DNA SEQUENCE MATCHING USING CNN

MINI PROJECT REPORT

Submitted to the Department of Computer Applications, Bharathiar University in partial fulfilment of the requirement for the award of the degree of

MASTER OF SCIENCE IN DATA ANALYTICS

Submitted by

JELAS LIBINA.W(22CSEG13)

Under the guidance of

Mr. P.K. PRAKAASH., MCA., M.Phil., ME(CSE)., NET., SET.,

Guest Faculty

DEPARTMENT OF COMPUTER APPLICATIONS

BHARATHIAR UNIVERSITY

COIMBATORE-641046

DECEMBER – 2023

DECLARATION

I hereby declare that this mini project report titled **"DNA SEQUENCE MATCHING USING CNN"** submitted to the Department of Computer Applications, Bharathiar University, Coimbatore is a record of original mini project work done by **JELAS LIBINA.W(22CSEG13)** under the supervision and guidance of **Mr. P.K. PRAKAASH., MCA., M.Phil., ME(CSE)., NET., SET.,** Department of Computer Applications, Bharathiar University, and that this project work has not previously formed the basis of the award of Degree / Diploma / Associate ship / Fellowship or similar titled to any candidate of any University.

Place: Coimbatore

Date:

Signature of Candidate

JELAS LIBINA.W

Countersigned by

Signature of the guide

CERTIFICATE

This is to certify that, this mini project work entitled "DNA SEQUENCE MATCHING USING CNN" submitted to the Department of Computer Applications, Bharathiar University in partial fulfilment of the requirements for the award of the degree of MASTER OF COMPUTER APPLICATONS, is a record of original work done by JELAS LIBINA.W(22CSEG13), during her period of study in the Department of Computer Applications, Bharathiar University, Coimbatore, under my supervision and guidance, and this project work has not previously formed the basis for the award of any Degree / Diploma / Associate ship / Fellowship or similar title to any candidate of any University.

Place: Coimbatore

Date:

Project Guide **Head of the Department**

Submitted for the University Viva-Voce Examination held on _____

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

I have taken efforts in this project; However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extent my sincere thanks to all of them.

I express my sincere gratitude to **Dr. M. PUNITHAVALLI, M.Sc., M.Phil., Ph.D.,** Professor & Head of the Department of Computer Applications, Bharathiar University, Coimbatore, thanks to department faculty and guest faculties supported to acquire knowledge from various inputs.

I am forever indebted to this mini project guides **Mr. P.K. PRAKAASH., MCA., M.Phil., ME(CSE)., NET., SET.,** Department of Computer Application, Bharathiar University for providing valuable suggestions and always been enthusiastically guiding us throughout the project.

Finally, I also extent my special thanks to my family, friends, who have kindly provided the necessary support for the successful completion of the project and their moral support.

JELAS LIBINA.W(22CSEG13)

ABSTRACT

Progress in genetic research has led to the exploration of new computational methods for understanding genetic connections. This study explores the similarity between the two individuals based on their DNA sequences using Convolutional Neural Network (CNNs). The methodology involves the transformation of DNA sequence into numerical representations, enabling the utilization of CNNs for pattern recognition. This process includes the one-hot-encoding of DNA sequences, the training of a CNN to identify intricate patterns indicative of genetic matches, and a comprehensive evaluation of its performance across diverse datasets. To validate the efficacy of the CNN based approach, extensive experimentation is conducted using diverse datasets containing DNA sequences from individuals with known genetic relationships. The results show the CNNs effectiveness in identifying genetic matches, highlighting its potential applications in forensic analysis, paternity testing, and population genetics. Challenges such as dataset size and interpretability are acknowledged and suggest the avenues for further research.

TABLE OF CONTENTS

1. INTRODUCTION

The project is entitled as “DNA Sequence Matching Using Convolutional Neural Network”. The field of bioinformatics has seen remarkable advancements with the integration of machine learning techniques, particularly convolutional neural network (CNNs), for analysing DNA sequences. DNA sequence matching is a critical task in genomics and molecular biology, aiming to identify similarities between individual DNA sequences. Convolutional Neural Networks, originally developed for image recognition, have proven to be effective in capturing hierarchical features within sequential data, making them-suited for tasks like DNA sequence analysis.

