COVID-19 detection using chest X-ray images: a multinomial logistic approach

Alexandru Stroilă

June 2023

Abstract

Chest X-ray imaging has long been utilized as an indispensable diagnostic tool for the detection of various respiratory diseases, including COVID-19. This paper aims at introducing a novel approach that employs multinomial logistic regression coupled with Stochastic Gradient Descent (SGD) for classifying chest X-ray images into COVID-19, Non-COVID, and Normal categories. By leveraging data augmentation techniques, we aimed to enhance the generalizability and robustness of the model, which ultimately achieved a respectable accuracy rate of 72%. This suggesting its potential utility in efficiently discriminating among the three classes, thereby facilitating accurate disease diagnosis.

While it is recognized that more advanced or recent machine learning models may achieve superior performance metrics, our research underscores that simplicity does not necessarily preclude efficiency. Indeed, our findings demonstrate the competency of less complex models, reinforcing their potential as valuable tools in the current medical imaging landscape. This perspective is particularly relevant for settings where computational resources may be limited or where interpretability is a priority. Future research could aim to further optimize such 'simpler' models or explore hybrid approaches that combine their benefits with those of more complex algorithms.

1 Introduction

SARS-CoV-2, the causative pathogen of coronavirus disease (COVID-19), leads to varying degrees of respiratory disease, with a subset of patients requiring intensive care. As of May 24, 2023, approximately 766 million confirmed COVID-19 cases have been reported globally, emphasizing the need for rapid and accurate diagnostic methods. Early detection of COVID-19 is critical in mitigating spread and enhancing treatment outcomes.

Multiple methodologies have been employed for COVID-19 patient categorization, leveraging wearable medical sensors and artificial neural networks to process data derived from questionnaires and physiological measurements (Hassantabar, et al., 2021).Furthermore, machine learning-based prediction algorithms have been developed to anticipate infection rates. Among diagnostic tools, Chest X-rays (CXR) and computed tomography (CT) scans remain common modalities for detecting lung pathologies such as pneumonia and tuberculosis, as well as COVID-19 (Apostolopoulos Mpesiana, 2020). Afshar, et al. (2020) pointed out the salient advantage of CXR is its operational simplicity, enabling rapid diagnosis with portable X-ray devices. Moreover, through artificial intelligence (AI) integration, it has been determined that CXRs possess a notable capacity for COVID-19 detection with less associated risk compared to CT scans. (Minaee, Kafieh, Sonka, Yazdani, Soufi, 2020).

Numerous studies have examined the implementation of deep-learning algorithms to detect COVID19 in X-ray images. Abbas et al. (2021), for instance, utilized a deep convolutional neural network (CNN) termed Decompose, Transfer, and Compose (DeTracC) to categorize COVID-19 CXR images. DeTracC's adeptness at managing image dataset anomalies resulted in an impressive accuracy of

93.1% (with a sensitivity of 100%) in distinguishing COVID-19 X-ray images from those of mild and severe acute respiratory syndrome cases. Khan, Shah, Bhat (2020) proposed CoroNet, another Deep Convolutional Neural Network model designed to automatically detect COVID-19 infection from CXR images. Based on the Xception architecture, CoroNet was trained end-to-end on a dataset comprising COVID-19, normal, and pneumonia CXR images, yielding an accuracy of 89.6% with notable precision and recall rates for COVID-19 cases at 93% and 98.2%, respectively. Additionally, Rahman, et al. (2021) investigated the performance of six different pre-trained CNNs (ResNet18, ResNet50, ResNet101, InceptionV3, DenseNet201, and ChexNet) and a shallow CNN model on both raw and segmented lung CXR images. The researchers further assessed the influence of various image enhancement techniques - including histogram equalization (HE), contrast-limited adaptive histogram equalization (CLAHE), image complement, gamma correction, and balanced contrast enhancement technique (BCET) - on COVID-19 detection. Nearly all the tested models delivered an accuracy close to 95%, illustrating the efficacy of these approaches in enhancing COVID-19 detection from CXR images.

2 Problem Framework

The research focuses on image classification, specifically of chest X-ray (CXR) images. Our objective is to categorize them into three classes: Normal, Non-COVID, and COVID-19. The distinctiveness of our approach resides in the pixel-by-pixel examination of the images, applying multinomial logistic regression as a simplified, yet effective machine learning model for this task. It extends traditional logistic regression, typically used for binary classification problems, to enable the handling of multiple categorical outcomes. In our case, we aim to classify each image into one of the three afore-mentioned categories, a task ideally suited for multinomial logistic regression.

