Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Deep Learning for Raman Spectroscopy: A Study of CNNs and Vision Transformers for COVID-19 Classification

Authors:
Nicolò Nicholas Zagami 829888
Vincenzo Pallini 907303

February 21, 2025

Abstract

Raman spectroscopy is a non-invasive analytical technique that provides insights into biological samples at the molecular level. In the context of medical diagnostics, it offers significant potential to detect diseases such as COVID-19 by analysing biochemical changes. This project integrates Raman spectroscopy with deep learning models to develop an automated diagnostic system capable of classifying patients into three categories. To improve the quality of spectral data, pre-processing techniques such as outlier detection, background correction, despiking and data augmentation have been applied. Three convolutional neural network (CNN) architectures - VGG, ResNet and DenseNet - were investigated to assess their ability to extract meaningful spectral features. In addition, a Vision Transformer for 1D spectral data (ViT1D) was implemented from scratch to investigate the feasibility of transformer-based architectures as an alternative to CNNs. The experimental results show that DenseNet achieved the highest accuracy (87.4%), followed by VGG (86.8%) and ResNet (85.6%). While CNN-based models showed strong performance, the Vision Transformer approach offers a promising alternative for spectral classification, warranting further exploration with larger datasets and optimised architectures. These results highlight the potential of deep learning in Raman-based medical diagnostics, providing an accurate and efficient tool for disease classification.

Contents

1 Introduction

Raman spectroscopy is a powerful analytical technique that analyzes molecular composition through the inelastic scattering of light. When light interacts with a sample, a small portion of the photons experience energy changes as a result of molecular vibrations, creating a unique spectral signature characteristic of the sample's molecular structure.

In medical diagnostics, Raman spectroscopy offers significant advantages as a non-invasive, label-free method capable of detecting subtle biochemical changes in biological samples. These capabilities make it particularly valuable for disease diagnosis, as it can identify molecular alterations associated with pathological conditions before clinical symptoms appear.

The COVID-19 pandemic has highlighted the need for rapid and accurate diagnostic tools. Raman spectroscopy presents an alternative approach that could provide insights into disease progression through molecular characterization of biological samples.

This project combines Raman spectroscopy with Convolutional Neural Networks (CNNs) to develop an automated COVID-19 diagnostic system. The proposed approach aims to classify patients into three categories based on their Raman spectral profiles, leveraging CNNs' proven capabilities in pattern recognition and feature extraction. This integration of spectroscopic analysis with deep learning techniques seeks to establish a reliable and efficient tool for COVID-19 diagnosis and patient stratification.

Lastly, a vision transformer for 1D spectral data (ViT1D) has been implemented to explore transformer-based architectures as a potential alternative to CNNs and to assess their feasibility for Raman spectroscopy analysis. This integration of spectroscopic analysis with deep learning techniques aims to establish a reliable and efficient tool for COVID-19 diagnosis and patient stratification.

2 Dataset

The data set comprises 2400 Raman spectra acquired from saliva samples from 101 patients, a minimally invasive biofluid that shows high diagnostic potential for COVID-19 and neurodegenerative diseases [1]. Each patient contributed an average of 24 spectra, with a minimum of 15 spectra and a maximum of 24. Participants are categorized into three groups:

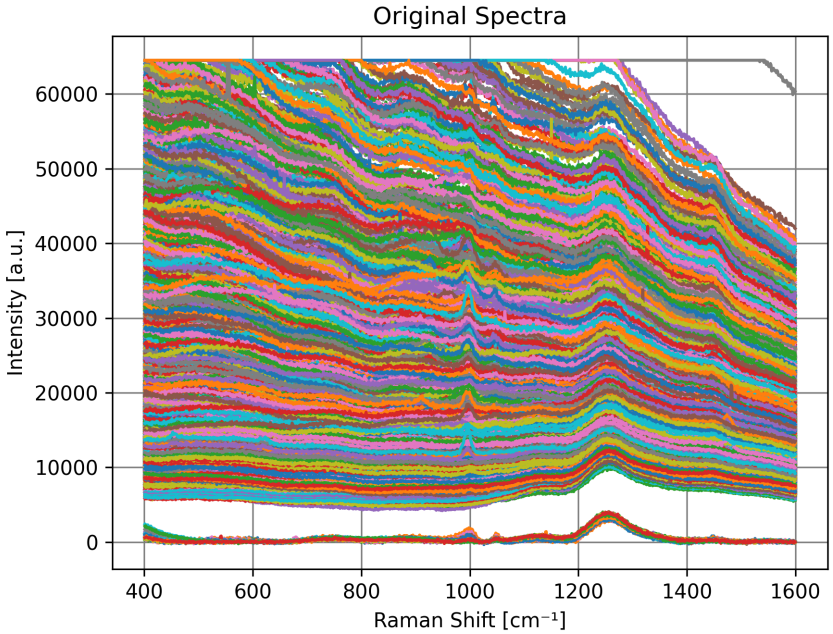
- **COV+**: COVID-positive patients (n=30), total spectra: 720,
- **COV-**: Post-COVID negativized patients (n=37), total spectra: 888,
- **CTRL**: Healthy controls without prior infection (n = 34), total spectra: 792.