DNA is the fundamental genetic material in living organisms, is composed of sequences of nucleotides, represented by the letters A(adenine), T(thymine), C(cytosine), and G(guanine). The arrangement of these nucleotides forms the unique genetic code for each individual. DNA sequence matching involves comparing these sequences to determine the degree of similarity or dissimilarity between individuals, which is crucial for various applications such as disease diagnosis, ancestry analysis, and forensic investigations.

Convolutional Neural Networks offer a powerful approach for DNA sequence matching due to their ability to automatically learn hierarchical patterns and features from input data. In the context of DNA sequences, these networks can recognize motifs, patterns, and dependencies within the sequences at different levels of abstraction. The convolutional layers in a CNN, originally designed for image processing, can be adapted to capture local patterns in DNA sequences, while the pooling layers help in reducing the dimensionality and preserving important features.

1.1 OBJECTIVE

Develop and implement a Convolutional Neural Network (CNN) for DNA sequence matching, with the primary goal of accurately determining genetic similarity between individuals and classifying whether given DNA sequences match or not. This study aims to address the limitations of manual DNA analysis, leveraging the power of CNNs to enhance efficiency and accuracy.

2. RESEARCH METHODOLOGY

2.1 PROPOSED SYSTEM

In addressing the challenges of DNA sequence matching for individual identification which was proposed by Convolutional Neural Network (CNN)-based model. This model, designed for accurate and efficient DNA sequence analysis, aims to identify individuals based on the genetic materials. The CNN extracts intricate patterns from DNA sequences, and our methodology includes unique features like Genetic sequence(genotype), Chromosomal information (chromosomes, positions, and genotypes for child and father genomes), Data preprocessing like Handling missing values, One-hot-encoding, Sequence padding. By implementing this proposed system, we anticipate improved accuracy in identifying genetic profiles and enhanced robustness against variations. This proposed system contributes to advancing DNA sequence matching capabilities, particularly in the domain of [specify the application, e.g., forensic analysis or personalized medicine].

2.2 CNN MODEL

Convolutional Neural Network (CNN) is a Deep Learning algorithm which is the specialized neural networks designed for processing grid-like data, such as images or, in this case, sequential data like genetic sequences. They are particularly effective in capturing hierarchical patterns and local dependencies in input data.

- **Embedding Layer:**
  The first layer is an Embedding layer, converting integer-encoded DNA sequences into dense vectors. It helps the network learn meaningful representations of sequences.
- **Convolutional Layers:**
  Convolutional layers use filters to detect local patterns in the input sequences. The number of filters and kernel size are adjustable parameters. Activation function ReLU (Rectified Linear Unit) introduces non-linearity.
- **MaxPooling Layers:**
  MaxPooling layers down-sample the spatial dimensions of the input, retaining the most important information.

- **Flatten Layer:**

  The Flatten layer transforms the 2D feature maps from convolutional layers into a 1D vector, suitable for input to dense layers.

- **Dense Layers:**

  Dense layers are fully connected layers responsible for classification. The final dense layer uses SoftMax activation for multi-class classification, producing probability distributions over classes.

- **Model Compilation:**

  Categorical cross entropy is used as the loss function, suitable for multi-class classification. The Adam optimizer is chosen for efficient weight updates during training. Accuracy is chosen as the evaluation metric.

- **Training:**

  The model is trained using the fit method on the training data, specifying the number of epochs and batch size. Validation data are used to monitor model performance during training.

- **Model Evaluation:**

  The trained model can be evaluated on test data to assess its generalization performance.

- **Potential Improvements:**

  Experimenting with different architectures, hyperparameters, and regularization techniques could enhance model performance. Consideration of attention mechanisms or deeper architectures might be beneficial for capturing long-range dependencies in genetic sequences.

2.3 ARCHITECTURE

- The CNN model is defined using the Keres Sequential API.
- It starts with an Embedding layer, mapping integer-encoded DNA sequence input to dense vectors.
- Convolutional layers follow the embedding layer. These layers apply filters to capture local patterns in the sequences.
- MaxPooling layers are used to down-sample the spatial dimensions, reducing computational complexity.
- Flatten layer transforms the 2D feature maps into a 1D vector, preparing the data for dense layers.
- Dense layers provide the final classification, with activation functions like ReLU and SoftMax.