However, this approach comes with several restrictions. Firstly, multinomial logistic regression operates under the assumption of independence of irrelevant alternatives (IIA), which stipulates that the odds of choosing one outcome relative to another do not depend on the availability or characteristics of additional outcomes. In the context of image classification, this might not always be true, as pixels in an image are often correlated with their immediate neighbors, challenging the IIA assumption.

Secondly, as multinomial logistic regression is a linear model, it may have difficulty accurately classifying more complex or subtle patterns within CXR images that could be better captured using non-linear models. The inherent complexity of CXR images, which may exhibit intricate textures and structural nuances indicative of diseases like COVID-19, can thus pose a challenge to our model. Additionally, multinomial logistic regression models could suffer from the issue of multicollinearity due to the high-dimensionality of our data. With each pixel treated as a data point, CXR images can potentially introduce thousands of variables into the model, increasing the risk of high intercorrelations that can destabilize the model and impact its predictive performance.

Lastly, training a multinomial logistic regression model using a pixel-by-pixel approach could prove computationally expensive, given the large volume of data involved. The model's performance, therefore, would depend heavily on the computational resources available and the efficiency of the chosen optimization algorithm. In our case, we employed Stochastic Gradient Descent (SGD), a well-known optimization algorithm suitable for handling large-scale datasets, to mitigate this issue.

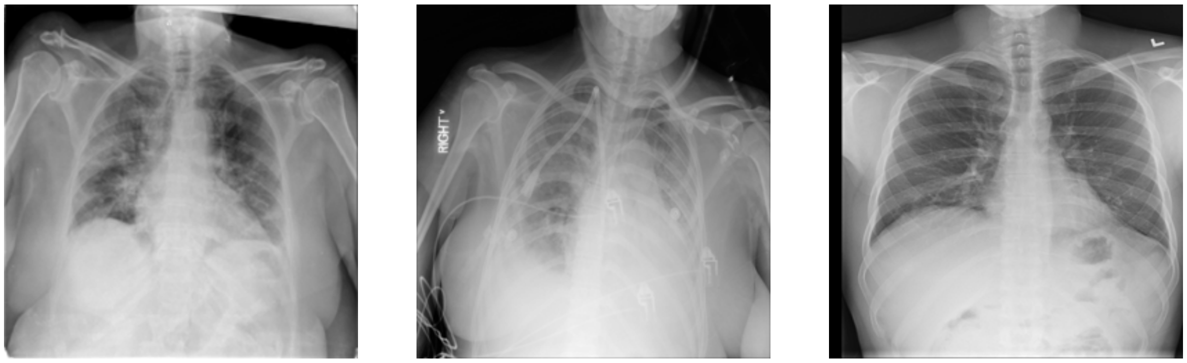
Notwithstanding these challenges, our work aims to demonstrate the potential applicability of simpler models like multinomial logistic regression in the realm of medical imaging, even when facing complex classification tasks. Furthermore, this investigation illuminates the inherent trade-offs between model complexity and interpretability, which may have significant implications, particularly in medical settings where explaining a model's decision could be as important as the decision itself.

3 Data & Methodology

3.1 Data Collection

The primary source of data for our project is obtained from a publicly accessible database provided by Kaggle, an open platform that hosts a plethora of datasets for research purposes. This specific dataset was compiled by a multinational team of researchers from Qatar University, Doha (Qatar) and the University of Dhaka, Bangladesh, and collaborators from Pakistan and Malaysia, all working closely with medical doctors.

The dataset consists of a collection of chest X-ray images **(Figure 1)**, documenting cases of COVID-19, normal lung images, and images showing Viral Pneumonia. The database was released in stages, allowing for progressive updates to the available data. Initially, the database contained 219 COVID-19, 1341 normal, and 1345 viral pneumonia chest X-ray (CXR) images. Subsequent updates expanded the COVID-19 class to 1200 CXR images, and the second update further expanded the database to 3616 COVID-19 positive cases. In addition to the COVID-19 cases, the second update also included 10,192 Normal, 6012 Lung Opacity (Non-COVID lung infection), and 1345 Viral Pneumonia images.



