Pre-processing and analysis of genomic data using data mining tools

Théo FIGINI
M2 Computer Science
Academic year 2024-2025

Hosting organization : *Université des Antilles*

January, 13th, 2025–June, 16th, 2025

Referring teacher:
Emmanuel BIABIANY

Tutor :
Wilfried SEGRETIER

June, 16th 2025

Contents

Introduction

1 Related Work

Genomics is a field that has been growing rapidly in the past few years. The advent of high-throughput sequencing technologies has made it possible to sequence the entire genome of an organism in a matter of days. This has led to an explosion of data, with the number of sequenced genomes increasing exponentially. This has created a need for new tools and algorithms to analyze this data. In this chapter, we review some of the existing tools and algorithms for analyzing genomic data.

1.1 Genomic data analysis

Random forests

Another method that has been used for genomic data analysis is random forests (RF) [2]. This method is based on the idea of ensemble learning, where multiple decision trees are trained on different subsets of the data and then combined to make a final prediction.

AI applications in genomic analysis

Many researchers have used AI techniques to analyze genomic data. For example, [1] reviews different AI techniques that have been used for genomic data analysis, including CNNs, autoencoders, etc.

More recently, the authors of [7] designed a tool based on multiple LLM backends for multi-omics analysis with minimal human intervention.

1.2 Introduction to CNNs

The fundamental idea behind **Convolutional Neural Networks (CNNs)** was introduced by Kunihiko Fukushima[3] in 1980 and later popularized by Yann LeCun in the 1990s with the LeNet architecture[4]. CNNs are a class of deep learning models specifically designed for processing structured grid data, such as images. They are particularly effective for tasks like image classification, object detection, and segmentation.

CNNs work by applying convolutional filters to the input data, which allows them to learn spatial hierarchies of features. It shares some similarities with the more conventional neural networks, but there is at least two different types of layers that are specific to CNNs:

- **Convolutional layers**: These layers apply convolutional filters to the input data, extracting local features that are invariant to translation. The filters slide over the input data, computing dot products and producing feature maps.

– **Pooling layers**: These layers downsample the feature maps, reducing their spatial dimensions while retaining important information. Common pooling operations include max pooling and average pooling.

The architecture of a CNN typically consists of multiple convolutional and pooling layers followed by fully connected layers. The convolutional layers learn to extract features from the input data, while the fully connected layers perform the final classification or regression task. The training process involves optimizing the weights of the filters and fully connected layers using backpropagation and a loss function, such as cross-entropy for classification tasks.

K-fold cross validation

K-fold cross validation is a specific type of cross validation where the dataset is divided into k equally sized folds. The model is trained on $k-1$ folds and tested on the remaining fold, and this process is repeated for each fold. The final performance is averaged over all folds to obtain a more reliable estimate of the model's performance. The graphical representation of K-fold cross validation is shown in Figure 1.

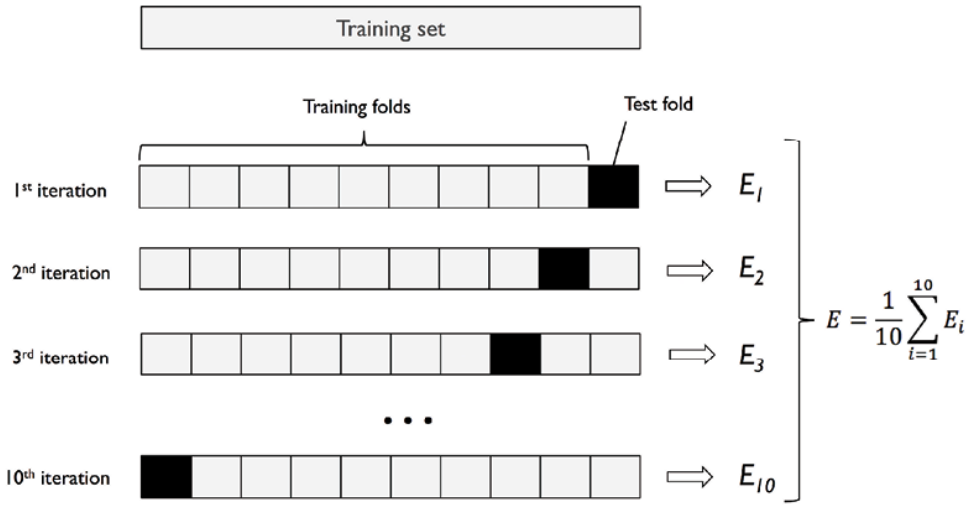

Figure 1: How K-fold cross validation works.[5]

Stratified K-fold cross validation

Stratified K-fold cross validation is a variation of K-fold cross validation that ensures that each fold has the same proportion of samples from each class. This is particularly useful for imbalanced datasets, as it ensures that each fold has a representative sample of each class. Even if this method is recommended for imbalanced datasets, we did not have enough time to implement it, so we used the standard K-fold cross validation.

SMOTE

Data augmentation is a technique used to artificially increase the size of a dataset by applying various transformations to the existing data. This can help improve the model's performance by providing more diverse training samples and reducing overfitting. In our project, we used SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples for the minority classes in our dataset.

SMOTE works by creating new samples by interpolating between existing samples in the feature space. It generates synthetic samples by selecting a minority class sample, finding its nearest neighbors, and creating new samples by interpolating between the selected sample and its neighbors. This helps to balance the dataset by increasing the number of samples in the minority classes, which can improve the model's performance on those classes.

New samples are generated by taking a random sample from the minority class and finding its nearest neighbors in the feature space. The new sample is then created by interpolating between the selected sample and its neighbors, as shown in Figure 2. It uses the following formula:

$$x_{new} = x_i + \lambda(x_j - x_i) \tag{1}$$

Where x_{new} is the new sample, x_i is the selected sample, x_j is one of its nearest neighbors, and λ is a random number between 0 and 1. This process is repeated until the desired number of samples is generated for the minority class.

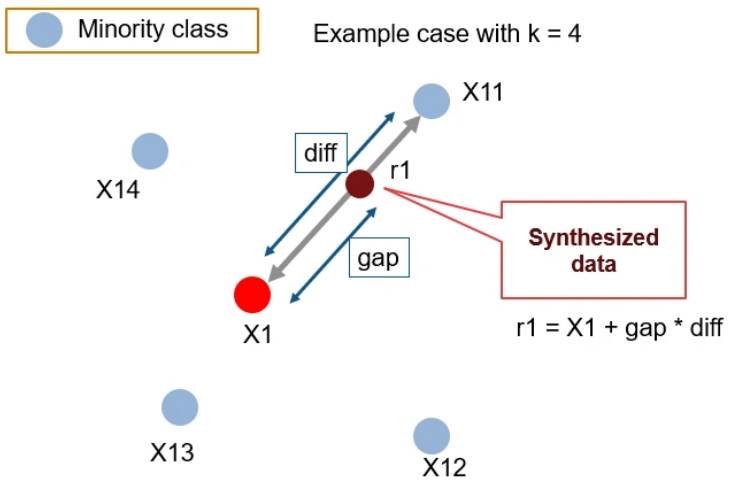

Figure 2: Illustration of the SMOTE algorithm.[6]

2 Problem Statement

2.1 Context

2.2 Work Environment

This project was done within the framework of the Master 2 Computer Science program at the **Laboratoire de Mathématiques Informatique et Applications (LAMIA)** of the **Université des Antilles**. The following tools and technologies were used to develop the algorithms and document the project:

- **Python**: a programming language widely used in scientific computing, data analysis, and machine learning.
- **GitHub**: a version control platform for managing the source code.
- **GitHub Copilot**: for code completion.
- **Jupyter Notebook**: an interactive environment for writing and executing Python code, ideal for data analysis and visualization.
- **Sklearn**: a Python library for machine learning that provides various algorithms and tools for data preprocessing, model selection, and evaluation.
- LaTeX: a typesetting system used for writing technical and scientific documents.
- A personal computer running Windows 11 with a **Ryzen 7 6800H** processor and **32 GB** of RAM.