Table 1: Patient and spectral distribution across classes.

Category	**COV+**	**COV-**	**CTRL**
Patients	30	37	34
Spectra	720	888	792

Each spectrum contains metadata and intensity values:

- `user`: Unique patient identifier,
- `category`: Class label (COV+, COV-, CTRL),
- `spectra`: Raman intensity values (arbitrary units),
- `x-axis`: Raman shift values (cm^{-1}).



3 Preprocessing phase

3.1 Outlier detection

The outlier detection methodology implemented in this study focuses on identifying and removing saturated spectra from the dataset. Saturation in spectroscopic data typically manifests **as sequences of identical intensity values above a certain threshold**, which can indicate detector saturation or other instrumental artifacts that compromise data quality.
The algorithm is implemented with a *saturation threshold* of 60,000 intensity units. This specific threshold enables the selection of spectra to be removed. This approach effectively identifies the spectra in which the detector has reached its maximum capacity to measure the intensity accurately. The implementation processes each spectra individually, checking for sequences of repeated values that exceed the threshold.

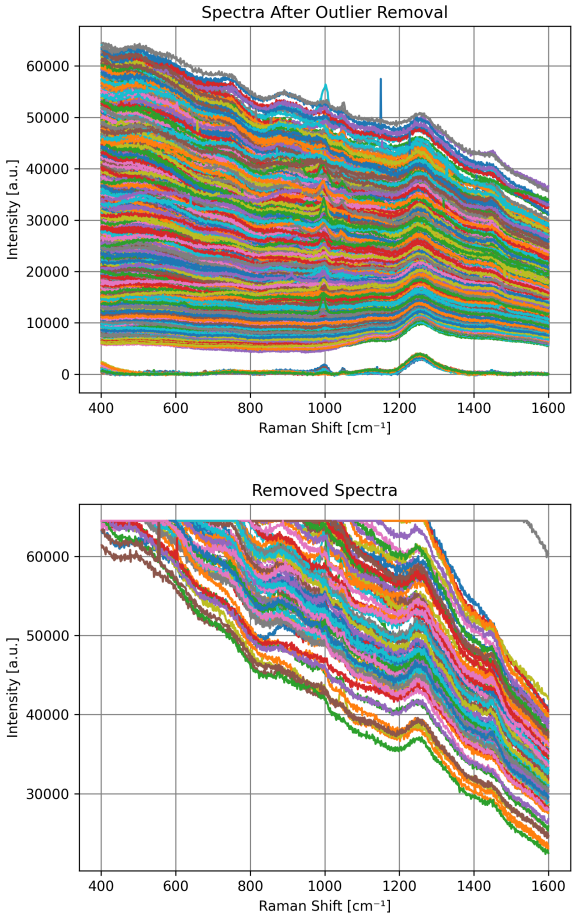


Figure 1: The top image represent the final result without outliers, on the bottom instead there are just spectra considered as outliers.

87 spectra are removed from the original dataset.