(a) CXR for Covid-19 (b) CXR for Non-COVID (c) CXR for Normal

Figure 1: Data samples

To further enrich our dataset and enhance the robustness of our model, we apply data augmentation techniques on the original images. This approach artificially expands the dataset by creating modified versions of the images, involving transformations like rotation, translation, rescaling, and brightness/-contrast adjustment (Chowdhury, et al., 2020) **(Figure 2)**. Consequently, our effective dataset size grows significantly, crossing the threshold of 100,000 images. The rationale behind data augmentation lies in its ability to simulate a larger variety of potential input scenarios for the model, thus improving the model's ability to generalize from the training data to unseen instances during prediction. This step is paramount in building a model that exhibits robust performance when confronted with real-world data.

3.2 Methodology

The methodology for this image classification task consists of several interconnected stages, including image preprocessing, model training, model evaluation, and data augmentation.

The starting point of our methodology involves preprocessing the CXR images to ensure consistent quality and dimensionality. These images are typically grayscale, meaning that each pixel can be represented as a single intensity value between 0 (black) and 255 (white). As a result, we conceptualize each pixel in an image as an individual data point (Rahman, et al., 2021). Consequently, a CXR image

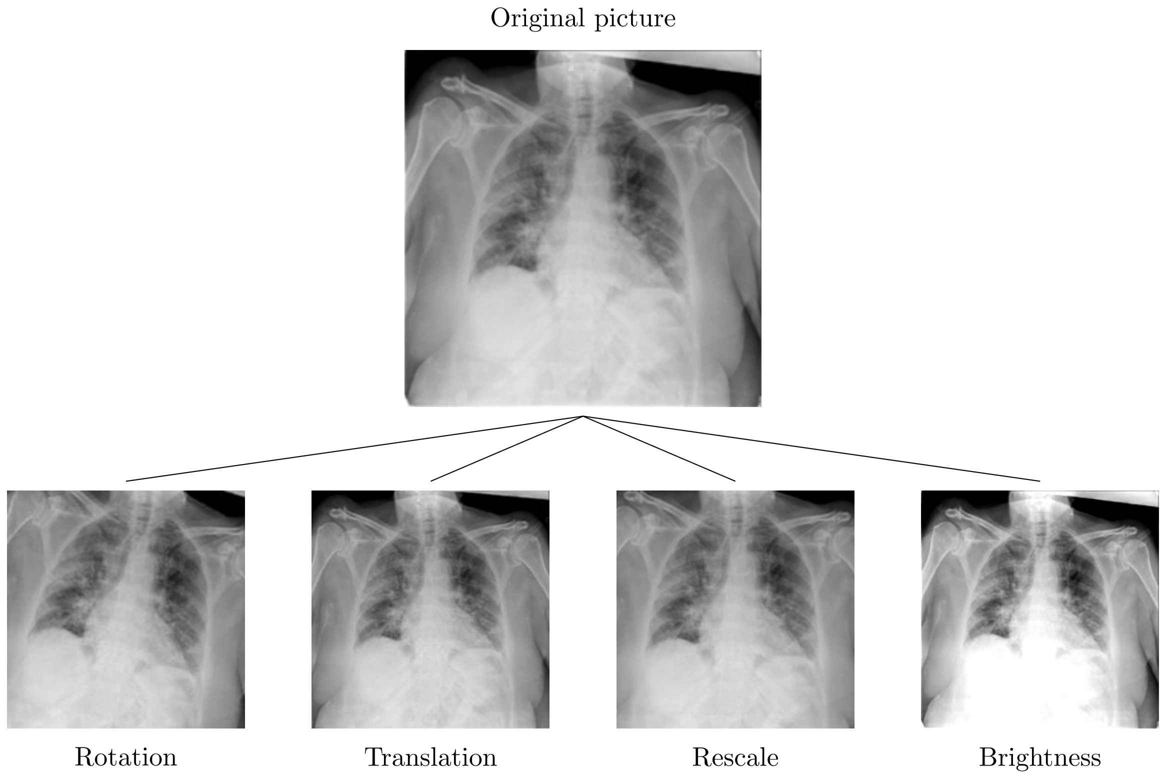


Figure 2: Example of data augmentation of a picture

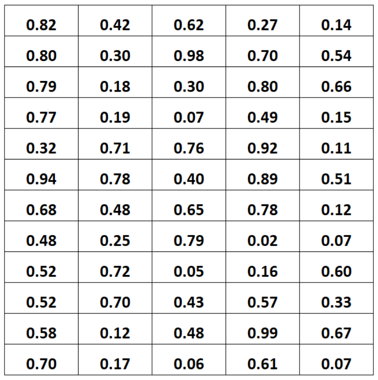
with a resolution of, for example, 256x256 pixels would offer a dataset of 65,536 individual points to be processed by our model.