2.3 Objectives

3 Genomic data

Genomic data refers to the information contained in the DNA of an organism, which includes the sequences of nucleotides (A, T, C, G). These are gathered through various methods, such as DNA sequencing, and can be used to study the genetic makeup of organisms, identify genetic variations, and understand the relationships between different species. The Institute Pasteur has a large collection of genomic data, including the genomes of various strains of Tuberculosis (TB), which is a major public health concern worldwide.

3.1 The FASTA format

The FASTA format is a text-based format for representing nucleotide or protein sequences. It is widely used in bioinformatics to store and exchange sequence data. A FASTA file consists of one or more sequences, each represented by a header line starting with a greater-than symbol (`>`), followed by the sequence itself on the next line. The header line can contain additional information about the sequence, such as its identifier, description, or source. An example of a FASTA file is shown in Figure 3.

```
>sequence_id
ATCGATCGATCGATCGATCGATCG
>sequence_id_2
GCTAGCTAGCTAGCTAGCTAGCTAGCTA
>sequence_id_3
TAGCTAGCTAGCTAGCTAGCTAGCTAGC
>sequence_id_4
AGCTAGCTAGCTAGCTAGCTAGCTAGCT
>sequence_id_5
CATCGATCGATCGATCGATCGATCGATC
```

Figure 3: Example of a FASTA file containing two sequences.

The FASTA format is simple and easy to read, making it suitable for storing large amounts of sequence data. It is also compatible with many bioinformatics tools and software, allowing for easy analysis and manipulation of sequence data.

4 Clustering genomic data

4.1 Preprocessing Genomic Data

Our data is a collection of Tuberculosis (TB) genomes, where each file contains the proteins and their respective nucleotide sequences in FASTA format, for a total of 7057 files. TODO: parler de l'institut Pasteur? The goal is to cluster these genomes based on their protein sequences and find ways to distinguish between different TB strains.

We started by creating a dataset containing the number of times each protein appears in each genome. This was done by parsing the FASTA files and counting the occurrences of each protein sequence, we then removed the outliers, namely "hypothetical proteins" and "putative proteins", which are not well characterized and do not provide useful information for clustering. The resulting dataset is a CSV file of 7057 columns plus 1 column for the protein names and 2732 rows plus 1 row for the header.

Considering the size of the dataset, we used Principal Component Analysis (PCA) set to 95% variance to reduce the dimensionality of the data while preserving as much variance as possible. PCA is a technique that transforms the data into a new coordinate system, where the first principal component captures the maximum variance, the second captures the second maximum variance, and so on.

We then calculated the silhouette score for different values of k, ranging from 2 to 10, to determine the optimal number of clusters. The silhouette score was calculated using the 'silhouette_score' function from the 'sklearn.metrics' module, which takes the data and the cluster labels as input. The optimal number of clusters is the one that maximizes the silhouette score.

4.1.1 First Results

The initial results showed that the silhouette score was highest for $k = 9$, indicating that this was the optimal number of clusters. However, the clusters were not well separated, or some points are alone in their cluster, as seen in Figure TODO:figure:clusters_k9. Which suggests that the data may not be well suited for clustering with K-means, or that the features used for clustering are not informative enough.

TODO: figure clusters_k9

We also tried without applying PCA but achieved similar results, we also observed fewer points than expected as seen in Figure TODO:figure:clusters_k9_no_PCA, after reviewing the data, we found that there is significant overlap between the points.

TODO: figure clusters_k9_no_PCA

So we switched to 3D visualization to better understand the clusters,

4.2 New Approach using DBSCAN and HDBSCAN

To address the limitations of K-means clustering, we explored alternative clustering algorithms, specifically **DBSCAN** and **HDBSCAN**.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups points that are close together based on a distance measurement and a minimum number of points. It is particularly effective for datasets with varying densities and can identify noise points that do not belong to any cluster.

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is an extension of DBSCAN that builds a hierarchy of clusters and allows for more flexible clustering by varying the density threshold. It can also handle varying cluster shapes and sizes.

4.2.1 Principle

DBSCAN

DBSCAN works by defining a neighborhood around each point based on a distance metric (usually Euclidean distance) and a minimum number of points required to form a dense region. The algorithm proceeds as follows:

1. For each point in the dataset, find its neighbors within a specified radius ε.
2. If the number of neighbors is greater than or equal to the minimum number of points, a new cluster is formed.
3. The algorithm recursively expands the cluster by including all points that are reachable from the initial point.
4. Points that do not belong to any cluster are classified as noise.

HDBSCAN

HDBSCAN builds on the principles of DBSCAN by introducing a hierarchical clustering approach. It is more robust to varying densities and can identify clusters of different shapes and sizes. The algorithm works as follows:

1. Compute the mutual reachability distance between points, which takes into account the distance to the nearest neighbor.
2. Build a minimum spanning tree (MST) based on the mutual reachability distances.
3. Extract clusters from the MST by varying the density threshold, resulting in a hierarchy of clusters.
4. Assign points to clusters based on their membership in the hierarchy, allowing for flexible clustering.

4.2.2 Results

4.3 Trying an autoencoder

5 Using CNNs to identify TB Strains

5.1 Building the dataset

For this experiment, each genome was converted into an image where each pixel is determined by a combination of three metrics of following metrics:

- **Chargaff** index: provides the fraction of nucleic bases G and C of the sequence. The formula is:

$$100 \times \frac{Gcount + Ccount}{Sequence\_size} \tag{2}$$

- **Component** index: evaluates the distribution of the bases in a sequence by comparing the number of each base A and T to the expected frequency of those bases in a balanced distribution. The formula is:

$$\frac{(Acount + Tcount) + \frac{Sequence\_size}{2}}{\frac{Sequence\_size}{2}} \tag{3}$$

- **Diversity** index: measures the diversity of bases in a sequence by comparing the number of different subsequences of a given size to the expected frequency of those subsequences in a balanced distribution. The formula is:

$$\frac{Number_{\text{unique subsequences}}}{Expected_{\text{subsequences frequency}}} \tag{4}$$

- **Skew** index: measures the asymmetry of the distribution of nucleobases in a sequence. The formula is:

$$\frac{(Gcount - Ccount)}{Sequence\_size} \tag{5}$$

- **Nuclescore** index: empirical measure that combines multiple metrics from a genome sequence to synthesize the nucleotide information of a genome in order to differentiate species based only on their sequences. The formula is:

$$\log_2\left(\frac{Variance \times GC_{\text{ratio}} \times AT_{\text{count}}/GC_{\text{ratio}}}{\sqrt{Sequence\_size}}\right) \tag{6}$$

We get ten combinations of these metrics, each represented by a different color channel in the image after being normalized to the range [0, 255]. This conversion is applied at different resolutions, from 4px (2x2) to 100px (10x10), but for the training, all images were resized to a 32x32px resolution. The resulting images put into their corresponding class based on the strain they belong to, as shown in Figure 4. We have the following classes:

- **M** with 185 samples.

- **East Asian** with 1651 samples.
- **East-African Indian** with 267 samples.
- **Euro-American** with 4409 samples.
- **Indo-Oceanic** with 259 samples.

We also made mosaic images by combining the corresponding images from each combination of metrics resulting in a 5x2 grid resulting in a rectangular image as shown in Figure 5.

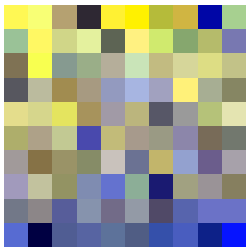

Figure 4: Example of an image generated from the genome of a TB strain using the Chargaff, Component and Diversity metrics at 100px resolution.

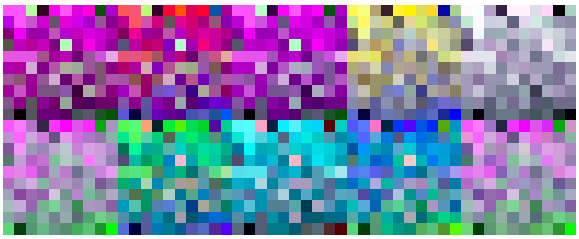

Figure 5: Example of a mosaic image obtained from the individual images at 100px resolution.

