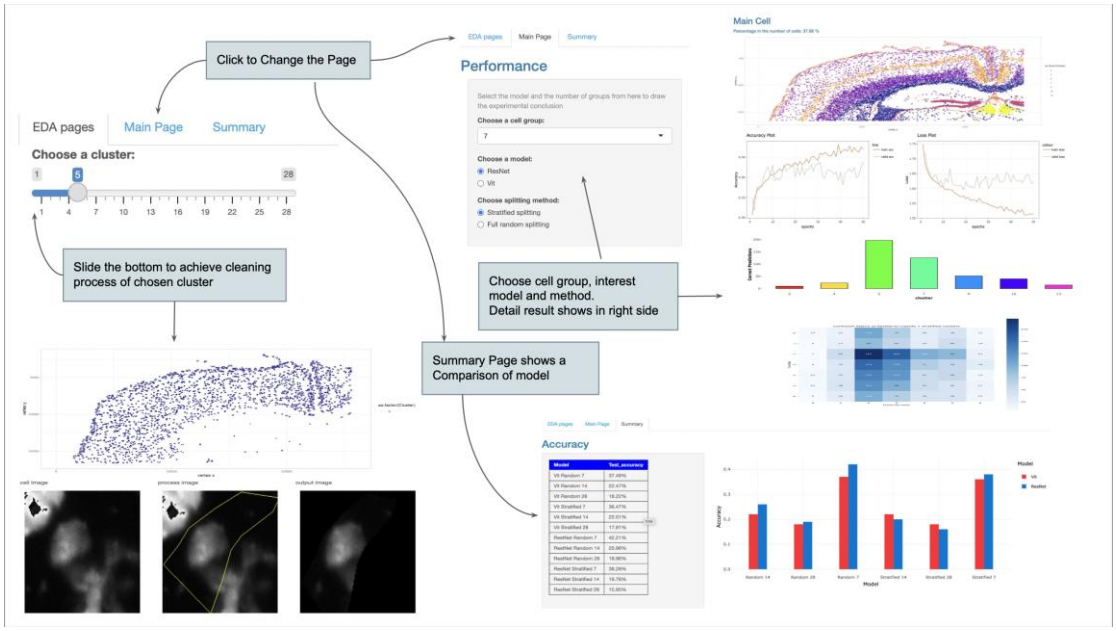


*Fig. 5. Shiny App Introductory Chart*

# Student contributions

1. **Generating data:** Guan Gui
2. **Tuning machine:** Guan Gui, Qiuze He
3. **Performing processing:** Guan Gui, Qiuze He
4. **Writing the code:** Guan Gui, Qiuze He, Xuwei Lu
5. **Making observations:** Guan Gui, Qiuze He
6. **Deploying final product:** Xuwei Lu, Hanzhang Peng, Yuni Wang, Guan Gui, Qiuze He
7. **Writing report:** Guan Gui
8. **Editing report:** Qiuze He, Ruby Portrate, Hanzhang Peng

# Reference


[1] M. Merrillees and L. Du, "Stratified Sampling for Extreme Multi-Label Data," eprint arXiv:2103.03494, March 2021.

[2] 10x Genomics, "Xenium V1 FF Mouse Brain Coronal Subset CTX HP Analysis Summary," 10x Genomics, 21 October 2022. Available: https://cf.10xgenomics.com/samples/xenium/1.0.2/Xenium_V1_FF_Mouse_Brain_Coronal_Subset_CTX_HP/Xenium_V1_FF_Mouse_Brain_Coronal_Subset_CTX_HP_analysis_summary.html.