Images are accessed from their respective category, resized to uniform dimensions, and their pixel values are normalized to fall within the range 0 to 1 **(Figure 3)**. This step ensures that images have consistent properties, eliminating potential discrepancies arising from diverse sources, formats, or image sizes.

In the following phase, we employ a generator function to yield batches of images and labels, effectively handling large datasets considering our memory limitations. This generator facilitates the iterative feeding of data into our model during the training and validation phases. This approach is particularly useful when dealing with datasets that cannot fit entirely into memory, allowing us to process the data in smaller manageable batches.

For training our image classification model, we choose the SGD classifier, or Stochastic Gradient Descent classifier. This optimization algorithm is well-suited for handling large-scale datasets, offering efficiency and the ability to update model parameters incrementally. Unlike traditional Gradient Descent, which computes the gradient on the entire dataset, SGD computes the gradient on a single random sample or mini-batch of samples. By incorporating stochasticity into the gradient estimation process, the SGD classifier introduces randomness that helps our model escape local optima and find better solutions. In each iteration, we adjust the model's parameters in the direction of the negative gradient, aiming to minimize the logistic loss function. This choice allows us to approach the image classification task as a multinomial logistic regression problem.

During training, our generator function feeds batches of images and their corresponding labels to our model iteratively. Each batch is used to partially fit the model, enabling it to update its parameters



0.82	0.42	0.62	0.27	0.14
0.80	0.30	0.98	0.70	0.54
0.79	0.18	0.30	0.80	0.66
0.77	0.19	0.07	0.49	0.15
0.32	0.71	0.76	0.92	0.11
0.94	0.78	0.40	0.89	0.51
0.68	0.48	0.65	0.78	0.12
0.48	0.25	0.79	0.02	0.07
0.52	0.72	0.05	0.16	0.60
0.52	0.70	0.43	0.57	0.33
0.58	0.12	0.48	0.99	0.67
0.70	0.17	0.06	0.61	0.07

Figure 3: Comparison of CXR Image and Normalized RGB Pixel Matrix

based on the gradient computed on that specific batch. This iterative approach facilitates incremental learning, where our model gradually improves its performance as it processes more batches of data. One notable aspect of our approach is the iterative testing of different weights for the classes. By experimenting with various class weights, we can identify the most optimal configuration that enhances our model's performance in classifying the chest X-ray images. This experimentation with class weights allows us to address the imbalanced nature of the dataset and improve our model's ability to accurately classify instances from all classes.

A validation set is used to test the classifier after each training iteration, measuring the F1 score. This score is a balanced measure of the model's precision and recall, providing a more comprehensive perspective on its performance than simple accuracy. The classifier that yields the best F1 score during the validation phase is chosen as the optimal model.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Next, the trained model is evaluated on a test dataset. True and predicted labels for the test images are collected to calculate the model's accuracy and produce a confusion matrix. This matrix provides a visual representation of the model's performance, showing the number of correct and incorrect predictions for each class. Further evaluation metrics, such as the Receiver Operating Characteristic (ROC) curve and Precision-Recall curve, are calculated and plotted for each class, providing a detailed view of the model's predictive capability.

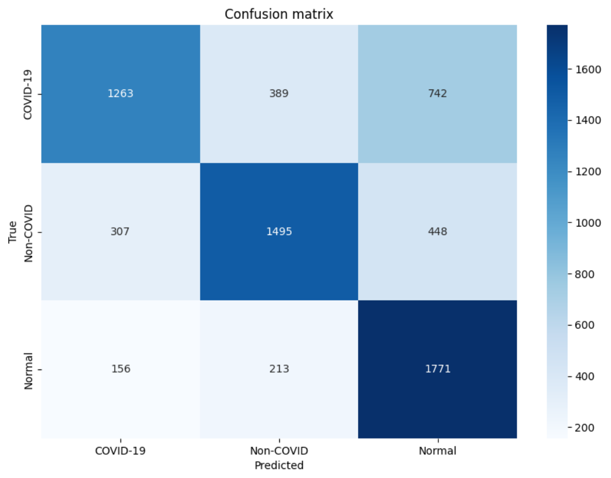
In addition to the stages previously described, a data augmentation stage is included to enhance the robustness of our model and potentially improve its performance. Data augmentation refers to the creation of synthetic data by applying transformations to the existing data, thus expanding the training dataset. It can help improve the generalization capabilities of the model, preventing overfitting by providing more diverse training examples. The data augmentation stage leverages the *Augmentor* library to apply a set of transformations to the training images. We performed the following four types of augmentations: rotation, translation (via random distortion), rescaling (zooming), and alterations of brightness and contrast.