5.1.1 Initial testing

The resulting dataset is highly imbalanced, with some classes having significantly more samples than others. This can lead to biased models that perform poorly on minority classes. We decided to train our CNN on the dataset as-is, without any preprocessing or balancing techniques, to see how it performs on the imbalanced dataset. For the sake of brevity, we will only present the results for the combination of Chargaff, Composition and Diversity metrics at 100px resolution, for the mosaic images we will present the results for the 100px resolution as well.

5.1.1.1 Results

As shown in Figure 6, the 4px images yielded the worst performance, with a validation accuracy between 67% and 76%. We then observe a significant improvement in performance as the resolution increases with the accuracy starting to stabilize around 92% and 94% starting from the 36px resolution. Figure 7 and Table 1 show that the overall performances of the model are good, except for the East-African Indian and Indo-Oceanic classes with more moderate performances.

East-African Indian shows a moderate recall of 0.779 and a precision of 0.904, indicating that the model is able to correctly classify most of the samples in this class, but it also misclassifies some samples as other classes. The Indo-Oceanic class has a recall of 0.988, indicating that the model is able to correctly classify most of the samples in this class, but it has a lower precision of 0.718, indicating that the model misclassifies some samples as this class.

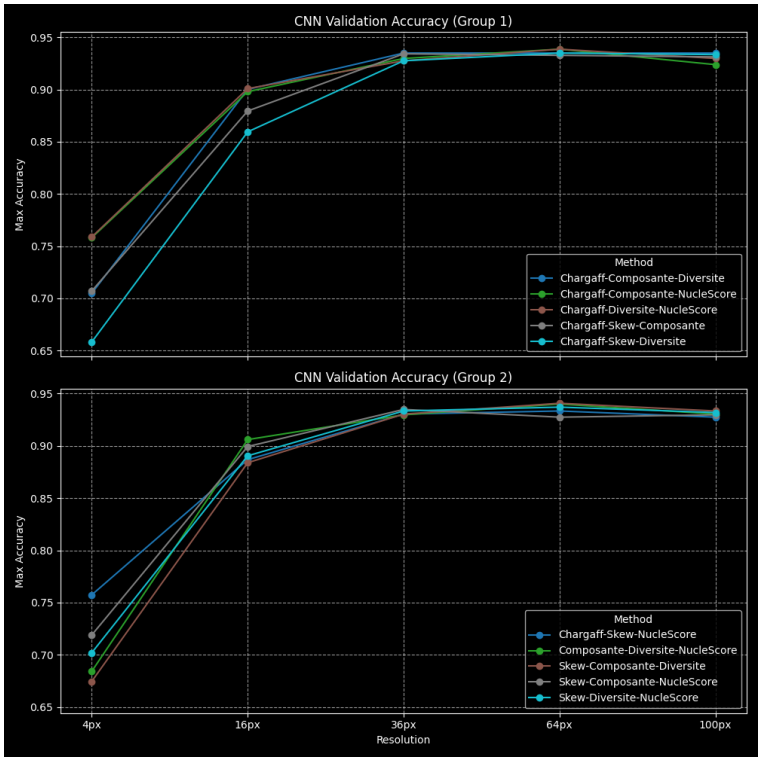

Figure 6: Graph showing the validation accuracy of the CNN model on the different methods at 100px resolution without any preprocessing or balancing techniques applied.

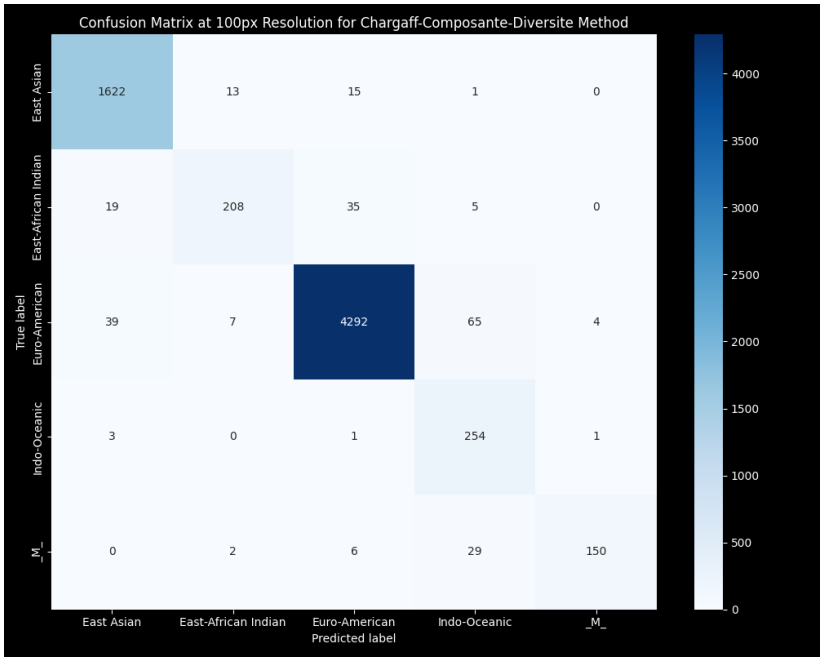

Figure 7: Confusion matrix of the CNN model on the images obtained from the Chargaff-Component-Diversity combination at 100px resolution without any preprocessing or balancing techniques applied.

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.965	0.982	0.988	0.987
East-African Indian	0.904	0.779	0.997	0.988
Euro-American	0.987	0.974	0.976	0.975
Indo-Oceanic	0.718	0.988	0.985	0.985
M	0.968	0.802	0.999	0.994

Table 1: Performance metrics of the CNN model on the images obtained from the Chargaff-Component-Diversity combination at 100px resolution without any preprocessing or balancing techniques applied.

Overall, this model shows good performance on the imbalanced dataset, but there is still room for improvement. But we will explore different techniques to address the class imbalance issue in the following sections.

5.2 Mitigating class imbalance

We have multiple ways of addressing the class imbalance issue in our dataset, including:

- Cross validation
- Under-sampling
- Data augmentation
- Combining under-sampling and data augmentation

We will explore each of these techniques in the following sections and compare the results to see which one yields the best performance.

5.2.1 Cross validation

Cross validation is a technique used to evaluate the performance of a model by splitting the dataset into multiple subsets, or folds. The model is trained on a subset of the data and tested on the remaining data, and this process is repeated for each fold. This allows for a more robust evaluation of the model's performance, as it reduces the risk of overfitting to a specific subset of the data.

Remaking the dataset

During our testing, we realized that we were testing our model with images the model was trained on, which is not a good practice. We decided to split our dataset into a training set and a test set, with 90% of the samples in the training set and 10% in the test set. The files are separated into two folders to avoid any overlap between the training and test sets.

5.2.1.1 Results

During our testing, we used the K-fold cross validation technique with 5 folds with 15 epochs and a batch size of 32. We observed that the model was able to achieve a validation accuracy of around 96% during validation for all of 10 combinations of the metrics at 100px resolution, as shown in Figure 8. The confusion matrix in Figure 9 shows that the model was able to correctly classify most of the samples, with only a few misclassifications.

With Table 2 we can see that the model has good performance across most classes. Surprisingly enough, the M class, which has the least number of samples, has better performance than the East-African Indian class, which has more samples. This suggests that the model is able to generalize well to the minority classes, even with the class imbalance issue.