3.2 Background Correction: Polynomial Fitting

Background correction is a critical preprocessing step in Raman spectroscopy, as various phenomena can compromise the prediction; therefore, polynomial fitting[4] is employed to mitigate this factor and reveal the true Raman spectral features.
The implementation uses a third-degree polynomial function to model and remove the background through the following steps:

- Initialize the array of indices of every spectra
- Find a "polynomial" that approximate the trends of our spectra giving back coefficents
- Once the coefficients are obtained, the expected background is computed using the polynomial evaluation function (`polyval`)

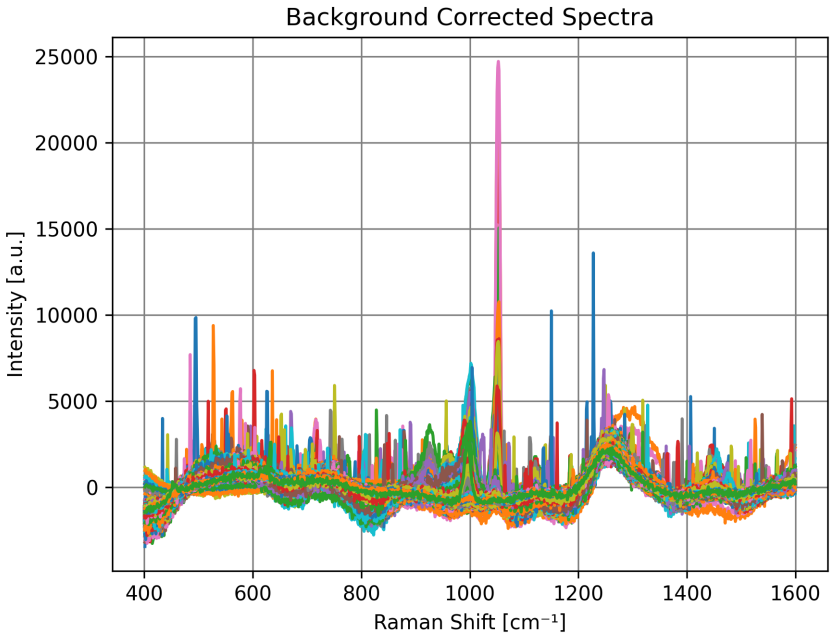


Figure 2: Spectras after the background correction.

3.3 Despiking: Hayes-Whitaker Algorithm

Despiking is a crucial preprocessing step in Raman spectroscopy since cosmic spikes, appearing as random, narrow positive peaks, can significantly interfere with subsequent multivariate analysis [6].
The implementation has the following steps:

- **Spike Detection**: To identify anomalous spikes in the given spectra, the modified Z-score for each point of the spectra is computed. A point is classified as a spike if its modified Z-score exceeds a predefined threshold.
- **Spike Removal Using Neighbor**: Once spikes are detected, they are replaced using neighboring values. The number of neighbors considered is determined by a fixed parameter called `neighborhood_size`. The spike is substituted with the **mean of its valid neighbors**, ensuring a smooth correction without distorting the overall spectra.

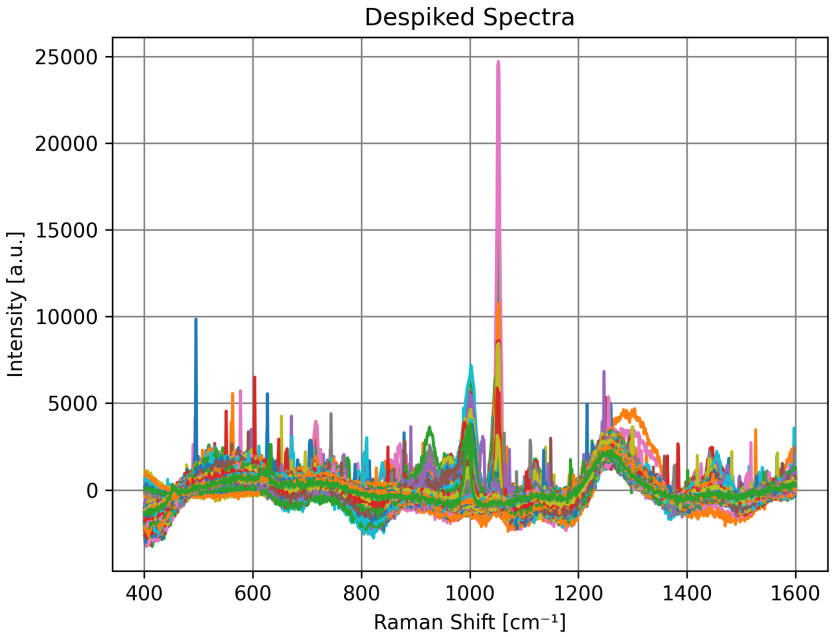


Figure 3: Spectras after the Despiking process.

