May 12, 2025

RNA-seq Data Generation and 1D-CNN Classification Workflow for Visual Impairment Research

DOI
https://dx.doi.org/10.17504/protocols.io.6qpvrqz2zlmk/v1

Dr. Ashit Kumar Dutta[1], Nasser Ali AlJarallah[1], Abdul Rahaman Wahab Sait[2]

[1]Almaarefa University; [2]King Faisal University

Ashit Kumar
AlMaarefa University

DOI: https://dx.doi.org/10.17504/protocols.io.6qpvrqz2zlmk/v1

Protocol Citation: Dr. Ashit Kumar Dutta, Nasser Ali AlJarallah, Abdul Rahaman Wahab Sait 2025. RNA-seq Data Generation and 1D-CNN Classification Workflow for Visual Impairment Research. **protocols.io** https://dx.doi.org/10.17504/protocols.io.6qpvrqz2zlmk/v1

Protocol status: In development
We are still developing and optimizing this protocol

Created: May 12, 2025

Last Modified: May 12, 2025

Protocol Integer ID: 218082

Keywords: cnn classification workflow for visual impairment research, visual impairment research, impairment research, cnn classification workflow, rna, realistic synthetic rna, cnn classifier, performance via roc, dataset, reproducible pipeline, seq data generation

Funders Acknowledgements:
King Salman Center for Disability Research
Grant ID: ERC -2024-01

Abstract

A robust, reproducible pipeline to (i) generate realistic synthetic RNA-seq data for visual-impairment research using a Negative Binomial model, (ii) build and train a 1D-CNN classifier, (iii) evaluate performance via ROC-AUC and SHAP interpretability, and (iv) publish both code and datasets to public repositories (Protocols.io & NCBI GEO).

Materials

- Hardware: Any workstation with ≥16 GB RAM and a GPU (e.g., NVIDIA RTX series)
- OS: Linux (Ubuntu 20.04+) or Windows 10/11
- Programming: Python 3.8+
- Libraries: numpy, pandas, scikit-learn, tensorflow (or PyTorch), shap, biopython
- NCBI Accounts: GEO submission account (https://www.ncbi.nlm.nih.gov/account/)
- Protocols.io: Free account (email/ORCID)

Prepare Synthetic RNA-seq Data

1 Open a Python environment and install dependencies:

1.1 pip install numpy pandas scipy

2 Simulate counts for N samples × G genes using a Negative Binomial distribution:

2.1 import numpy as np

2.2 data = np.random.negative_binomial(n=10, p=0.3, size=(100,500))

3 Assign binary labels (0=healthy, 1=visually impaired) and save as CSV:

3.1 import pandas as pd

3.2 labels = np.random.choice([0,1], size=100)

3.3 df = pd.DataFrame(data, columns=[f"Gene_{i+1}" for i in range(500)])

3.4 df["Condition"] = labels

3.5 df.to_csv("synthetic_RNAseq.csv", index=False)

Data Preprocessing

4 Load your CSV and apply log2-CPM normalization:

4.1 import numpy as np, pandas as pd

4.2 df = pd.read_csv("synthetic_RNAseq.csv")

4.3 counts = df.iloc[:,:-1]

4.4 log_cpm = np.log2((counts.div(counts.sum(axis=1), axis=0) * 1e6) + 1)

5 Scale features (zero mean, unit variance):

5.1 from sklearn.preprocessing import StandardScaler

5.2 scaler = StandardScaler()

5.3 X = scaler.fit_transform(log_cpm)

5.4 y = df["Condition"].values

Train-Validation-Test Split

6 Split into 70/15/15 proportions:

6.1 from sklearn.model_selection import train_test_split

6.2 X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, stratify=y, random_state=42)

6.3 X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.50, stratify=y_temp, random_state=42)

Build the 1D-CNN Model

7 Define architecture (TensorFlow/Keras example):

7.1 import tensorflow as tf

7.2 model = tf.keras.Sequential([tf.keras.layers.Input(shape=(X_train.shape[1],1)), tf.keras.layers.Conv1D(64, 3, activation="relu"), tf.keras.layers.MaxPooling1D(2), tf.keras.layers.Conv1D(128, 3, activation="relu"), tf.keras.layers.MaxPooling1D(2), tf.keras.layers.Flatten(), tf.keras.layers.Dense(128, activation="relu"), tf.keras.layers.Dropout(0.5), tf.keras.layers.Dense(1, activation="sigmoid")])

7.3 model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

Model Training

8 Reshape data for Conv1D and train with early stopping:

8.1 X_train_c = X_train[..., np.newaxis]

8.2 X_val_c = X_val[..., np.newaxis]

8.3 es = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True)

8.4 history = model.fit(X_train_c, y_train, validation_data=(X_val_c, y_val), epochs=50, batch_size=32, callbacks=[es])

Evaluation & Interpretation

9 Evaluate on test set:

9.1 X_test_c = X_test[..., np.newaxis]

9.2 model.evaluate(X_test_c, y_test)

10 Compute ROC-AUC:

10.1 from sklearn.metrics import roc_auc_score

10.2 y_pred = model.predict(X_test_c).ravel()

10.3 roc_auc_score(y_test, y_pred)

11 SHAP analysis for feature importance:

11.1 import shap

11.2 explainer = shap.KernelExplainer(model.predict, X_train_c[:50])

11.3 shap_values = explainer.shap_values(X_test_c[:20])

11.4 shap.summary_plot(shap_values, X_test[:20])