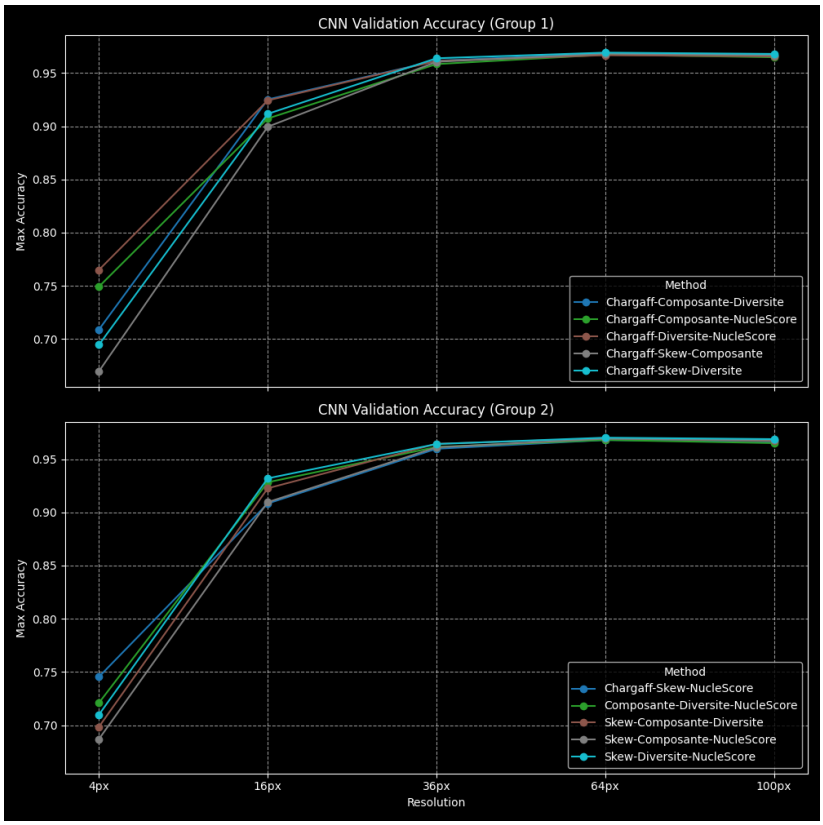

Figure 8: Graph showing the validation accuracy of the CNN model with k-fold cross validation applied on the different methods.

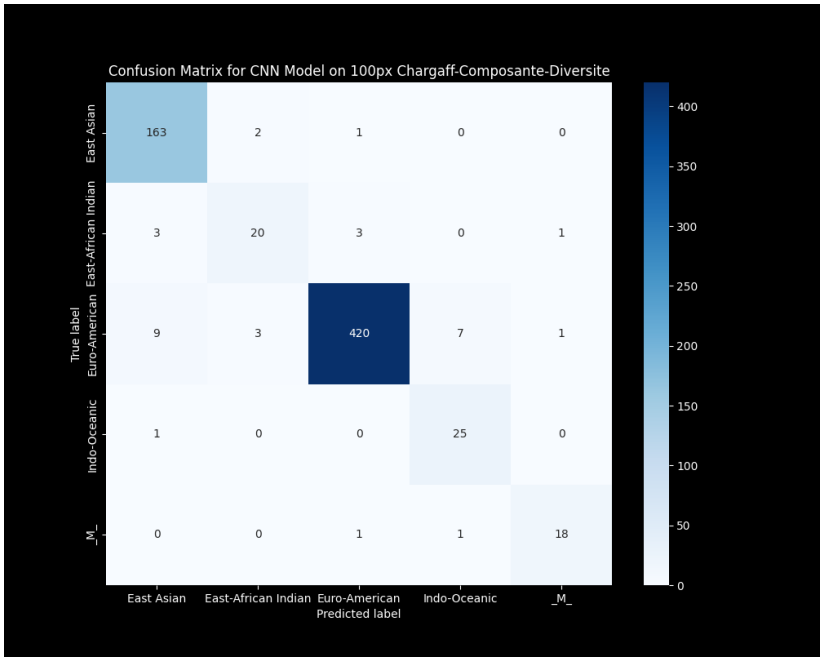

Figure 9: Confusion matrix of the CNN model with k-fold cross validation applied on the images obtained from the Chargaff-Component-Diversity combination at 100px resolution.

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.926	0.982	0.975	0.976
East-African Indian	0.800	0.741	0.992	0.982
Euro-American	0.988	0.955	0.979	0.963
Indo-Oceanic	0.758	0.962	0.988	0.987
M	0.900	0.900	0.997	0.994

Table 2: Performance metrics of the CNN model with k-fold cross validation applied on the images obtained from the Chargaff-Component-Diversity combination at 100px resolution.

For the mosaic images, we applied the same k-fold cross validation technique with 5 folds and 15 epochs, and we observed similar results compared to the individual images. The East-African Indian class has a slightly lower precision but a slightly higher recall, indicating that the model is able to correctly classify most of the samples in this class, but it also misclassifies some samples as other classes. The Indo-Oceanic class has a recall of 1.000, indicating that the model was able to correctly classify all the samples in this class, but it has a lower precision of 0.684, indicating that the model misclassifies some samples as this class. Specificity and accuracy are nearly the same as the individual images, as shown in Table 3.

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.932	0.988	0.977	0.979
East-African Indian	0.724	0.778	0.988	0.979
Euro-American	0.995	0.945	0.992	0.962
Indo-Oceanic	0.684	1.000	0.982	0.982
M	0.944	0.850	0.998	0.994

Table 3: Performance metrics of the CNN model with k-fold cross validation applied on the mosaic images at 100px resolution.

The performances of the model with cross validation are close to the ones obtained without cross validation. It seems to perform slightly better, but the difference is not significant enough to conclude that cross validation improves the performance of the model. However, it is important to note that cross validation is a good practice to evaluate the performance of a model, as it reduces the risk of overfitting to a specific subset of the data and provides a more robust evaluation of the model's performance.

5.2.2 Under-sampling

Under-sampling is a technique used to address the class imbalance issue by reducing the number of samples in the majority classes. In our application, we decided to reduce the number of samples in the Euro-American and East Asian classes to a total of 1000 samples each, while keeping the other classes unchanged. This was done to balance the dataset and ensure that the model is not biased towards the majority classes.

Results

This time we observe a drop in the validation accuracy of the model (Figure 12) compared to the previous results, with the accuracy being between 89% and 93% for the 100px resolution. There is also a significant drop in the performance of the model on the East-African Indian class, while the recall slightly improves, the precision drops significantly, indicating that the model identify other classes as East-African Indian. This is likely due to the reduced number of samples in the Euro-American and East Asian classes, which may have led to a less representative training set for these classes. It can be seen in Figure 13 and Table 4.

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.964	0.982	0.988	0.987
East-African Indian	0.571	0.889	0.972	0.969
Euro-American	0.998	0.936	0.996	0.957
Indo-Oceanic	0.765	1.000	0.988	0.988
M	0.905	0.950	0.997	0.996

Table 4: Performance metrics of the CNN model with under-sampling applied on the images obtained from the Chargaff-Component-Diversity combinations at 100px resolution.

For the mosaic images, we observe an even more significant drop in the precision of the East-African Indian class, which drops to 0.361 from 0.724 with cross validation on these images (Figure 5). The model misclassifies a lot of Euro-American samples as East-African Indian, more than there are actual East-African Indian samples in the dataset (Figure 15).

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.952	0.952	0.984	0.976
East-African Indian	0.361	0.815	0.940	0.935
Euro-American	0.992	0.893	0.987	0.926
Indo-Oceanic	0.765	1.000	0.988	0.988
M	0.864	0.950	0.995	0.994

Table 5: Performance metrics of the CNN model with under-sampling applied on the mosaic images at 100px resolution.

Instead of improving the performance of the model like we hoped, under-sampling led to a drop in the performance of the model, especially on the East-African Indian class. Still, even with the lowest number of samples, the M class keeps stable metrics.

5.2.3 Data augmentation

Results

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.976	0.976	0.992	0.988
East-African Indian	0.759	0.815	0.989	0.982
Euro-American	0.986	0.968	0.975	0.970
Indo-Oceanic	0.812	1.000	0.991	0.991
M	0.950	0.950	0.998	0.997

Table 6: Performance metrics of the CNN model with under-sampling applied on the images obtained from the Chargaff-Component-Diversity combinations at 100px resolution with data augmentation applied on the training set.