# Appendix

**Appendix A – Evidence to Reduce Number of Clusters: UMAP Plot and Confusion Matrix**

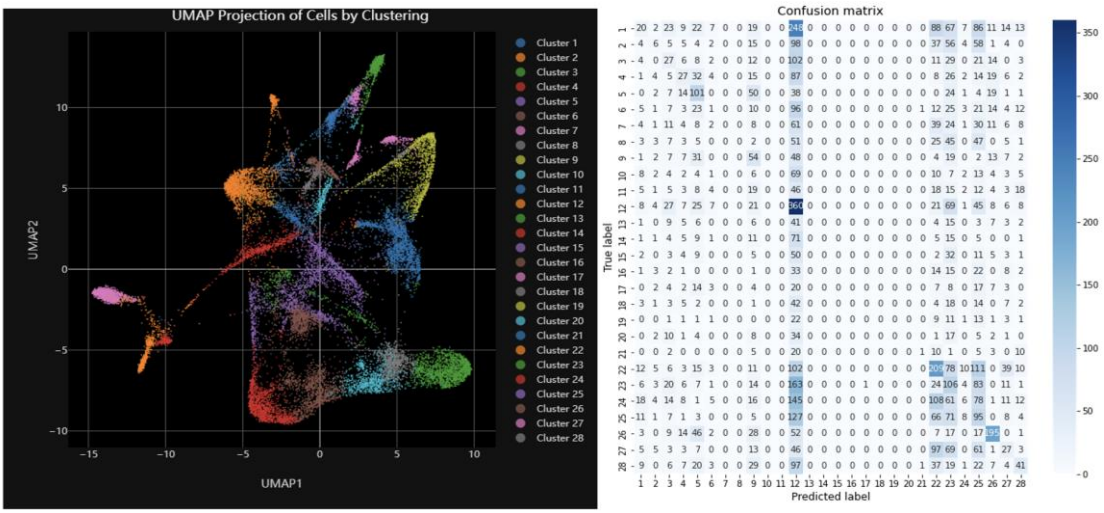


*Fig. Appendix A. UMAP Plot [2] (Left) and Confusion Matrix (Right)*

**Appendix B – Comparison of Basic ViT Model on Raw Dataset & Processed Dataset**

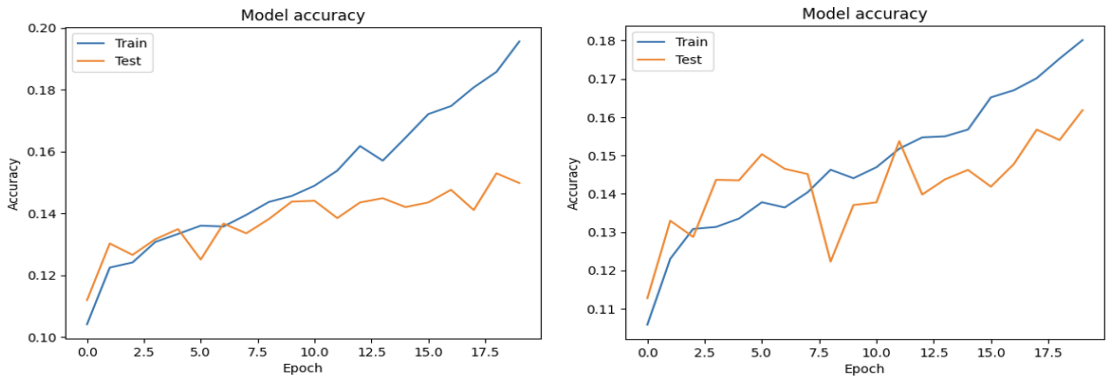


*Fig. Appendix B. Accuracy Plot of Basic ViT Model on Raw Dataset (Left) & Processed Dataset (Right)*

**Appendix C – Confusion Matrix of the Stratified Splitting ViT Model on the 7 Clusters and 14 Clusters**

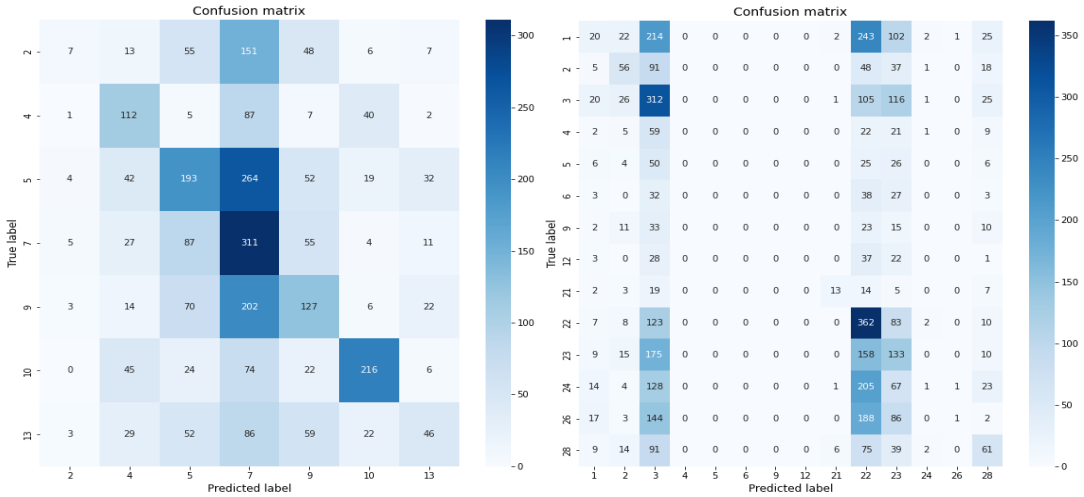


*Fig. Appendix C. Confusion Matrix with 7 Clusters (Left) and 14 Clusters (Right)*

**Appendix D – To Access All the Code Relevant to Report with Full Details, Please Click the Link and Read the README.md:** https://github.sydney.edu.au/RPOR2575/DATA3888.