This method creates an expanded and more diverse dataset, providing the model with more examples of potential variability in the images. This can lead to better feature extraction, thereby improving the model's predictive accuracy. Importantly, this augmentation strategy also helps mitigate the limitations of the multinomial logistic regression model by providing a richer, more complex training set. This may lead to the extraction of more relevant and diverse features, thereby enhancing the performance and robustness of the model. In conclusion, this methodology offers a comprehensive and robust approach to the image classification task. It includes several measures to ensure quality and performance, taking into consideration the caveats of the multinomial logistic regression model, and implementing strategies to mitigate them.

4 Results & Discussions

In this section, we present the results obtained from our experiments and provide a comprehensive discussion of the outcomes. We conducted our evaluation using two different datasets: the original dataset (containing around 21,000 images in total) and an augmented dataset that includes the original images as well as additional artificially generated images.

First, let's discuss the results obtained from the original dataset. The multinomial logistic regression model trained on this dataset demonstrated promising performance. The model achieved an F-1 score of 65% on the test set **(Figure 4 and Figure 5)**, showcasing its ability to effectively classify chest X-ray images into the three target categories: COVID-19, Non-COVID, and Normal. Furthermore, the confusion matrix provided insights into the model's performance for each class, illustrating the number of correct and incorrect predictions.



Class	Precision	Recall	F1-Score	Support
COVID-19	0.73	0.53	0.61	2394
Non-COVID	0.71	0.66	0.69	2250
Normal	0.60	0.83	0.69	2140
Accuracy	-	-	0.67	6784
Macro Avg	0.68	0.67	0.67	6784
Weighted Avg	0.68	0.67	0.66	6784

Figure 4: Confusion matrix and classification report for the original dataset

However, it is important to highlight that the augmentation of the dataset significantly improved the model's performance. By incorporating the augmented images, which expanded the dataset to over 100,000 images, we observed a substantial enhancement in the model's predictive capabilities. The F1-score on the test set increased to 72%, indicating a notable improvement compared to the results obtained from the original dataset alone. This outcome demonstrates the benefits of data augmentation in effectively capturing the diverse variations and complexities present in real-world chest X-ray images.