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.970	0.976	0.990	0.987
East-African Indian	0.815	0.815	0.992	0.985
Euro-American	0.993	0.970	0.987	0.976
Indo-Oceanic	0.722	1.000	0.985	0.985
M	0.947	0.900	0.998	0.996

Table 7: Performance metrics of the CNN model with under-sampling applied on the mosaic images at 100px resolution with data augmentation applied on the training set.

5.2.4 Combining techniques

Results

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.953	0.970	0.984	0.981
East-African Indian	0.706	0.889	0.985	0.981
Euro-American	0.995	0.952	0.992	0.966
Indo-Oceanic	0.743	1.000	0.986	0.987
M	0.950	0.950	0.998	0.997

Table 8: Performance metrics of the CNN model with combined under-sampling and data augmentation techniques applied on the images obtained from the Chargaff-Component-Diversity combinations at 100px resolution.

Class	Precision	Recall	Specificity	Accuracy
East Asian	0.968	0.922	0.990	0.973
East-African Indian	0.307	1.000	0.906	0.910
Euro-American	0.995	0.864	0.992	0.909
Indo-Oceanic	0.839	1.000	0.992	0.993
M	0.950	0.950	0.998	0.997

Table 9: Performance metrics of the CNN model with combined under-sampling and data augmentation techniques applied on the mosaic images at 100px resolution.

Conclusion

Conclusion

A Additional Figures

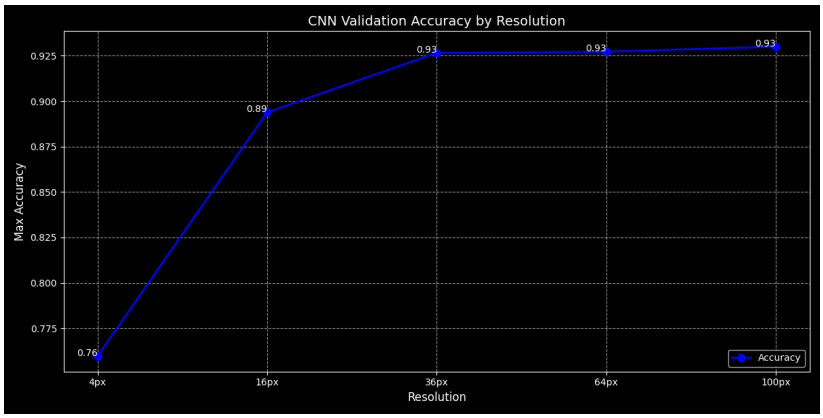

Figure 10: Graph showing the validation accuracy of the CNN model with k-fold cross validation applied on the mosaic images.

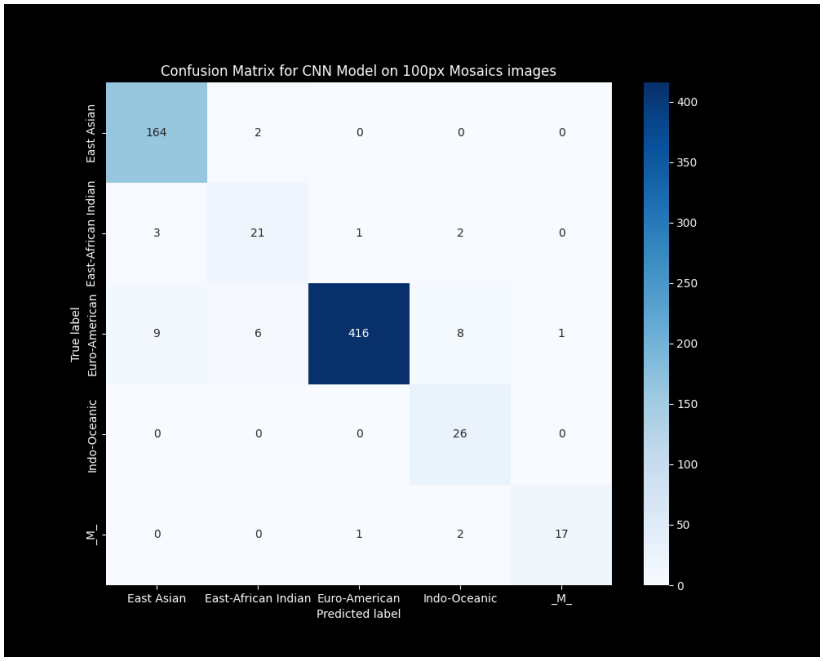

Figure 11: Confusion matrix of the CNN model with k-fold cross validation applied on the mosaic images at 100px resolution.

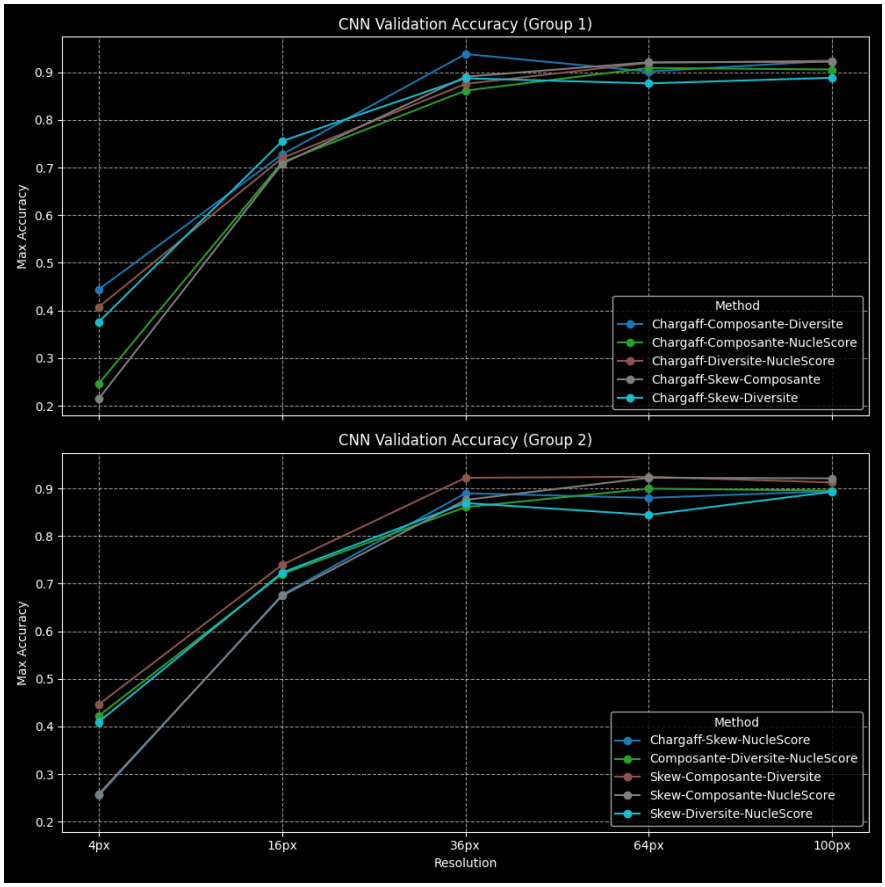

Figure 12: Graph showing the validation accuracy of the CNN model with under-sampling applied on the different methods.

