Fundamentals of Machine Learning Laboratory Report - Chromosomes Recognition

Leonardo Pampaloni
Matteo Tinacci

December 28, 2025

1 Introduction

This project was conceived to be integrated into genome analysis software used in some laboratories. The original system performed a kind of chromosome recognition based on image similarities but didn't incorporate any form of artificial intelligence for recognition. The software was competing for a contract where one of the requirements was to include deep learning systems. Thus, it seemed interesting to combine business with pleasure and embark on this chromosome recognition project.

The original software, without any AI integration, could only achieve a score of about 70%, which was insufficient to ensure a reliable system. This meant that while it saved time, human presence was still required to supervise the work. Our idea was to try to completely automate this process through AI.

2 First shot

To start our work, we needed a dataset, which can be found at the following link: Insert Link Here. This dataset contained many images of karyotypes (images containing all 24 chromosomes) without any preprocessing. They were real images obtained from a laboratory.

2.1 The dataset

The dataset has a structure with two folders: one contains a series of images, and the other has a series of XML files representing the annotations for each corresponding image. A single image, as we can see, contains all 24 chromosomes along with noise and artifacts.

Figure 1: Example of karyotype

The dataset containing these images had to be refined before being fed into the network. Fortunately, annotations were provided for each image, and alongside the class to which each chromosome belonged, the position of each chromosome was also provided. This made it easy to crop each class from each image. In the figure below, we show the segmentation performed.

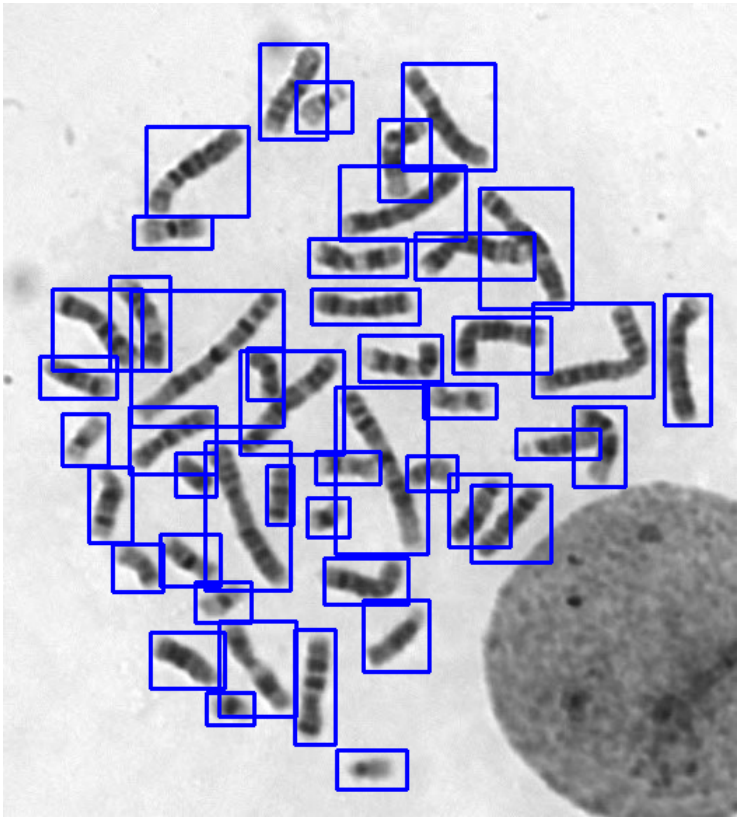


Figure 2: Segmentation performed on the chromosomes

From these bounding boxes, we managed to create subfolders containing chromosomes associated with each respective class, extracted from the original karyotype image. Below, we show how a cropped image of a chromosome appears.

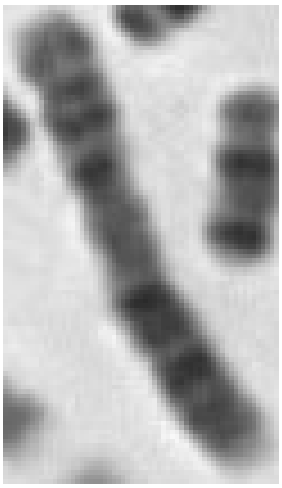


Figure 3: Example of a cropped chromosome

2.2 Preprocessing

At this point, we have the images containing the individual chromosomes ready and classified. What remains to be done is preprocessing the images so that they are ready for the model.