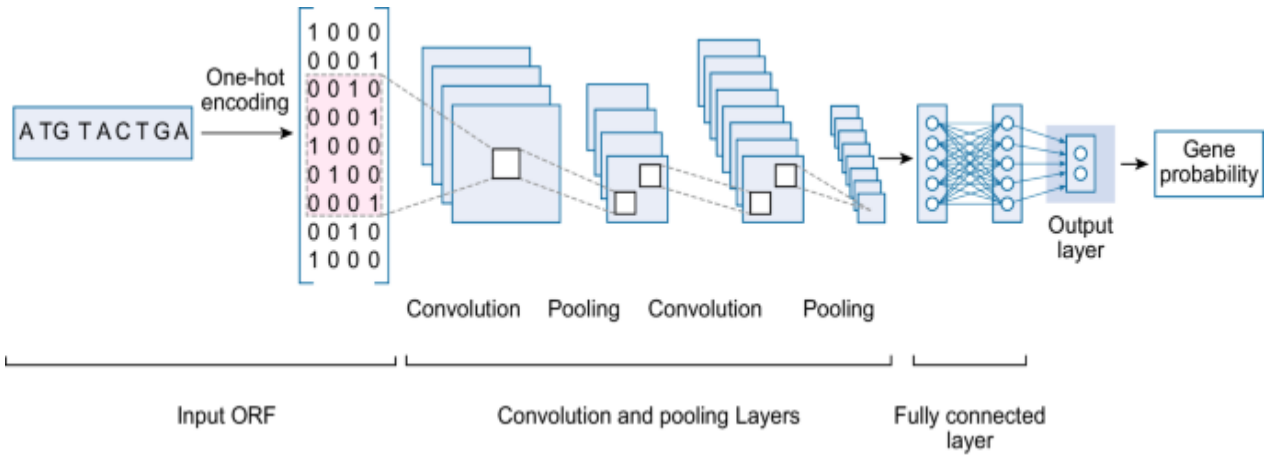

Fig:2.1-CNN Architecture

3. IMPLEMENTAION

3.1 DATASET DESCRIPTION

For this mini-project, I utilized a secondary dataset available on the Kaggle platform—the Family of Five – Genome Dataset. The dataset comprises two parent and three sibling CSV files containing information on chromosomes, genotypes, and positions. I specifically used the father's genome and one of the child's genomes for my analysis and for training and predicting the model. These files are released as a training set for academic machine learning purposes.

The dataset contains attributes like

- Rsid (Reference SNP cluster ID)
- Chromosome
- Position
- Genotype

Rsid:

It is a reference SNP cluster ID, a unique identifier for a specific variation in the genome.

Chromosome:

It indicates the chromosome number where the variation is located.

Position:

It denotes the specific position on the chromosome where the variation occurs.

Genotype:

It represents the genetic code at that particular position. For example, "AA," "AG," "AC," or "GG" are possible genotypes, indicating the combination of alleles at the specified position.

3.2 LIBRARIES USED

- **pandas (pd):** Used for data manipulation and analysis.
- **NumPy (np):** Provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these.
- **TensorFlow (tf):** An open-source machine learning library developed by the Google Brain team. It is used for building and training neural network models.
- **keras:** A high-level neural networks API that runs on top of TensorFlow, simplifying the process of building and training deep learning models.
- **matplotlib. pyplot (plt):** A plotting library for creating visualizations in Python.
- **seaborn (sns):** A statistical data visualization library based on Matplotlib, providing an interface for drawing attractive and informative statistical graphics.
- **Bio.motifs:** From the Bio python library, used for working with biological sequence motifs.
- **sklearn. model selection. train_test_split:** Used for splitting datasets into training and testing sets.
- **sklearn. preprocessing. Label Encoder:** Used for converting categorical class labels into numerical labels.
- **sklearn. model selection. GridSearchCV:** A scikit-learn module for performing hyperparameter tuning using grid search.
- **Sequential, Conv1D, MaxPooling1D, Flatten, Dense, Masking, GlobalAveragePooling1D, GlobalMaxPooling1D:** Classes from Keras for building and configuring the Convolutional Neural Network (CNN) model.