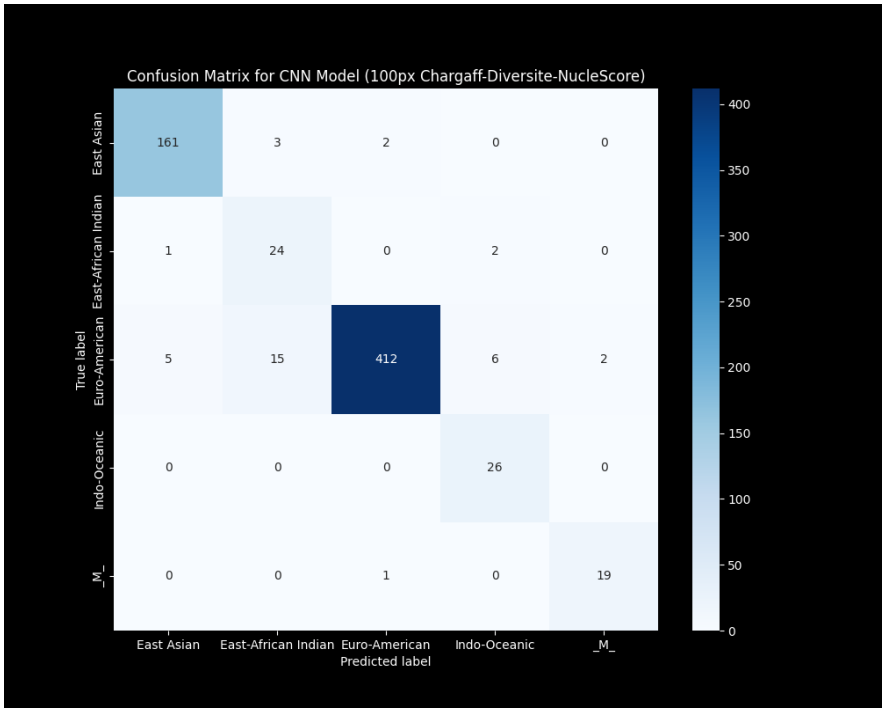

Figure 13: Confusion matrix of the CNN model with under-sampling applied on the images obtained from the Chargaff-Diversity-NucleScore combination at 100px resolution.

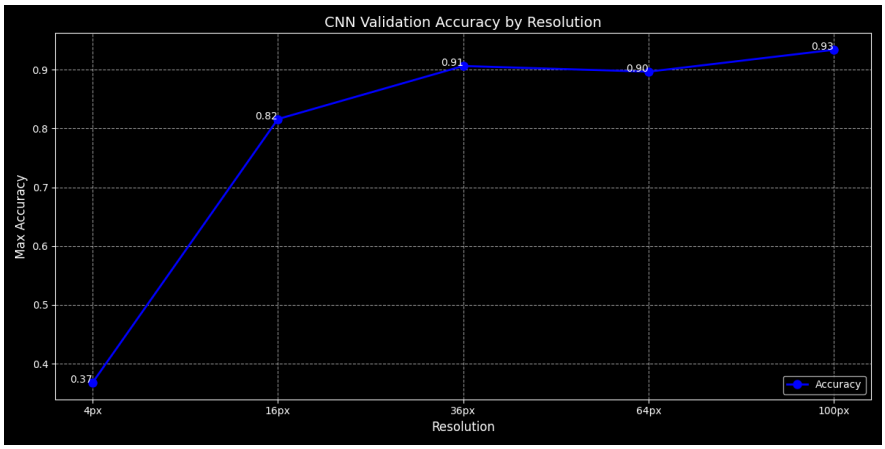

Figure 14: Graph showing the validation accuracy of the CNN model with under-sampling applied on the mosaic images.

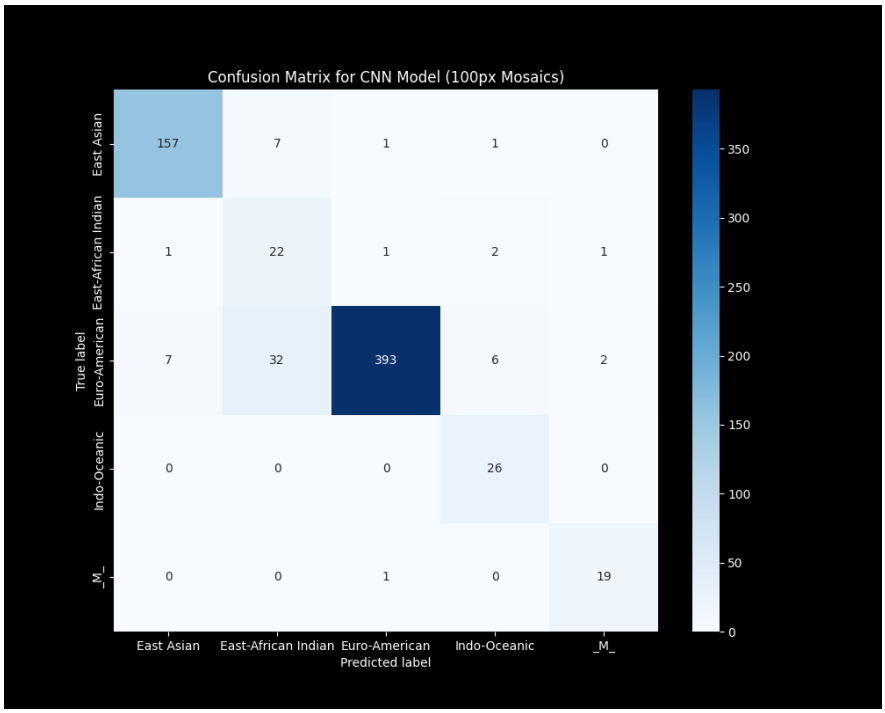

Figure 15: Confusion matrix of the CNN model with under-sampling applied on the mosaic images at 100px resolution.

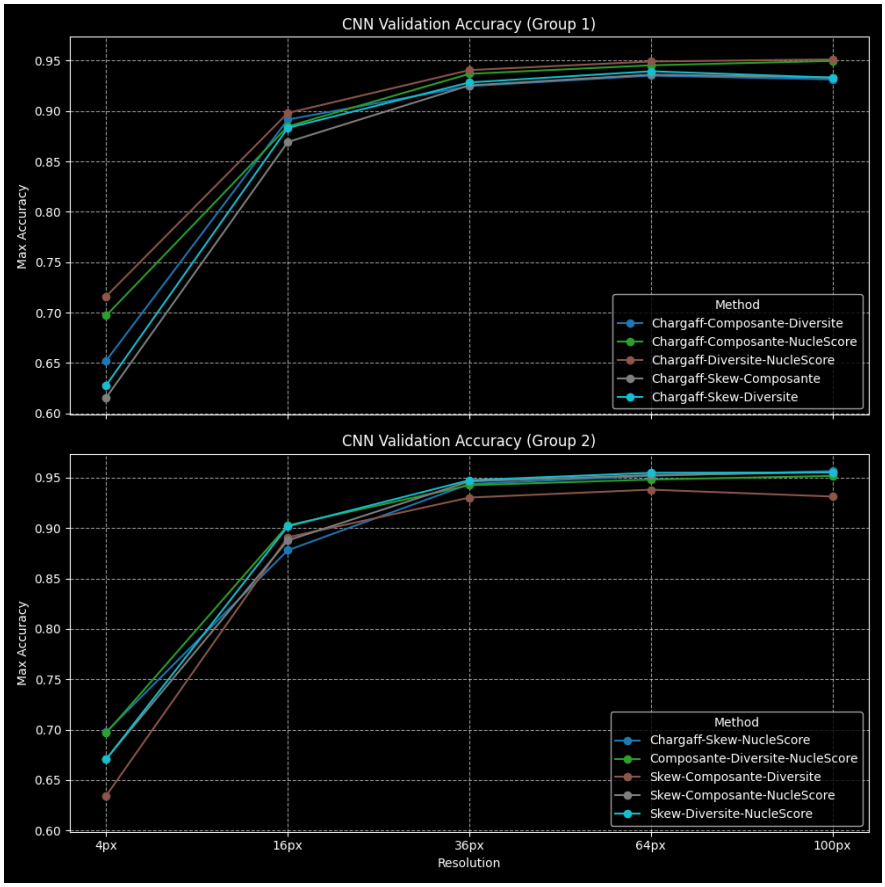

Figure 16: Graph showing the validation accuracy of the CNN model with under-sampling applied on the different methods.