- **Grayscale Conversion**: Convert the images to grayscale to simplify analysis and reduce data complexity.
- **Normalization**: Scale pixel values to a specific range, typically [0, 1] or [-1, 1], to standardize the input data.
- **Cropping**: All images have to be with just one object.
- **Resizing**: Resize images to a consistent size for uniformity across the dataset and to match the input size expected by the model.
- **Noise Reduction**: Apply noise reduction techniques (e.g., Gaussian blur) to minimize the impact of noise and artifacts.
- **Contrast Enhancement**: Use histogram equalization or contrast stretching to enhance the contrast for better feature detection.
- **Data Augmentation**: Apply transformations like rotation, flipping, and scaling to artificially increase the dataset size and improve model generalization.
- **Binarization**: Convert the grayscale images to binary images using thresholding techniques for further feature extraction.

- **Morphological Operations**: Apply morphological operations (e.g., erosion, dilation) to clean up binary images and improve shape features.

In the following images we show what changes have been made.

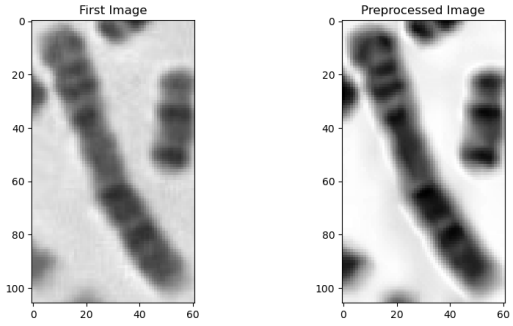


Figure 4: Example of a first preprocessed image

The image was first read in grayscale to reduce its complexity and focus on the essential details. Subsequently, denoising was applied using the Non-Local Means denoising algorithm to remove noise and enhance the visual quality of the image. To improve the visibility of features further, contrast enhancement was performed using CLAHE (Contrast Limited Adaptive Histogram Equalization), which adjusts the contrast by considering the local regions within the image. This process ensures that the image has improved clarity and feature visibility for subsequent analysis. The final result is a contrast-enhanced image that is ready for further processing.

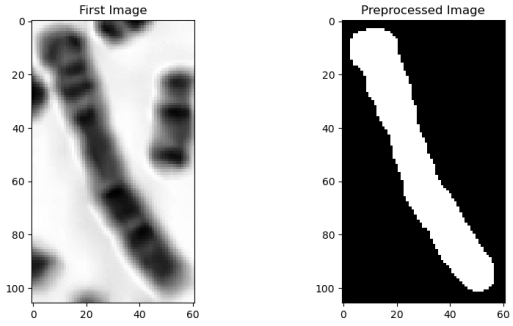


Figure 5: Image with removed objects

The process begins by identifying all connected components within a binary image, including their statistics such as area, using the connectedComponentsWithStats function from OpenCV. The function returns the number of

detected labels, the labels themselves, and statistics about the connected components. Since the first label represents the background, the code skips it and proceeds to identify the label of the largest component based on its area. A mask is then created to isolate this largest component, setting its pixels to white (255) and leaving the rest of the image as black. This mask effectively highlights the largest connected component, making it easy to analyze separately.

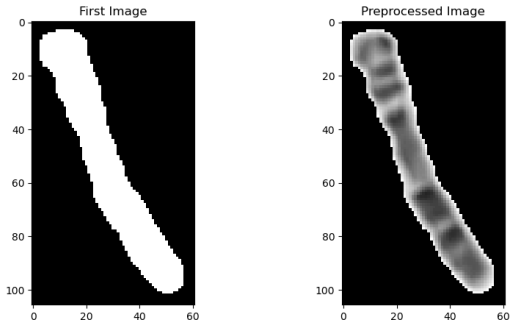


Figure 6: Image with the reconstructed mask

From the largest component we can use it as a mask to extract the original preprocessed image.

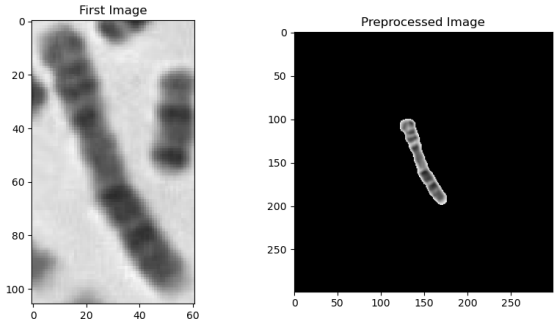


Figure 7: Starter and final image

The final image is ready to be fed into the network, sized 299x299 in the case of Inception v3. All these procedures appear to be correct and ideal for our case. Unfortunately, while these steps have been verified in various situations, due to the low quality of the dataset images, issues such as overlapping often occurred.