3.3 PROCESSING OF DATA

Data Transformation:

Pre-processing refers to the transformation applied to our data before providing the dataset to the algorithm. Data pre-processing technique is used to convert the raw data into an understandable data set. In other words, whenever the information is gathered from various sources is collected in raw format that is not possible for the analysis.

```
[ ]  def one_hot_encode(sequence):
         mapping = {'A': [1, 0, 0, 0], 'C': [0, 1, 0, 0], 'G': [0, 0, 1, 0], 'T': [0, 0, 0, 1]}
         encoded_sequence = []

         for base in sequence:
             if base in mapping:
                 encoded_sequence.append(mapping[base])
             else:
                 # Handle unknown characters (e.g., '-')
                 encoded_sequence.append([0, 0, 0, 0])  # You can modify this line based on your requirements

         return np.array(encoded_sequence)

     # Ensure each sequence has the same length
     max_sequence_length = max(len(seq) for seq in child_sequences)
     child_sequences_padded = [seq.ljust(max_sequence_length, 'N') for seq in child_sequences]
     father_sequences_padded = [seq.ljust(max_sequence_length, 'N') for seq in father_sequences]
```

```
[ ]  max_sequence_length

     2
```

```
[ ]  child_sequences_padded

     ['AA',
      'AG',
      'AC',
      'AG',
      'GG',
      'GG',
      'AA',
      'AC',
      'CT',
      'AG'
```

```
[ ]  child_sequences_encoded = np.array([one_hot_encode(seq) for seq in child_sequences_padded])
     father_sequences_encoded = np.array([one_hot_encode(seq) for seq in father_sequences_padded])
```

```
[ ]  father_sequences_encoded
```

```
array([[[1, 0, 0, 0],
        [1, 0, 0, 0]],

       [[1, 0, 0, 0],
        [0, 0, 1, 0]],

       [[1, 0, 0, 0],
        [0, 1, 0, 0]],

       ...,

       [[0, 1, 0, 0],
        [0, 0, 0, 0]],

       [[0, 1, 0, 0],
        [0, 0, 0, 0]],

       [[0, 1, 0, 0],
        [0, 0, 0, 0]]])
```

```
[ ]  child_sequences_encoded
```

```
array([[[1, 0, 0, 0],
        [1, 0, 0, 0]],

       [[1, 0, 0, 0],
        [0, 0, 1, 0]],

       [[1, 0, 0, 0],
        [0, 1, 0, 0]],

       ...,

       [[0, 1, 0, 0],
        [0, 0, 0, 0]],

       [[0, 1, 0, 0],
        [0, 0, 0, 0]],

       [[0, 1, 0, 0],
        [0, 0, 0, 0]]])
```

The provided code transforms DNA sequences for machine learning. It employs one-hot encoding to convert nucleotides into binary vectors and pads sequences for consistent length. This standardized representation (child_sequences_padded and father_sequences_padded) is vital for training models, particularly in genetic trait prediction. The process ensures a uniform format, crucial for effective machine learning model training and prediction accuracy.

Data cleaning:

Improper data can cause confusion and results in unreliable and poor output. Hence first step in Data Pre-processing is Data Cleaning. Cleaning of data is done by filling in missing values, smoothing noisy data by identifying and/or removing inconsistencies.

```
[ ]  child_data.isnull().sum()
```

```
# rsid        0
chromosome    0
position      0
genotype      0
dtype: int64
```

```
[ ]  father_data.isnull().sum()
```

```
# rsid        0
chromosome    0
position      0
genotype      0
dtype: int64
```

In this dataset there is no missing values.