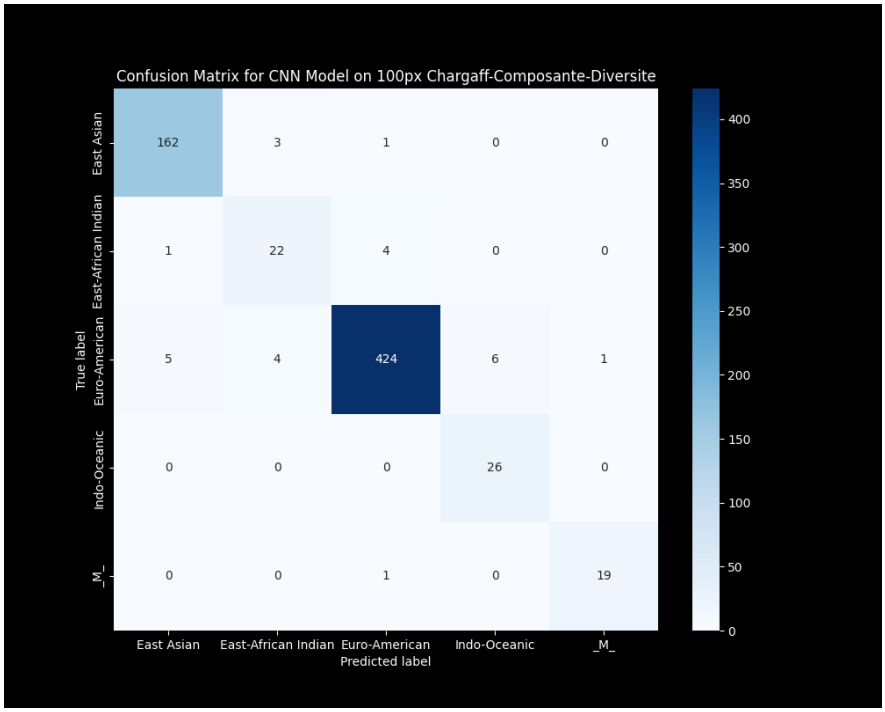

Figure 17: Confusion matrix of the CNN model with under-sampling applied on the images obtained from the Chargaff-Diversity-NucleScore combination at 100px resolution.

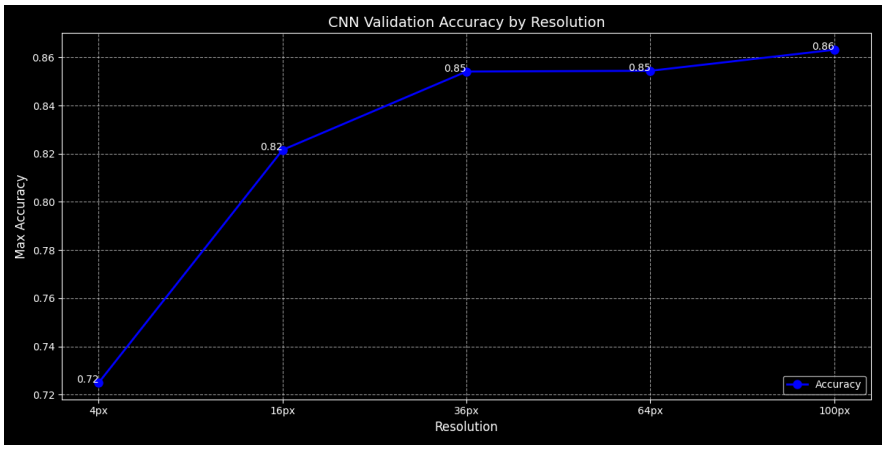

Figure 18: Graph showing the validation accuracy of the CNN model with under-sampling applied on the mosaic images.

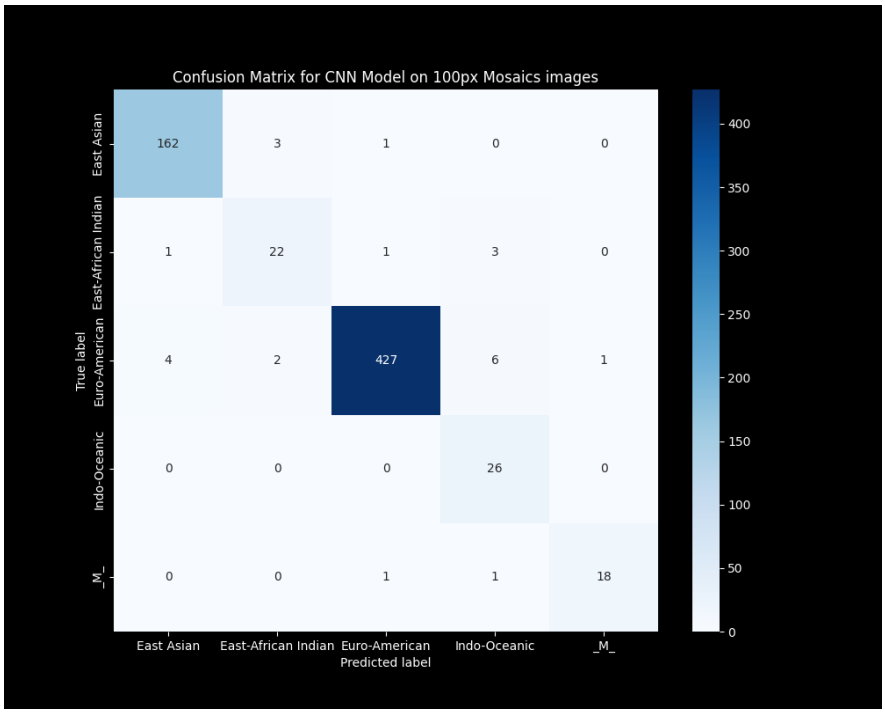

Figure 19: Confusion matrix of the CNN model with under-sampling applied on the mosaic images at 100px resolution.

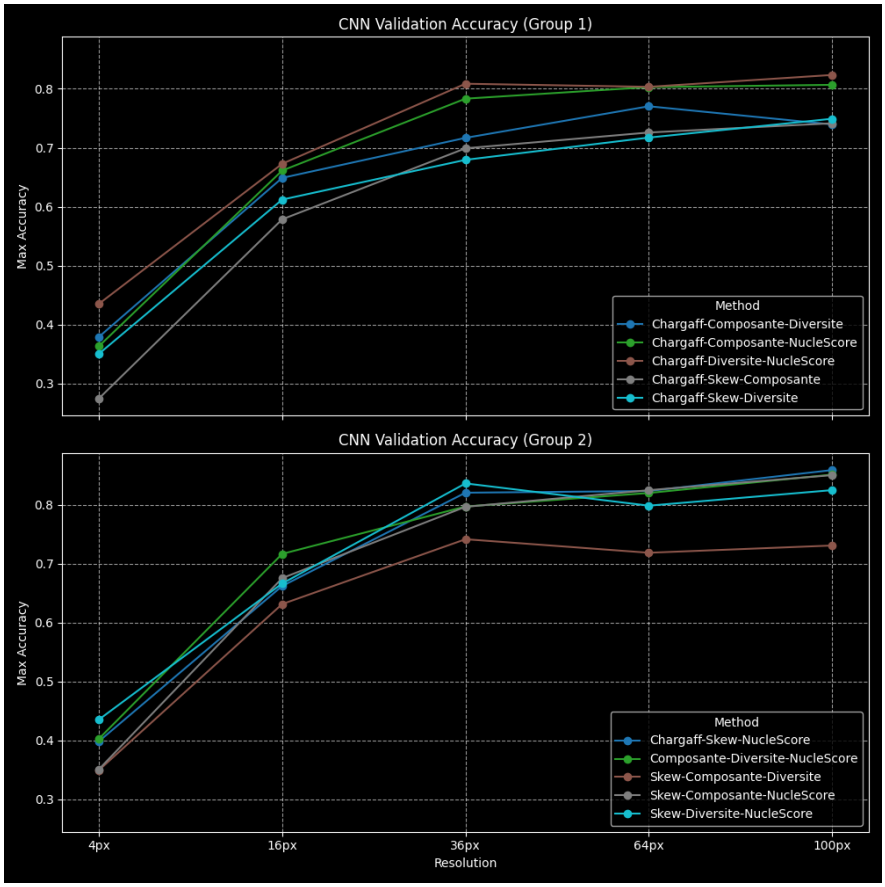

Figure 20: Graph showing the validation accuracy of the CNN model with combined undersampling and data augmentation techniques applied on the different methods.