3.4 Data Augmentation

Due to the limited amount of original data, data augmentation is applied to generate additional synthetic spectra and improve prediction performance. This is achieved by introducing random variations to each spectra in the dataset [2].
For every original spectra, **7** augmented versions are generated by applying the following transformations:

- **Offset**: Shifts the data points slightly up or down relative to the original spectra.
- **Slope**: Modifies the inclination of the spectra.
- **Multiplicative factor**: Introduces small variations, adding controlled noise to the augmented spectra.

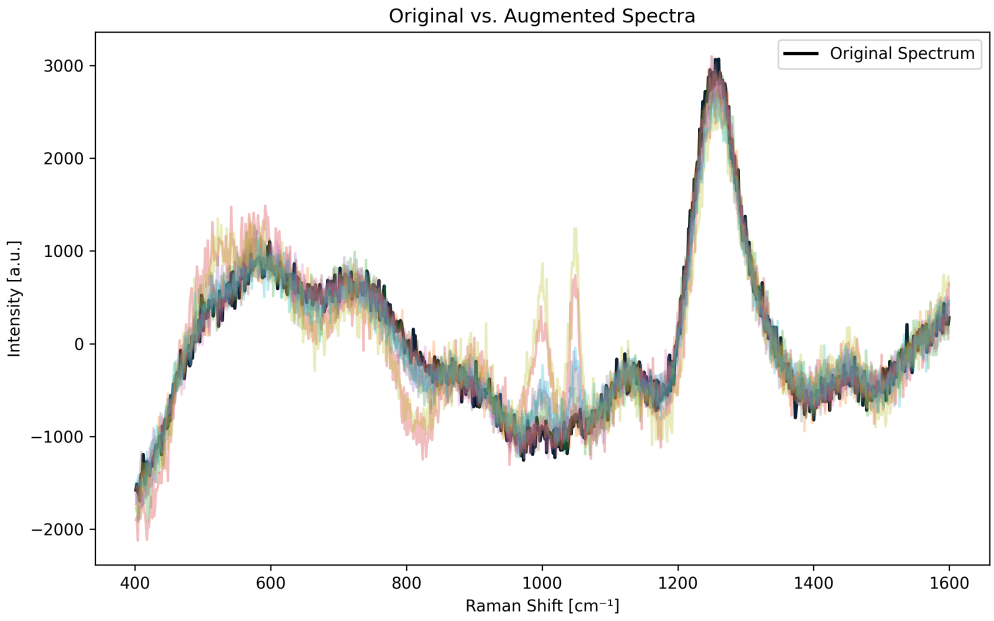


Figure 4: The difference between original and augmented spectras.

3.5 Dataset Splitting And Normalization

The data are normalized using `StandardScaler()`, and the classes are encoded with `LabelEncoder()`. The dataset is split 70/30, where 30% is further divided into 15% **validation** data and 15% **test** data.

4 Models

Initial parameters of each model such as the number of filters, dense layer units, dropout rates, and learning rates are chosen by a trial and error process. After the preliminary models are built, hyperparameter tuning (using `keras-tuner` by 10 trials) was applied to systematically optimize these settings and **improve performance**. The following sections present three architectures (inspired implementation) that have been modified to process one-dimensional Raman spectral data: *VGG*, *ResNet*, and *DenseNet*.

4.1 VGG

The VGG model is implemented as a 1D convolutional neural network inspired by the classical VGG architecture. In the **trial phase**, the network was configured as follows:

- **Convolutional Layers:** The first convolution block employed 32 filters, the second block used 16 filters, and the third block had 8 filters (all with a kernel size of 3).
- **Fully Connected Layers:** Two dense layers with 128 and 64 units, respectively.
- **Regularization:** Dropout layers (with a rate of 0.5) and Batch Normalization were applied.
- **Optimization:** The learning rate was set to 10^{-4}.

Table 2: Hyperparameter Tuning Ranges for the VGG Model

Parameter	**Min Value**	**Max Value**	**Step**
First Block Filters	16	64	16
Second Block Filters	8	32	8
Third Block Filters	4	16	4
Dense Layer 1 Units	64	256	64
Dense Layer 2 Units	32	128	32
Dropout Rate 1	0.1	0.7	0.1
Dropout Rate 2	0.1	0.7	0.1

The learning rate was selected from the discrete set $\{10^{-3}, 10^{-4}, 10^{-5}\}$.