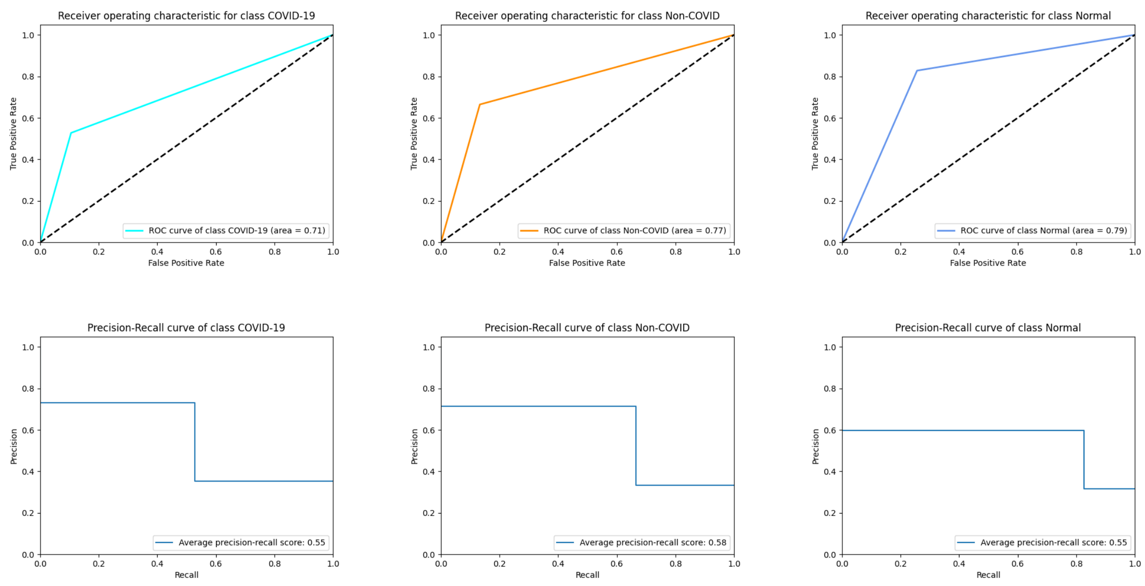


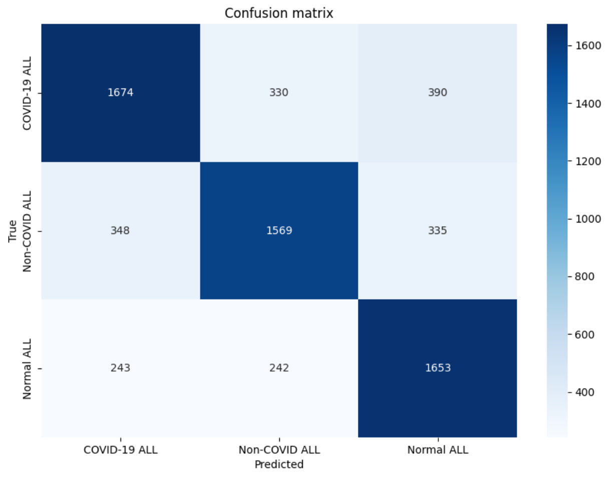
Figure 5: ROC and Precision-Recall curve for each class for the original dataset

The improved performance achieved with the augmented dataset can be attributed to the ability of the model to learn from a more extensive and diverse set of examples. The augmented images provided a wider range of variations in terms of rotation, translation, rescaling, brightness, and contrast, enabling the model to extract more robust and discriminative features. As a result, the model became more adept at generalizing its knowledge to unseen instances, resulting in better accuracy and predictive capabilities(**Figure 6**).

The classification reports for both datasets further highlight the differences in performance. For the first model, the precision, recall, and F1-scores for each class were relatively lower compared to second model. This suggests that the model trained solely on the original dataset struggled to achieve balanced performance across all classes. However, in the second model, we can observe higher precision, recall, and F1-scores for each class, indicating improved classification accuracy and effectiveness.

The ROC curves and Precision-Recall curves plotted for each class **(Figure 7)**, further validate the superiority of the augmented dataset. The curves demonstrate a more favorable trade-off between true positive rate and false positive rate, as well as between precision and recall, indicating an improved ability to discriminate between different classes. This finding suggests that the augmented dataset allows the model to better differentiate COVID-19 cases from Non-COVID cases and Normal cases.

In summary, our experiments revealed that the incorporation of augmented images significantly improved the performance of the multinomial logistic regression model for chest X-ray image classification. The augmented dataset, facilitated more accurate and robust predictions compared to the original dataset alone. The results emphasize the importance of data augmentation in enhancing the model's ability to handle the complexities and variations present in real-world medical images. This finding highlights the potential of utilizing larger, diverse datasets in combination with appropriate data augmentation techniques to achieve more accurate and reliable classification outcomes in the field of medical imaging.



Class	Precision	Recall	F1-Score	Support
COVID-19	0.74	0.70	0.72	2394
Non-COVID	0.73	0.70	0.71	2252
Normal	0.70	0.77	0.73	2138
Accuracy	-	-	0.72	6784
Macro Avg	0.72	0.72	0.72	6784
Weighted Avg	0.72	0.72	0.72	6784