Checking and resizing of DNA sequence:

```
[ ]  # Assuming sequence_length is known, replace it with the actual length
     sequence_length = 2  # Replace with the actual length of your sequences

     num_samples = 1000
     sequence_length = 2
     num_classes = 2

     X_train = np.random.randint(0, 4, size=(num_samples, sequence_length, 4))
     y_train = np.random.randint(0, 2, size=(num_samples, num_classes))
     # Reshape X_train and X_test to ensure each element is a 2D array
     X_train = np.array([np.array(seq).flatten() for seq in X_train])
     X_test = np.array([np.array(seq).flatten() for seq in X_test])

     # Reshape each element to have the correct shape
     X_train = X_train.reshape(-1, sequence_length, 4)
     X_test = X_test.reshape(-1, sequence_length, 4)

     # Print types and shapes after reshaping
     print("X_train type:", type(X_train), "shape:", X_train.shape)
     print("X_test type:", type(X_test), "shape:", X_test.shape)

     # Convert NumPy arrays to TensorFlow tensors
     X_train_tf = tf.convert_to_tensor(X_train, dtype=tf.float32)
     y_train_tf = tf.convert_to_tensor(y_train, dtype=tf.float32)
     X_test_tf = tf.convert_to_tensor(X_test, dtype=tf.float32)
     y_test_tf = tf.convert_to_tensor(y_test, dtype=tf.float32)
```

```
X_train type: <class 'numpy.ndarray'> shape: (1000, 2, 4)
X_test type: <class 'numpy.ndarray'> shape: (120361, 2, 4)
```

3.4 EXPLORATORY DATA ANALYSIS

Exploring the different features of the dataset and their distribution.

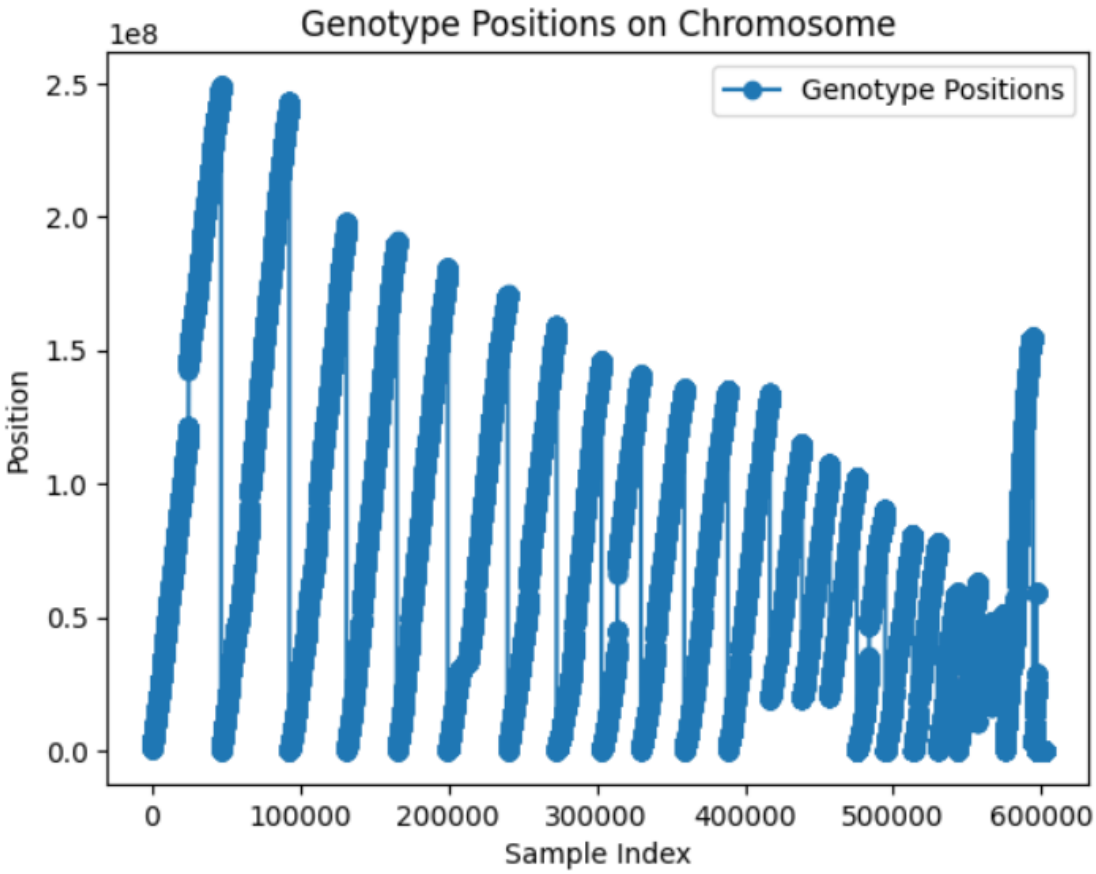

Fig:3.1-Genotype position on chromosome

The above figure of visualization helps in understanding the distribution of genotype positions on the chromosome across the samples in the dataset.