During the **tuning phase**, hyperparameter optimization leads to the following modifications:

- **Convolutional Layers:** The first block was updated to 64 filters while the second and third blocks were maintained at 16 filters each.

- **Fully Connected Layers:** The dense layers were revised to 256 and 128 units.
- **Regularization:** Dropout rates were reduced to 0.1 and 0.2, respectively.
- **Optimization:** The learning rate was increased to 10^{-3}.

4.2 ResNet

The ResNet model leverages residual blocks with shortcut connections to enhance gradient flow and facilitate deeper network architectures. In the **trial phase**, the network was designed as follows:

- **Initial Layer:** A convolutional layer with 8 filters and a kernel size of 7, followed by Batch Normalization and ReLU activation.
- **Residual Blocks:** The network is structured in four stages with residual blocks composed of 3, 4, 6, and 3 layers, respectively, using a base filter size of 8.
- **Fully Connected Layer:** A single dense layer with 128 units.
- **Optimization:** A learning rate of 10^{-5} was employed with no dropout.

Table 3: Hyperparameter Tuning Ranges for the ResNet Model

Parameter	**Min Value**	**Max Value**	**Step**
Initial Convolution Filters	8	64	8
Base Filter Size	8	64	8
Dense Layer Units	64	256	64
Dropout Rate	0.1	0.5	0.1

The learning rate was selected from the discrete set $\{10^{-3}, 10^{-4}, 10^{-5}\}$.

During the **tuning phase**, hyperparameter optimization leads to the following modifications:

- **Initial Layer:** A convolutional layer with 30
- **Residual Blocks:** The network structure remain the same, but the base filter size is increased to 32
- **Fully Connected Layer:** Units of Dense layer were doubled (256).
- **Optimization:** The dropout rate increases to 0.5

4.3 DenseNet

The DenseNet architecture exploits dense connectivity by ensuring that each layer within a Dense Block receives the concatenated outputs of all preceding layers. In the **trial phase**, the network was configured as follows:

- **Initial Layer**: A convolutional layer with 30 filters.
- **Dense Blocks**: Two dense blocks were used, each containing 20 filters
- **Compression Factor**: Set to 0.2 for each layer that contains it
- **Regularization**: A dropout rate of 0.2 was applied.

Table 4: Hyperparameter Tuning Ranges for the DenseNet Model

Parameter	**Min Value**	**Max Value**	**Step**
Initial Convolution Filters	16	64	8
Number of Dense Blocks	2	5	1
Filters per Block	16	64	8
Compression Factor	0.2	0.6	0.1
Dropout Rate	0.1	0.5	0.1

The learning rate was selected from the discrete set $\{10^{-3}, 10^{-4}, 10^{-5}\}$.

During the **tuning phase**, hyperparameter optimization produced the following updates:

- **Initial Layer:** The number of filters was reduced to 24.
- **Dense Blocks:** The architecture was enhanced by increasing to 3 dense blocks, each with 24 filters.
- **Compression Factor:** Adjusted to 0.3.
- **Regularization:** Minor refinements were made to the dropout rates.

4.4 Bonus model: Vision Transformer 1D

A custom *1D Vision Transformer* architecture was implemented to explore its potential for Raman spectroscopy classification tasks. These Python classes inherit from `torch.nn.Module` and build a common structure of two methods:

- __init__: constructor that usually defines the layer of the model
- forward: Defines how the input "go" through the layers.

4.4.1 PatchEmbeddingAndPositionalEncoding1D

While Vision Transformers traditionally process 2D image inputs, this implementation adapts the architecture for 1D spectral data. The Raman spectra are segmented into equal-length patches based on a defined `patch_size`.
To ensure uniform patch dimensions, padding is applied when the total spectrum length **is not divisible** by the `patch_size`. These patches are then projected into an embedding space for transformer processing.