**Appendix E – Code Example: ViT-B/32 Model Using a Stratified Splitting Method to Classify 28 Clusters & Generate Accuracy and Loss Plots & Generate Confusion Matrix**

*# Image pre-processing, Vision Transformer model creation, training, and evaluation.*

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import os
import cv2
import re
import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from vit_keras import vit
from tensorflow.keras.optimizers import Adadelta
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix

# Define a function to create a Vision Transformer model
def create_vit_model(image_size, num_classes):
    vit_model = vit.vit_b32(
        image_size=image_size,
        activation='softmax',
        pretrained=False,
        include_top=True,
        pretrained_top=False,
        classes=num_classes
```

```
    )
    return vit_model

# Preprocessing the data
data_dir = 'full_clean_image_224/'
cluster_numbers = list(range(1, 29))  # Update cluster_numbers
cluster_folders = [os.path.join(data_dir, f'cluster_{i}') for i in cluster_numbers]

# Data preprocessing loop
train_images, val_images, test_images = [], [], []
train_labels, val_labels, test_labels = [], [], []

# Loop over the cluster folders
for folder in cluster_folders:
    cluster_images = []
    cluster_labels = []

    file_names = os.listdir(folder)
    for file_name in file_names:
        cell_id = int(file_name.split('_')[1].split('.')[0])
        img = cv2.imread(os.path.join(folder, file_name))

        cluster_images.append(img)
        cluster_labels.append(f'cluster_{folder.split("_")[-1]}')

    # Perform train-val-test split for each cluster
    cluster_train_images, cluster_test_images, cluster_train_labels, cluster_test_labels = train_test_split(
        cluster_images, cluster_labels, test_size=0.2, random_state=42
    )

    # Further split the train set into train and validation sets for each cluster
    cluster_train_images, cluster_val_images, cluster_train_labels, cluster_val_labels = train_test_split(
        cluster_train_images, cluster_train_labels, test_size=0.25, random_state=42
    ) # 0.25 x 0.8 = 0.2

    # Merge train-val-test sets of each cluster
    train_images.extend(cluster_train_images)
    val_images.extend(cluster_val_images)
    test_images.extend(cluster_test_images)
    train_labels.extend(cluster_train_labels)
    val_labels.extend(cluster_val_labels)
    test_labels.extend(cluster_test_labels)

# Convert to numpy arrays
train_images = np.array(train_images)
```

```
val_images = np.array(val_images)
test_images = np.array(test_images)
train_labels = np.array(train_labels)
val_labels = np.array(val_labels)
test_labels = np.array(test_labels)

# Initializing and fitting a label encoder
lb = sklearn.preprocessing.LabelEncoder()
train_labels = lb.fit_transform(train_labels)
test_labels = lb.transform(test_labels)
val_labels = lb.transform(val_labels)

# Transforming labels to categorical and normalizing images
val_labels = tf.keras.utils.to_categorical(val_labels, num_classes=len(cluster_numbers))
val_images = val_images.astype('float32') / 255.0
train_labels = tf.keras.utils.to_categorical(train_labels, num_classes=len(cluster_numbers))
test_labels = tf.keras.utils.to_categorical(test_labels, num_classes=len(cluster_numbers))
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
image_size = 224
num_classes = len(cluster_numbers)

# Creating a Vision Transformer model
model = create_vit_model(image_size, num_classes)

# Defining learning rate schedule and optimizer
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-4,
    decay_steps=10000,
    decay_rate=0.9
)
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

# Compiling the model
model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

epochs = 50
batch_size = 128

# Defining data augmentation
data_augmentation = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
```

```
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Creating train generator
train_generator = data_augmentation.flow(train_images, train_labels,
    batch_size=batch_size)

# Setting up early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Create a folder to save the best model
model_save_folder = "saved_models_addtest_percluster"
os.makedirs(model_save_folder, exist_ok=True)

# Configure the ModelCheckpoint callback
model_checkpoint = ModelCheckpoint(
    os.path.join(model_save_folder, "best_model_28_b32_percluster.h5
"),
    monitor='val_loss',
    save_best_only=True,
    mode='min',
    verbose=1
)

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_images) // batch_size,
    epochs=epochs,
    validation_data=(val_images, val_labels),
    callbacks=[model_checkpoint]
)

# Load the best model
best_model = tf.keras.models.load_model(os.path.join(model_save_fold
er, "best_model_28_b32_percluster.h5"))

# Evaluate the best model on test data
loss, accuracy = best_model.evaluate(test_images, test_labels)
print(f"Test loss: {loss}, Test accuracy: {accuracy}")
```

*# Create accuracy and loss plot*

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
```

```
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

*# Create confusion matrix*

```
# Get the predictions for the test set
y_pred = model.predict(test_images)

# Convert the predictions from one-hot encoding to labels
y_pred_labels = lb.inverse_transform(np.argmax(y_pred, axis=1))
y_true_labels = lb.inverse_transform(np.argmax(test_labels, axis=1))

# Get the confusion matrix
cm = confusion_matrix(y_true_labels, y_pred_labels)

# Set the figure size
plt.figure(figsize=(10, 10))

# Plot the confusion matrix using seaborn
sns.heatmap(cm, annot=True, cmap="Blues", xticklabels=cluster_number
s, yticklabels=cluster_numbers, fmt="d")

# Set the font size of the axis labels and annotations
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.xlabel('Predicted label', fontsize=12)
plt.ylabel('True label', fontsize=12)
plt.title('Confusion matrix', fontsize=14)

plt.show()
```