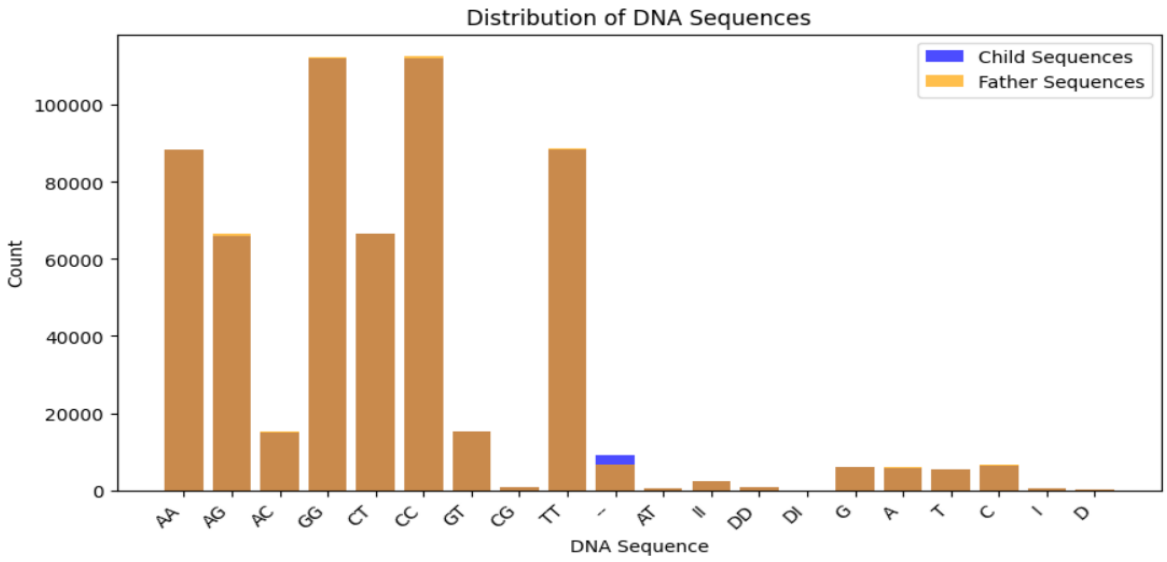

Fig:3.2-Distribution of DNA sequence

The resulting bar chart shows the distribution of DNA sequences for child and father samples, providing a visual comparison of the frequencies of different DNA sequences in the two datasets.