Figure 6: Confusion matrix and classification report for the original dataset

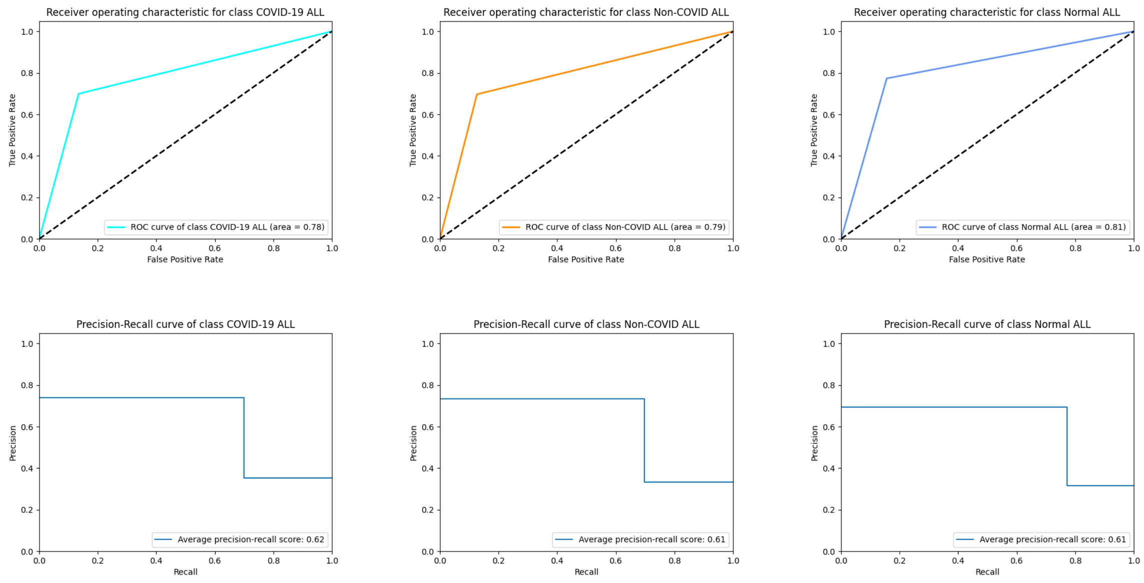


Figure 7: ROC and Precision-Recall curve for each class for the augmented dataset

5 Conclusions

In this study, we have proposed a method utilizing multinomial logistic regression with Stochastic Gradient Descent (SGD) for the classification of chest X-ray images into COVID-19, Non-COVID, and Normal categories. The trained model achieved good discrimination among the three classes, with an accuracy of 72%. Our findings highlight the potential of simpler models, such as multinomial logistic regression, in effectively addressing image classification problems in the context of respiratory disease diagnosis.

While more advanced and complex AI models, such as convolutional neural networks (CNNs), have demonstrated superior performance in various image classification tasks, our study emphasizes that even simpler models can achieve decent results. This is particularly significant considering the computational complexity and resource requirements associated with more advanced models. The multinomial logistic regression model offers an efficient and accessible approach for accurate triage and diagnosis in respiratory diseases, including COVID-19.

Moreover, we acknowledge that data augmentation techniques played a crucial role in improving the model's performance. By artificially expanding the dataset through transformations such as rotation, translation, rescaling, and brightness/contrast adjustments, we enhanced the model's ability to handle diverse variations and complexities in real-world chest X-ray images. Data augmentation proved to be an effective strategy in improving the model's generalization capabilities and reducing the risk of overfitting.

In conclusion, our study demonstrates that the multinomial logistic regression model, coupled with data augmentation techniques, can be a valuable tool for efficient and accurate triage in respiratory disease diagnosis, including COVID-19. While more advanced AI models may outperform simpler models, our research highlights that even with the multinomial logistic regression approach, decent performance can be achieved. This implies that simpler models can still be practical options, particularly in scenarios where computational resources or model complexity constraints are present. Moving forward, further investigations can explore the combination of more sophisticated models with data augmentation techniques to improve the accuracy and robustness of respiratory disease classification systems.

6 References

1. Abbas, A. A. (2021). Classification of COVID-19 in chest X-ray images using DeTraC deep convolutional neural network. Appl Intell 51, 854–864.

2. Apostolopoulos, I., Mpesiana, T. (2020). Covid-19: automatic detection from x-ray images utilizing transfer learning with convolutional neural networks. *Phys. Eng. Sci. Med.*

3. Afshar, P., Heidarian, S., Naderkhani, F., Oikonomou, A., Plataniotis, K. N., Mohammadi, A. (2020). Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *Pattern Recognition Letters*, 638-643.

4. Chowdhury, M. E., Rahman, T., Khandakar, A., Mazhar, R., Kadir, M. A., Mahbub, Z. B., . . . Islam, M. T. (2020). Can AI Help in Screening Viral and COVID-19 Pneumonia? *Ieee Access.*

5. Hassantabar, S., Stefano, N., Ghanakota, V., Ferrari, A., Nicola, G. N., Bruno, R., ... Jha, N. K. (2021). CovidDeep: SARS-CoV-2/COVID-19 test based on wearable medical sensors and efficient neural networks. *IEEE Transactions on Consumer Electronics*, 244-256.