2.3 The problem

As we can see in the image, the techniques used so far have proven insufficient for certain chromosomes like this one. Moreover, the images were often of poor quality, leading to this kind of result.

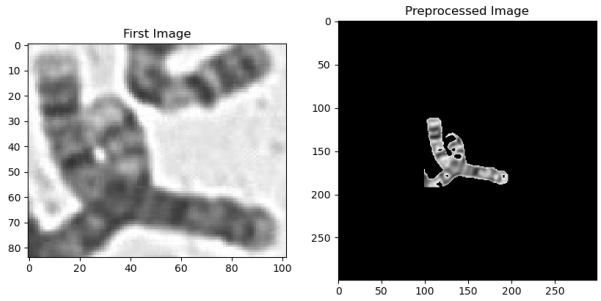


Figure 8: bad image

Our idea was to try using algorithms to avoid possible overlaps, such as the watershed algorithm and some erosion techniques which, when possible, managed to separate certain chromosomes. However, this work became very complex and was not focused on our goal, which was training a network for recognition. For this reason, we attempted to feed the images to the network in their current state but obtained unacceptable results. Consequently, we opted to search for a more suitable dataset.

3 Dataset 2

Searching online for clean datasets that could allow us to train the model better than the previous one, we came across the article *Deep-Learning-Based Human Chromosome Classification: Data Augmentation and Ensemble* [3], which deals with chromosome recognition and classification and exploits the dataset present in the GitHub repository of Lin et al [1]. In particular, this dataset contains 130 images for each type of individual chromosomes that are not overlapped and therefore perfect for our needs.
In particular, as we can see in Figure 9, the images have a black background, are noise-free, and each presents a single chromosome. These images have dimensions of 224x224 in TIFF format, and the last number in their name indicates their classification. For example, "71562828259.067297.18.tiff" is a chromosome corresponding to class 18.

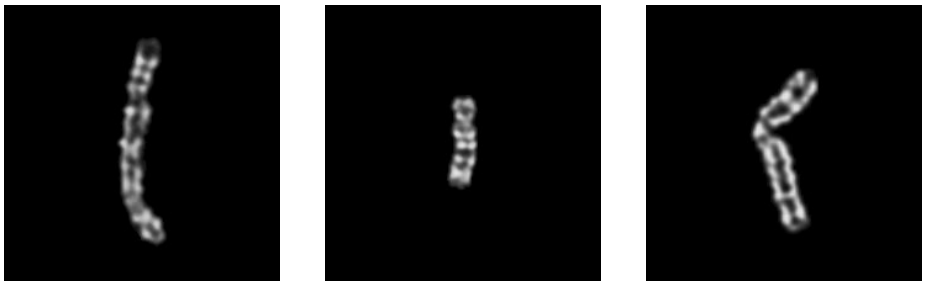


Figure 9: Examples of Chromosomes from the second dataset

The repository also contains the code to train a model capable of recognizing chromosomes, so initially, we thought of implementing it to see if it was very accurate and to understand which path to take. An InceptionResNet v2 model was used, which is a convolutional neural network model that combines two popular approaches: the Inception Network and the Residual Network. It is known for its effectiveness in balancing model complexity with image classification performance.
Unfortunately, we were unable to implement it correctly since the code developed by ChengchuangLin used the TensorFlow library, which caused several issues and forced us to change our approach and decide to use PyTorch. However, the InceptionResNetv2 model is not available in PyTorch, so we chose to implement other models that are similar in terms of features and performance. In particular, we chose to try the InceptionV3, ResNet50, and ResNet18 models, as they are considered comparable to InceptionResNetV2 and VGG16, which is a very good model in the state-of-the-art of chromosome recognition.

In the article published by D'Angelo and Nanni [3], three algorithms are analyzed that allow processing images in a way that trains the models more effectively. In particular, they claim to be able to achieve an area under the ROC curve of 0.9999 for their model. They propose an algorithm for augmentation and two algorithms to straighten bent chromosomes, such as the one shown in Figure 9. The first one allows expanding the dataset, not too large, by applying translation and rotation transformations to the existing images, while the latter, by straightening all the chromosomes vertically, allows the model to better focus on characteristic details of the chromosomes rather than their curvature or orientation. These latter algorithms have the function of calculating a Rotation Score, useful for finding the centromere of the chromosome, which is the narrowest point and where a curvature can occur, and then straightening it. The score is calculated as the global minimum of the horizontal projection vector, and thanks to this, we have the possibility to separate the chromosome into two sub-images that are rotated until obtaining the vertical projection vector with the smallest amplitude, which is associated with the vertically straightened chromosome.