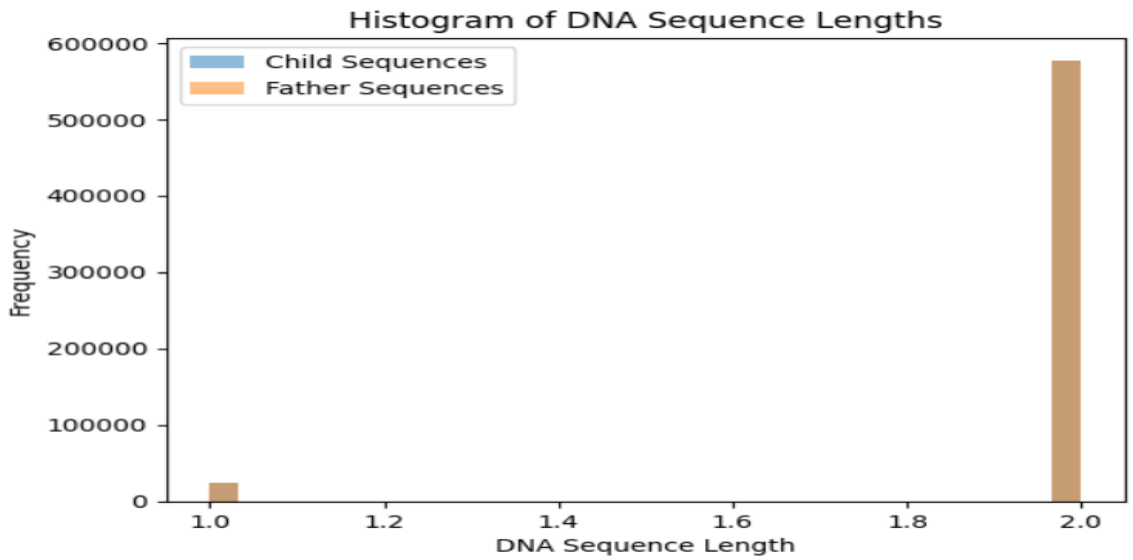

Fig:3.3-DNA sequence length

The above figure shows the distribution of DNA sequences for child and father samples, providing a visual comparison of the frequencies of different DNA sequences in the two datasets.

4. MODEL BUILDING

4.1 SPLITTING OF TRAINING, VALIDATION, AND TEST

For the training and testing purpose of our model, we should split the data

Training Set

It is the set of data that is used to train and make the model learn the hidden features/patterns in the data.

In each epoch, the same training data is fed to the neural network architecture repeatedly, and the model continues to learn the features of the data. The training set should have a diversified set of inputs so that the model is trained in all scenarios and can predict any unseen data sample that may appear in the future.

The Test Set

The test set is a separate set of data used to test the model after completing the training. It provides an unbiased final model performance metric in terms of accuracy, precision, etc.

The Validation Set

The validation set is a set of data, separate from the training set, that is used to validate our model performance during training. This validation process gives information that helps us tune the model's hyperparameters and configurations accordingly. The model is trained on the training set, and simultaneously, the model evaluation is performed on the validation set after every epoch.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    father_sequences_encoded,child_sequences_encoded, test_size=0.2, random_state=42)
```

4.2 BUILDING MODEL

A Convolution Neural Network (CNN) is a class of deep learning neural networks models that is most commonly applied to analysing visual imagery. Its effectiveness comes from the fact that CNNs are fully connected feed forward neural networks that reduce the number of parameters very efficiently without losing out on the quality of models. Image has high dimensionality features with every layer.

```
[ ] from keras.utils import to_categorical
    # Build a simple CNN model
    model = Sequential()
    model.add(Conv1D(32, kernel_size=2, activation='relu', input_shape=(sequence_length, 4)))
    model.add(MaxPooling1D(pool_size=1))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.add(Dense(1, activation='sigmoid'))
```

```
[ ] # Build the model
    model = Sequential()
    model.add(Conv1D(32, kernel_size=2, activation='relu', input_shape=(sequence_length, 4)))
    model.add(MaxPooling1D(pool_size=1))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))  # Use 'sigmoid' for binary classification

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

5. MODEL EVALUATION

5.1 FITTING THE MODEL

```
# Train the model
model.fit(X_train, y_train_binary, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
25/25 [==============================] - 10s 196ms/step - loss: 0.5994 - accuracy: 0.7375 - val_loss: 0.5411 - val_accuracy: 0.7750
Epoch 2/10
25/25 [==============================] - 4s 168ms/step - loss: 0.5769 - accuracy: 0.7425 - val_loss: 0.5366 - val_accuracy: 0.7750
Epoch 3/10
25/25 [==============================] - 0s 11ms/step - loss: 0.5701 - accuracy: 0.7425 - val_loss: 0.5355 - val_accuracy: 0.7750
Epoch 4/10
25/25 [==============================] - 0s 13ms/step - loss: 0.5661 - accuracy: 0.7425 - val_loss: 0.5346 - val_accuracy: 0.7750
Epoch 5/10
25/25 [==============================] - 0s 17ms/step - loss: 0.5621 - accuracy: 0.7425 - val_loss: 0.5367 - val_accuracy: 0.7750
Epoch 6/10
25/25 [==============================] - 0s 17ms/step - loss: 0.5628 - accuracy: 0.7425 - val_loss: 0.5405 - val_accuracy: 0.7750
Epoch 7/10
25/25 [==============================] - 0s 15ms/step - loss: 0.5597 - accuracy: 0.7425 - val_loss: 0.5369 - val_accuracy: 0.7750
Epoch 8/10
25/25 [==============================] - 0s 11ms/step - loss: 0.5567 - accuracy: 0.7425 - val_loss: 0.5343 - val_accuracy: 0.7750
Epoch 9/10
25/25 [==============================] - 0s 12ms/step - loss: 0.5547 - accuracy: 0.7425 - val_loss: 0.5371 - val_accuracy: 0.7750
Epoch 10/10
25/25 [==============================] - 0s 14ms/step - loss: 0.5514 - accuracy: 0.7425 - val_loss: 0.5356 - val_accuracy: 0.7750
<keras.src.callbacks.History at 0x7bd0218c3730>
```