4.4.2 TransformerEncoderBlock

The Transformer Encoder block is not changed from the original implementation, it is composed by two main blocks:

- **MSA Block**: The self-attention mechanism allows each patch to attend to every other patch in the sequence. This means that the transformer can model long-range dependencies and relationships between different parts of the image [5]
- **FFN Block**: After the Multi-Head Self-Attention (MSA) block, the Vision Transformer processes data through a Feed-Forward Neural Network (FNN) block, which serves the same purpose as the MLP block in standard ViT architectures.

4.4.3 ViT1D

The final architecture integrates the previous elements and make some of them configurable: the *MultiheadAttention* layer with adjustable number of heads (`num_head`) and multiple *TransformerEncoderBlocks*, whose quantity is determined by the depth parameter. The model concludes with three sequential components: a **normalization layer**, followed by **global average pooling** to compress spatial dimensions into a vector, and finally an **MLP head** for class prediction.

4.4.4 Training pipeline

- **Data Preparation**: The training (x_train, y_train) and testing (x_test, y_test) data are converted into PyTorch tensors and loaded into *DataLoader* objects for efficient batching
- **Model**: The model is moved to GPU (if available), and *CrossEntropyLoss* is used as the loss function, optimized with *Adam*
- **Training Loop**: compute Forward pass, Loss and accuracy and backpropagation
- **Evaluation**: The model switches to eval mode and computes test loss and accuracy
- **Visualization**: plot of accuracy and loss

5 Metrics

The metrics used in the training pipeline are as follows:

- **Overall Accuracy**: Obtained from the model evaluation and tracked during the fitting process.
- **Classification Report**: Includes metrics such as Precision, Recall, and F1-Score.
  - **Precision**: The proportion of positive predictions that are actually correct

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

  - **Recall**: The proportion of actual positive instances that were correctly identified by the model.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

  - **F1-Score**: The harmonic mean of precision and recall, giving a balance between the two. It's especially useful when the data is imbalanced.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

- **Confusion Matrix**: A detailed breakdown of the predictions, providing the raw data used to calculate the metrics in the classification report.

6 Results

Tuning phase significantly improves model performance, with ResNet showing the most significant **increase from 64.0% to 85.6% accuracy**. Similarly, VGG and DenseNet reached strong final accuracies of 86.8% and 87.4%, representing substantial improvements over their initial configurations.

Model	Overall Accuracy (%)
VGG	83.5
ResNet	64.0
DenseNet	78.5

Model	Overall Accuracy (%)
VGG	86.8
ResNet	85.6
DenseNet	87.4

Table 5: Overall accuracy comparison between initial (left) and optimized (right) models

6.1 Classification Report

For each model, two sets of classification reports are provided separately: one corresponding to the initial parameters chosen and the other to the optimized configuration.

6.1.1 VGG Model

Class	Precision	Recall	F1-Score
COV+	0.89	0.87	0.88
COV-	0.85	0.79	0.82
CTRL	0.75	0.83	0.79

Class	Precision	Recall	F1-Score
COV+	0.90	0.90	0.90
COV-	0.87	0.86	0.86
CTRL	0.83	0.84	0.83

Table 6: Classification Report for the VGG Models (initial and best)

6.1.2 ResNet Model

Class	Precision	Recall	F1-Score
COV+	0.75	0.65	0.70
COV-	0.68	0.65	0.66
CTRL	0.58	0.67	0.62

Class	Precision	Recall	F1-Score
COV+	0.94	0.86	0.90
COV-	0.89	0.82	0.86
CTRL	0.77	0.91	0.83

Table 7: Classification Report for the ResNet Models (initial and best)

6.1.3 DenseNet Model

Class	Precision	Recall	F1-Score
COV+	0.95	0.68	0.79
COV-	0.76	0.82	0.79
CTRL	0.71	0.83	0.76

Class	Precision	Recall	F1-Score
COV+	0.89	0.93	0.91
COV-	0.94	0.80	0.86
CTRL	0.80	0.91	0.85