The algorithms are all proposed as pseudocode; given their non-trivial implementation, we only took inspiration to apply augmentation to the dataset, as we observed that even without using the straightening algorithm, the models responded very well.

4 Models and Training

4.1 Preprocessing

Regarding the training of the models, we decided to adapt the dataset to the various models by applying padding and resizing to the images as needed. For example, InceptionV3 requires images of size 299x299. We then applied augmentation to the dataset, composed of the following transformations assigned randomly:

- Contrast, brightness, saturation, and intensity variation to vary the chromatic characteristics of the image.
- Rotation, translation, and scaling to change the position, orientation, and size of the various chromosomes.
- Gaussian blur applied to an image with variable kernel sizes (5x5 to 9x9) and standard deviations (0.1 to 5).

We then randomly split the dataset into Train, Test, and Validation sets, with portions of 70%, 10%, and 20%, respectively.

4.2 Parameters

Regarding the hyperparameters used during model training, we chose to perform a Grid Search by varying the following parameters:

- **Optimizer**: We tested the models with three optimizers: Adam, Stochastic Gradient Descent (SGD), and RMSprop. Adam is an adaptive optimization algorithm that combines the benefits of momentum and RMSprop. SGD optimizer updates model parameters by computing gradients on small batches of data, gradually minimizing the loss function to optimize the model. RMSprop is an adaptive learning rate optimization algorithm that adjusts the learning rate for each parameter based on the magnitude of recent gradients, allowing for effective convergence and improved performance, especially in dealing with sparse gradients.
- **Learning rate**: We tested on a set of three values: 0.0001, 0.001, 0.01.

We used a batch size of 16, as it was the maximum the GPU on which we ran the code could support.

After plotting the graphs related to the grid search and observing the results,

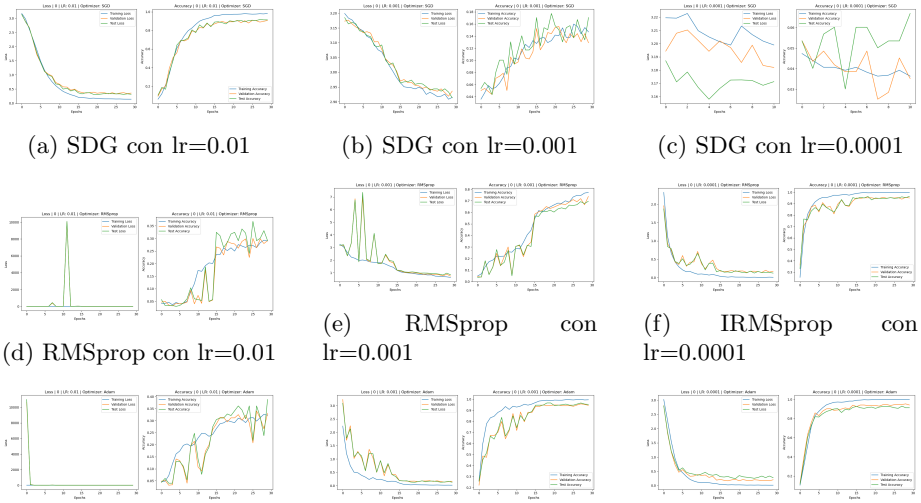
we noticed that they were rather discontinuous with unexpected peaks. Therefore, we decided to implement a learning rate scheduler that would decrease the learning rate as the epochs increased. This technique dynamically adjusts the learning rate during training based on predefined rules or conditions. In particular, we chose a scheduler that multiplies the learning rate by 0.1 every 30 epochs.

Also, observing the graphs during the initial training attempts, we noticed that some models tended to overfit. For this reason, we decided to implement a weight decay system to prevent this problem. Weight decay is a regularization technique used to prevent overfitting by penalizing large weights in the model. In particular, we chose to apply a weight decay of 1e-5.

Finally, since during runs where we trained the models using a very high number of epochs, we decided to implement an early stop system when the model's accuracy stabilized for a certain number of cycles. In particular, we observed the results and chose a threshold of 10 epochs beyond which no stabilized graph would change its value.