5.2 EVALUATING THE MODEL

More the accuracy, better the model. Every model is evaluated on the accuracy achieved and the loss obtained. There are two accuracy involved validation accuracy involved validation accuracy and test accuracy.

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test_reshaped, y_test_binary)

print(f"Test Accuracy: {accuracy}")
```

```
3762/3762 [==============================] - 56s 14ms/step - loss: 0.4875 - accuracy: 0.9318
Test Accuracy: 0.9317947626113892
```

5.3 PREDICTION

```
[ ]  father_sequences_encoded_reshaped = father_sequences_encoded.reshape(-1, sequence_length, 4)

     # Make predictions
     father_predictions = model.predict(father_sequences_encoded_reshaped)

     # Print or use the predictions as needed
     print(father_predictions)
```

```
18807/18807 [==============================] - 227s 12ms/step
[[0.3956111 ]
 [0.37294647]
 [0.3246621 ]
 ...
 [0.43150097]
 [0.43150094]
 [0.43150094]]
```

Fixing the threshold value to find the matching results

```
[ ]  # Assuming a threshold of 0.5 for binary classification
     matching_threshold = 0.2
     matching_results = father_predictions > matching_threshold
     matching_results
```

```
array([[ True],
       [ True],
       [ True],
       ...,
       [ True],
       [ True],
       [ True]])
```

Thus, the model predict the DNA child matches to father

```
[ ]  # Output results
     for i, prediction in enumerate(father_predictions):
         print(f"Father-{i+1} DNA Prediction: {prediction[0]}")
         # You can also use a threshold to determine if it's a match or not
         match = prediction[0] > matching_threshold
         print(f"Father-{i+1} DNA matches Child DNA: {match}")
         not_match =prediction
```

```
Streaming output truncated to the last 5000 lines.
Father-51490 DNA matches Child DNA: True
Father-51491 DNA Prediction: 0.2983151972293854
Father-51491 DNA matches Child DNA: True
Father-51492 DNA Prediction: 0.4234350025653839
Father-51492 DNA matches Child DNA: True
Father-51493 DNA Prediction: 0.4234350025653839
Father-51493 DNA matches Child DNA: True
Father-51494 DNA Prediction: 0.38782721757888794
Father-51494 DNA matches Child DNA: True
```

6. CONCLUSION

Our study explores using Convolutional Neural Networks (CNNs) to identify genetic similarities based on DNA sequences. We convert DNA sequences into numerical forms using one-hot-encoding, allowing CNNs to recognize patterns that indicate genetic matches. The study trains and evaluates the CNN on diverse datasets with known genetic relationships, showing its effectiveness in identifying genetic matches. This approach has promising applications in forensic analysis, paternity testing, and population genetics.

However, we acknowledge challenges like dataset size and interpretability. These challenges suggest opportunities for future research, highlighting the need to improve and explore CNN-based methods in genetic analysis. Despite these obstacles, our study provides valuable insights into applying computational methods to understand genetic connections, contributing to advancements in genetic research with real-world implications.

BIBILOGRAPHY

- https://www.kaggle.com/code/singhakash/dna-sequencing-with-machine-learning/notebook
- https://www.kaggle.com/datasets/zusmani/family-genome-dataset/code
- https://www.healthline.com/health/what-is-dna#what-is-dna
- https://www.genome.gov/genetics-glossary/Nucleotide#:~:text=A%20nucleotide%20is%20the%20basic,)%20and%20thymine%20(T)
- https://www.youtube.com/watch?v=gG7uCskUOrA
- https://www.geeksforgeeks.org/introduction-deep-learning/
- https://www.kaggle.com/code/shtrausslearning/biological-sequence-alignment
- International Journal of Scientific Engineering and Technology Research Volume.03, IssueNo.35, November-2014