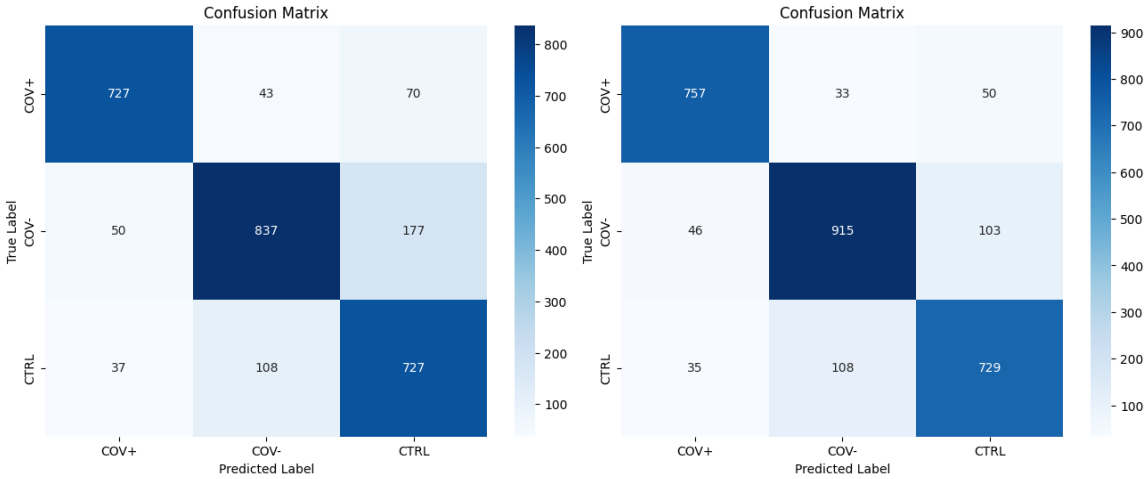
Table 8: Classification Report for the DenseNet Models (initial and best)

F1-Score provides a crucial measure of model performance by indicating prediction quality for each class. Through tuning phase, model performance improved across all architectures, with *DenseNet* emerging as the best architecture cause its superior **overall accuracy** (87.4%) and well-balanced **F1-scores** across all classes (0.91, 0.86, 0.85).

6.2 Confusion Matrices

The confusion matrices for each model are also presented separately for the initial and optimized configurations.