4.3 Inception v3

The first model we considered was InceptionV3. InceptionV3 is a deep convolutional neural network (CNN) architecture designed for image classification and feature extraction tasks. It is known for its sophisticated "Inception" modules, which efficiently capture features at different scales using various kernel sizes. InceptionV3 achieves state-of-the-art performance on image classification benchmarks and is widely used in computer vision applications. We implemented the model starting from the pre-trained model available in the PyTorch library and modified its last layer to obtain the desired output of 24 classes, one for each chromosome. In the following images, we will show the behavior of the model during the grid search for parameters.



(a) SDG con lr=0.01 (b) SDG con lr=0.001 (c) SDG con lr=0.0001

(d) RMSprop con lr=0.01 (e) RMSprop con lr=0.001 (f) IRMSprop con lr=0.0001

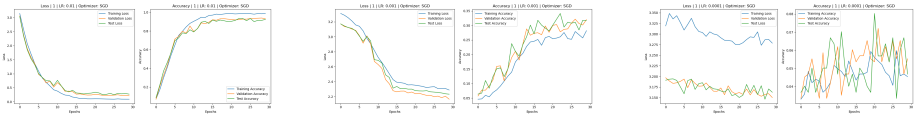
(g) Adam con lr=0.01 (h) Adam con lr=0.001 (i) Adam con lr=0.0001

Figure 10: Grid Search for InceptionV3

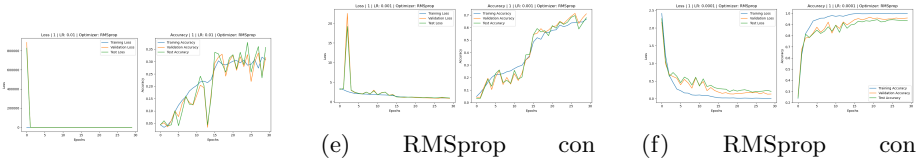
As we can see from the graphs in Figure 10, the best optimizers are SDG and Adam. Observing the loss functions, we can see that for SDG, the best result is achieved when choosing a higher learning rate, while for Adam, the opposite occurs, with a very low learning rate. Regarding RMSprop, we can notice that it also performs decently with a low learning rate, comparable to Adam, but it exhibits spikes during the validation and test phases not present in the other two optimizers, so we can say that it is the worst of the three.

4.4 Inception v3 paper

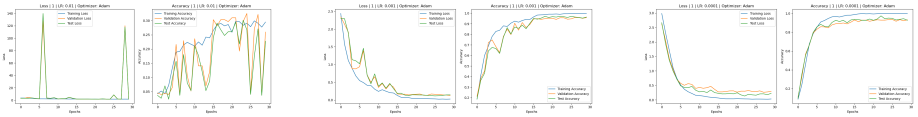
While searching online, we came across a paper [2] published by the University of Stockholm containing a study conducted by Gongchang Chu, proposing an alternative version of InceptionV3 with modified fully connected layers, still aimed at chromosome recognition. We then attempted to modify the network according to the instructions in the article and obtained the following results:



(a) SDG con lr=0.01 (b) SDG con lr=0.001 (c) SDG con lr=0.0001



(d) RMSprop con lr=0.01 (e) RMSprop con lr=0.001 (f) RMSprop con lr=0.0001



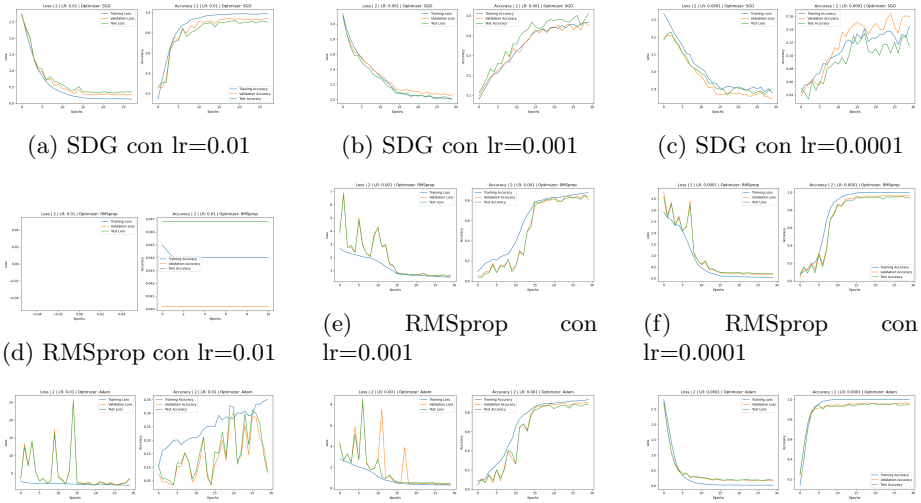
(g) Adam con lr=0.01 (h) Adam con lr=0.001 (i) Adam con lr=0.0001