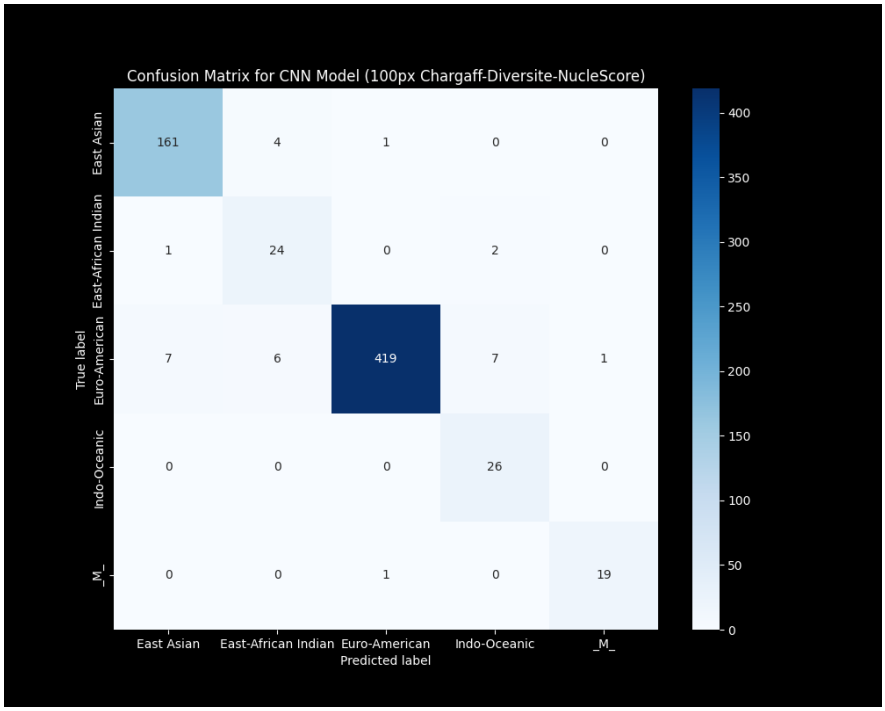

Figure 21: Confusion matrix of the CNN model with combined under-sampling and data augmentation techniques applied on the images obtained from the Chargaff-Diversity-NucleScore combination at 100px resolution.

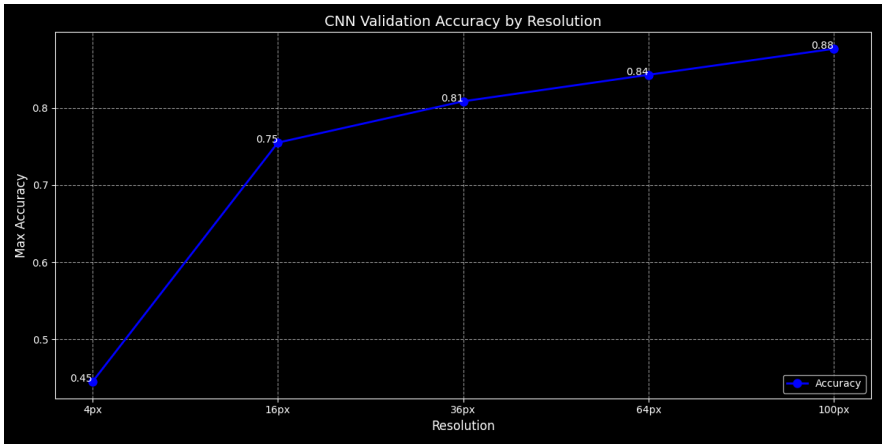

Figure 22: Graph showing the validation accuracy of the CNN model with combined under-sampling and data augmentation techniques applied on the mosaic images.

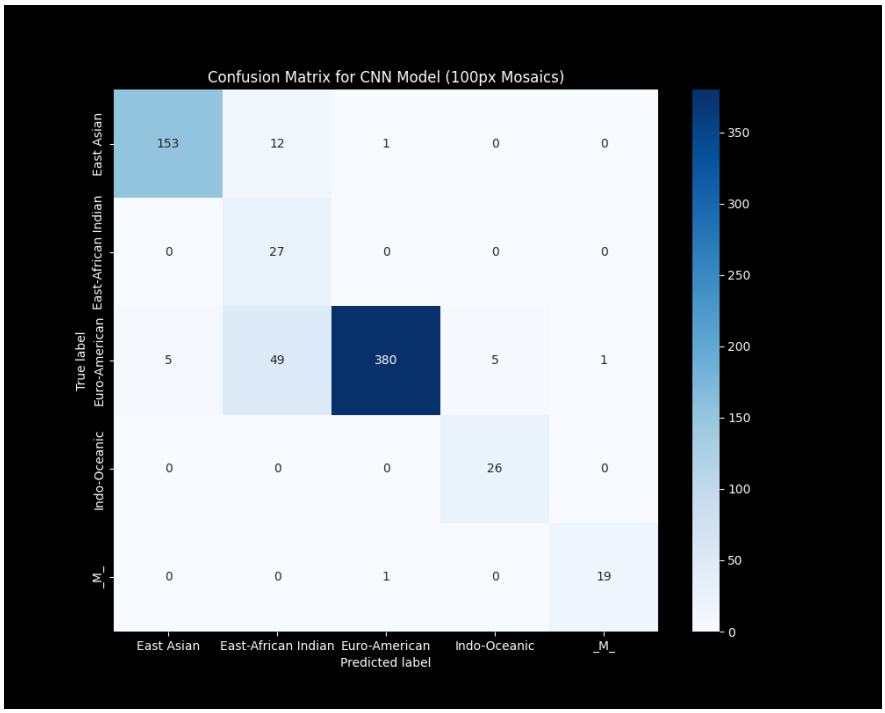

Figure 23: Confusion matrix of the CNN model with combined under-sampling and data augmentation techniques applied on the mosaic images at 100px resolution.

Bibliography

[1] Claudia Caudai et al. "AI applications in functional genomics". In: *Computational and Structural Biotechnology Journal* 19 (2021), pp. 5762–5790. ISSN: 2001-0370. DOI: https://doi.org/10.1016/j.csbj.2021.10.009. URL: https://www.sciencedirect.com/science/article/pii/S2001037021004311.

[2] Xi Chen and Hemant Ishwaran. "Random forests for genomic data analysis". In: *Genomics* 99.6 (2012), pp. 323–329. ISSN: 0888-7543. DOI: https://doi.org/10.1016/j.ygeno.2012.04.003. URL: https://www.sciencedirect.com/science/article/pii/S0888754312000626.

[3] Kunihiko Fukushima. "Self-organizing Neural Network Models for Visual Pattern Recognition". In: *Plasticity of the Central Nervous System*. Ed. by Keiji Sano and Shozo Ishii. Vienna: Springer Vienna, 1987, pp. 51–67. ISBN: 978-3-7091-8945-0.

[4] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[5] Sebastian Raschka and Vahid Mirjalili. "Python machine learning second edition". In: *Birmingham, England: Packt Publishing* (2017), p. 192.

[6] SWASTIK. *Smote for Imbalanced Classification with Python, Technique*. Oct. 2020. URL: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/.

[7] Juexiao Zhou et al. "An AI Agent for Fully Automated Multi-omic Analyses". In: *bioRxiv* (2024). DOI: 10.1101/2023.09.08.556814. eprint: https://www.biorxiv.org/content/early/2024/01/05/2023.09.08.556814.full.pdf. URL: https://www.biorxiv.org/content/early/2024/01/05/2023.09.08.556814.