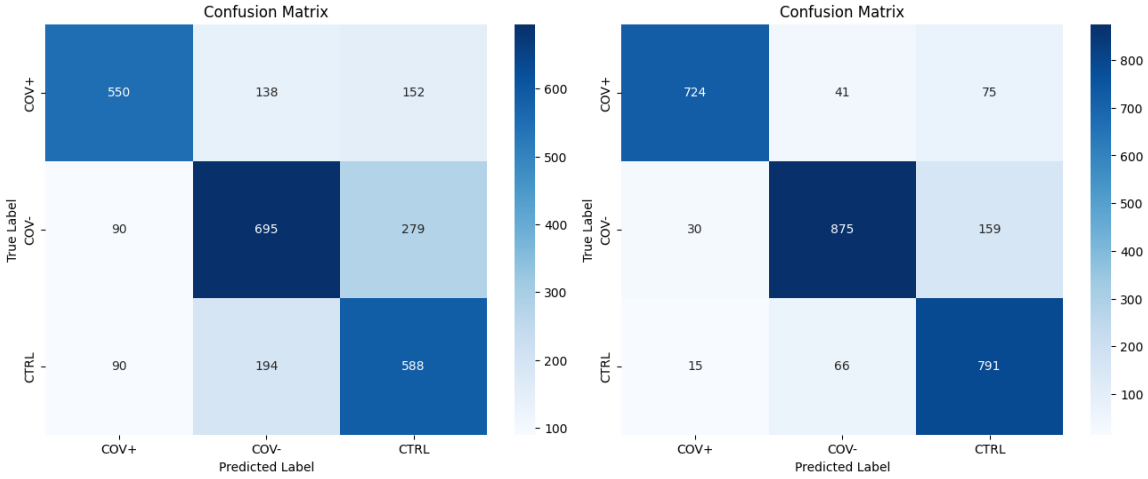
6.2.1 VGG Model



(a) Initial Parameters (b) Optimized Parameters

Figure 5: Confusion Matrices for the VGG Model

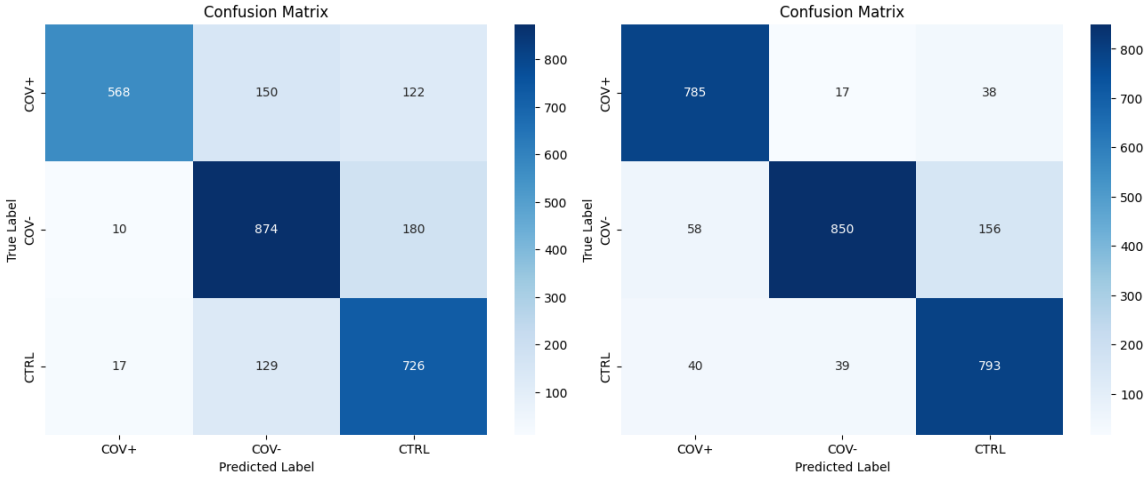
6.2.2 ResNet Model



(a) Initial Parameters (b) Optimized Parameters

Figure 6: Comparison of Confusion Matrices for the ResNet Model

6.2.3 DenseNet Model



(a) Initial Parameters (b) Optimized Parameters

Figure 7: Comparison of Confusion Matrices for the DenseNet Model

7 Conclusions and final considerations

This project explored the integration of **Raman spectroscopy** with **deep learning** techniques for **COVID-19 classification**. A dataset of Raman spectra was preprocessed through **outlier detection**, **background correction**, **despiking**, and **data augmentation** to enhance data quality and improve model performance.

Several deep learning architectures were implemented and evaluated, including **VGG, ResNet, DenseNet**. Each model underwent hyperparameter tuning, leading to significant performance improvements. The final results demonstrated that DenseNet achieved the highest accuracy (87.4%), followed by VGG (86.8%) and ResNet (85.6%).

A **Vision Transformer for 1D spectral data** (ViT1D) was implemented from scratch to explore the feasibility of Transformer-based architectures for Raman spectroscopy analysis and evaluate their **potential as an alternative** to convolutional networks.

Finally, these results highlight the potential of deep learning models in Raman-based medical diagnostics, providing a non-invasive and reliable approach to disease classification. Future research could focus on **expanding the dataset**, **improving spectral pre-processing methods**, and **developing more advanced Transformer based architectures** to improved obtained results.

References

[1] Dario Bertazioli et al. "An integrated computational pipeline for machine learning-driven diagnosis based on Raman spectra of saliva samples". In: *Computers in Biology and Medicine* 171 (2024), p. 108028. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2024.108028. URL: https://www.sciencedirect.com/science/article/pii/S0010482524001124.

[2] Esben Jannik Bjerrum, Mads Glahder, and Thomas Skov. "Data Augmentation of Spectral Data for Convolutional Neural Network (CNN) Based Deep Chemometrics". In: *CoRR* abs/1710.01927 (2017). arXiv: 1710.01927. URL: http://arxiv.org/abs/1710.01927.

[3] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[4] Rekha Gautam et al. "Review of multidimensional data processing approaches for Raman and infrared spectroscopy". In: *EPJ Techniques and Instrumentation* 2.8 (2015), pp. 1–38. DOI: 10.1140/epjti/s40485-015-0018-6.

[5] GeeksforGeeks. *Vision Transformer (ViT) Architecture*. https://www.geeksforgeeks.org/vision-transformer-vit-architecture/. 2023.

[6] Darren A. Whitaker and Kevin Hayes. "A simple algorithm for despiking Raman spectra". In: *Chemometrics and Intelligent Laboratory Systems* 179 (2018), pp. 82–84. ISSN: 0169-7439. DOI: https://doi.org/10.1016/j.chemolab.2018.06.009. URL: https://www.sciencedirect.com/science/article/pii/S0169743918301758.