Figure 11: Grid Search for InceptionV3 paper

Observing the graphs, we can notice that the behavior is rather similar to that of the original model. We do not observe a substantial improvement or worsening but only a lower height of the spikes when using RMSprop with a learning rate of 0.0001 and a slight improvement in accuracy for Adam with the same learning rate.

4.5 VGG16

VGG16 is a convolutional neural network (CNN) architecture renowned for its simplicity and effectiveness in image classification tasks. It consists of 16 layers, including convolutional layers with small 3x3 filters and max-pooling layers, resulting in a deep network capable of learning rich hierarchical features. VGG16 has been widely used as a baseline model in various computer vision applications and is known for its strong performance on standard image classification benchmarks. In this case, we modified the last fully connected layers according to the paper [2] previously mentioned. The following images show the hyperparameter search using Greed Search.



(a) SDG con lr=0.01 (b) SDG con lr=0.001 (c) SDG con lr=0.0001

(d) RMSprop con lr=0.01 (e) RMSprop con lr=0.001 (f) RMSprop con lr=0.0001

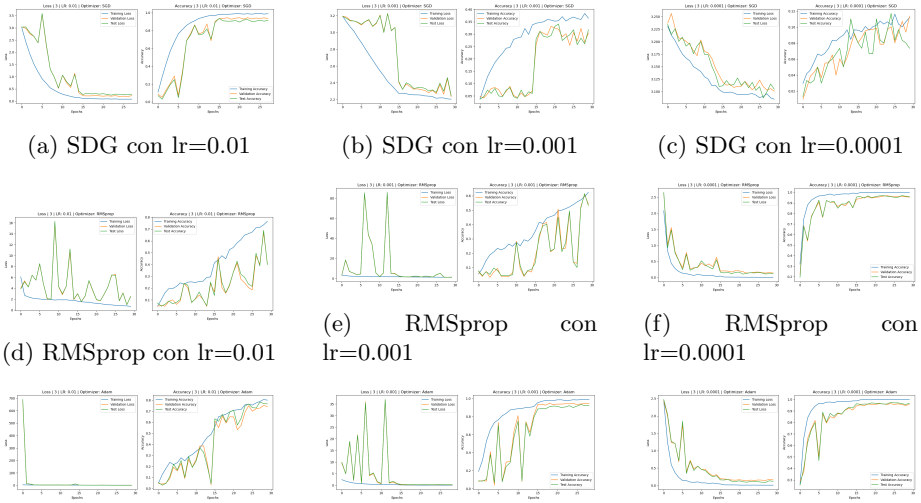
(g) Adam con lr=0.01 (h) Adam con lr=0.001 (i) Adam con lr=0.0001

Figure 12: Grid Search for VGG16

Also in this case, we can notice that the best among the 3 optimizers is Adam with a learning rate of 0.0001, particularly performing slightly better than the previous InceptionV3 models. SGD is good for high learning rates while worsening as the learning rate decreases, but not as much as for the previous InceptionV3 models. RMSprop is a valid optimizer for very low learning rates but cannot keep up with Adam.

4.6 resnet50

ResNet50 is a deep convolutional neural network (CNN) with 50 layers, notable for its introduction of residual learning. Its architecture includes residual blocks with skip connections, enabling effective training of very deep networks by addressing the vanishing gradient problem. With pre-trained weights available, it's widely used in various computer vision tasks like image classification, object detection, and segmentation. Similar to the previous models, we took the pre-trained model available in the PyTorch library and modified the last layer to classify the 24 chromosomes. The following images represent the results of the Grid Search.



(a) SDG con lr=0.01 (b) SDG con lr=0.001 (c) SDG con lr=0.0001

(d) RMSprop con lr=0.01 (e) RMSprop con lr=0.001 (f) RMSprop con lr=0.0001

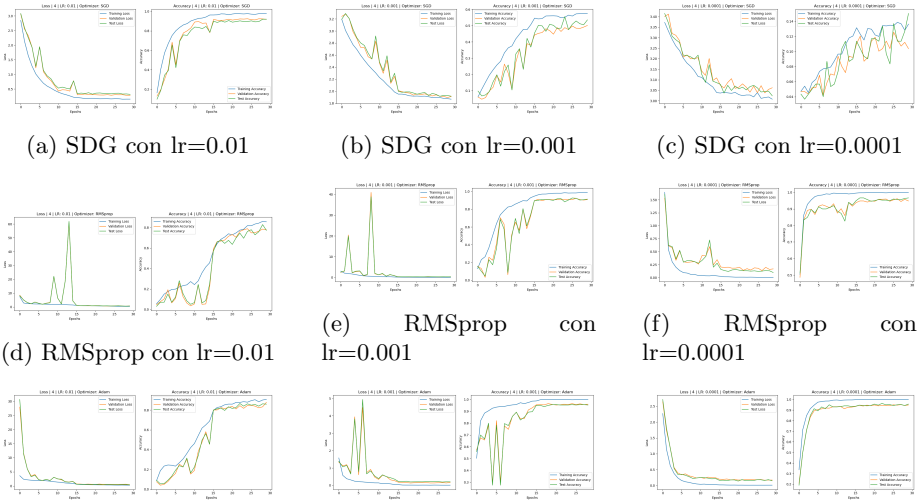
(g) Adam con lr=0.01 (h) Adam con lr=0.001 (i) Adam con lr=0.0001

Figure 13: Grid Search for Resnet50

Observing the graphs, we can notice that unlike the previous models, the most successful optimizer is RMSprop, which performs well for very low learning rates. SGD achieves a very low loss for high learning rates but exhibits rather high spikes, while Adam has excellent results but significantly worse than the previous models.
Since this is a particularly deep network trained with a relatively small dataset, we thought that the model might have "memorized" the images, causing overfitting as the epochs increased. For this reason, we decided to try a lighter version of ResNet, namely ResNet18.

4.7 resnet18

ResNet18 is a lighter variant of ResNet with 18 layers, retaining residual learning principles but designed for scenarios with limited computational resources. Despite its simpler architecture, it offers strong performance in tasks like image classification. It's favored for applications where memory and computation resources are constrained, such as mobile devices and real-time systems.



(a) SDG con lr=0.01 (b) SDG con lr=0.001 (c) SDG con lr=0.0001

(d) RMSprop con lr=0.01 (e) RMSprop con lr=0.001 (f) RMSprop con lr=0.0001

(g) Adam con lr=0.01 (h) Adam con lr=0.001 (i) Adam con lr=0.0001

Figure 14: Grid Search for Resnet18

In this case, the Adam optimizer has excellent results comparable to VGG16 and better than InceptionV3 and resnet50. We can also observe that the other two optimizers are very effective compared to the previous models but still worse than Adam with a learning rate of 0.0001. In particular, we can notice that the assumption made about the previous model may be valid, since the Loss function in the case of resnet50 initially does not have excellent results but suddenly after about 20 epochs, unlike that related to resnet18, which remains very low from the beginning.

4.8 Best model for our case

As observed from the previous analyses, there are several models with their respective hyperparameters that perform at a similar level of accuracy. Therefore, we decided to opt for the practical use of the model that was lighter and faster. For this reason, we chose to perform inference using the ResNet18 model with the Adam optimizer and a learning rate of 0.0001. During the final run, we employed a methodology called early stopping to halt the training epochs when the model ceased to learn further after 10 epochs. We observed that the training typically concluded around the 40th epoch.

4.8.1 Inference

When conducting inference, we aimed to engineer the model loading process to be as straightforward as possible for implementation in external software. To achieve this, we saved the best-performing model during training iterations. Subsequently, we reloaded the model to verify its functionality. All images

fed into the model are preprocessed in the same manner to ensure optimal interpretation by the network.

The model score a stunning 95% on newly tested images, marking a 25% increase compared to the previous version. Nevertheless, reaching a 100% score is not feasible because some tested images exhibit irregularities that are nearly impossible to predict.

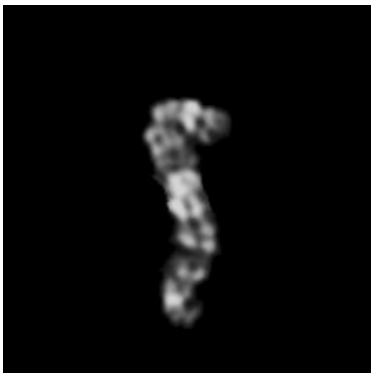


Figure 15: Irregular image

This image needs some state-of-art methods to be classified.

4.9 State of Art

During our work, we applied the most popular and "easy" methods found in the scientific literature. The models and preprocessing strategies we tried are consistent with current practices. However, there are more advanced methodologies to address some challenging situations. As mentioned earlier, some images contain unusable sections, particularly regarding the orientation of chromosomes. Many images are "folded," causing difficulties for the model to learn and predict accurately. Below, we outline a technique that has not been implemented but partially addresses this issue.

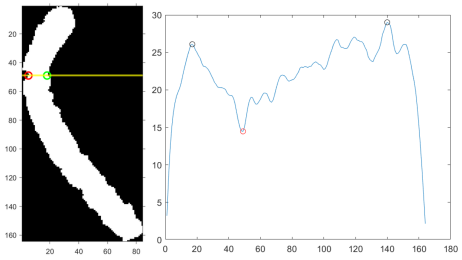
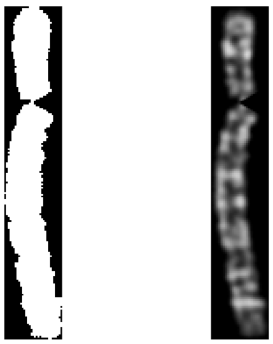


Figure 16: Centromere identification

As we can see, one strategy is to identify the centromere and use it to straighten the chromosome.



(a) (b)

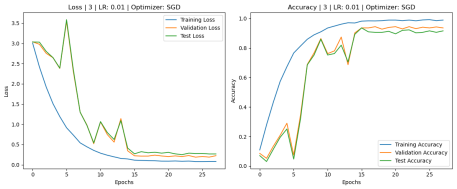
Figure 17: Straightened chromosome

Following what the observed papers report, this strategy can achieve an accuracy of 99%.

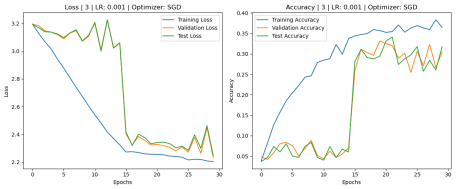
The other aspect that differs significantly from our work is the dataset used. In all the studies we researched, much larger and more refined datasets were utilized. These datasets typically contained images that had to adhere to a certain format, and those that did not were removed. Generally, the image quality and resolution were higher since they employed better acquisition technologies. Due to computational constraints and availability issues, we opted for a small but clean dataset without overlaps.

5 A small API

To get an idea of a possible application, we thought of implementing a small app using the Flask framework. This app uses the models we saved after training with the best hyperparameters found through Greed Search to obtain a classification from an image uploaded at will.
To call the API, we created a small HTML file from which you can upload an image and get its classification.



(a) SDG con lr=0.01



(b) SDG con lr=0.001

Figure 18: Esempi di classificazione con API

As we can see from the images, the app works correctly and is able to classify the chromosomes very well, as we expected from the accuracy results we found during testing.

6 Conclusion

Possible future implementations of the system are numerous. It would be very interesting to try using Meta's SAM 3.0, the new network for segmentation that could be used to appropriately segment chromosomes in a sparse karyotype image. This could help solve some issues such as overlaps, although it would increase the complexity and size of the package to be integrated into other projects.

It would also be interesting to study a way to resolve overlaps. When two chromosomes are overlapped, it not only presents a problem in segmentation but also a challenge in their identification, as while one can be reconstructed, the other would have a gap in the image. It could be interesting to use models for image inpainting to reconstruct missing parts of the image.

In conclusion, this system represents an excellent starting point for its integration into genome analysis software. The next study, to be conducted outside of this project, will focus on adapting the network and addressing the challenges associated with a dataset used in a real-world scenario and its integration into larger-scale software.

7 Bibliography

References

[1] ChengchuangLin. Cir-net. GitHub repository, 2021. URL: `https://github.com/CloudDataLab/CIR-Net.git`.

[2] Gongchang Chu. Machine learning for automation of chromosome based genetic diagnostics. *DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE*, 2020.

[3] Mattia D'Angelo and Loris Nanni. Deep-learning-based human chromosome classification: Data augmentation and ensemble. MDPI, 2023. URL: `https://www.mdpi.com/2078-2489/14/7/389`.