6. Khan, s. I., Shah, J. L., Bhat, M. M. (2020). CoroNet: A deep neural network for detection and diagnosis of COVID-19 from chest x-ray images, Computer Methods and Programs in Biomedicine. Computer methods and programs in biomedicine.

7. Minaee, S., Kafieh, R., Sonka, M., Yazdani, S., Soufi, G. J. (2020). Deep-COVID: Predicting COVID-19 from chest X-ray images using deep transfer learning. *Medical image analysis.*

8. Rahman, T., Khandakar, A., Qiblawey, Y., Tahir, A., Kiranyaz, S., Kashem, S. B., . . . Chowdhury, M. E. (2021). Exploring the effect of image enhancement techniques on COVID-19 detection using chest X-ray images. Computers in biology and medicine.

***Kaggle. COVID-19 Radiography Database. Available at https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database

7 Appendix: Code Snippets

7.1 main.py

```python
import os
from itertools import cycle
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from tqdm import tqdm

from image_generator import image_generator
from train import train_classifier
from sklearn.metrics import (confusion_matrix, roc_curve, auc,
                             accuracy_score, precision_recall_curve,
                             average_precision_score, classification_report)
from sklearn.preprocessing import label_binarize

# Your directory
TRAIN_DATA_FOLDER = r"Lung␣Segmentation␣Data\Train"
TEST_DATA_FOLDER = r"Lung␣Segmentation␣Data\Test"

# Calculate the total number of images
total_samples = sum([len(files) for r, d, files in os.walk(TRAIN_DATA_FOLDER)
    ])

# Instantiate a generator
batch_size = 64
gen = image_generator(TRAIN_DATA_FOLDER, batch_size)

# Train the classifier
clf, label_to_class = train_classifier(gen, total_samples, batch_size)

# Instantiate a generator for test data
test_gen = image_generator(TEST_DATA_FOLDER, batch_size)

# Calculate the total number of images in the test set
total_test_samples = sum([len(files) for r, d, files in os.walk(
    TEST_DATA_FOLDER)])

# Define steps for the test set
test_steps = total_test_samples // batch_size

# Initialize an empty array for the true and predicted labels
y_true = []
y_pred = []

# Loop over each batch from the test generator
for i in tqdm(range(test_steps), desc="Evaluating␣test␣set"):
    (batch_x, batch_y), _ = next(test_gen)
    y_true.extend(batch_y)
    predictions = clf.predict(batch_x)
    y_pred.extend(predictions)

# Calculate the accuracy
accuracy = accuracy_score(y_true, y_pred)
print("Accuracy␣on␣the␣test␣set:␣", accuracy)
```

```

# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=range(len(label_to_class)))
class_names = [label_to_class[i] for i in range(len(label_to_class))]

plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=class_names,
    yticklabels=class_names)
plt.title('Confusion matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# Binarize the output
y_true_bin = label_binarize(y_true, classes=range(len(label_to_class)))
y_pred_bin = label_binarize(y_pred, classes=range(len(label_to_class)))

n_classes = y_true_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_pred_bin[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Find the index of the 'COVID-19' class
covid_class_idx = None
for i, class_name in label_to_class.items():
    if class_name == 'COVID-19 3505':
        covid_class_idx = i
        break

# Plot ROC curves for each class
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.figure()  # Start a new figure for this plot
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(label_to_class[i], roc_auc[i]))
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic for class {0}'.format(
        label_to_class[i]))
    plt.legend(loc="lower right")
    plt.show()

# Compute Precision-Recall curve and average precision for each class
for i in range(n_classes):
    precision, recall, _ = precision_recall_curve(y_true_bin[:, i], y_pred_bin
        [:, i])
    average_precision = average_precision_score(y_true_bin[:, i], y_pred_bin
        [:, i])

    # Plot Precision-Recall curve for each class
    plt.figure()
    plt.step(recall, precision, where='post', label='Average precision-recall 
        score: {0:0.2f}'.format(average_precision))
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall curve of class {0}'.format(label_to_class[i]))
    plt.legend(loc="lower right")
    plt.show()

# Error analysis
print("\nClassification Report:\n")
print(classification_report(y_true, y_pred, target_names=class_names))
```

7.2 train.py

```
import os
from sklearn.linear_model import SGDClassifier
import numpy as np
from sklearn.metrics import f1_score
from image_generator import image_generator
from tqdm import tqdm

VALIDATION_DATA_FOLDER = r"Lung Segmentation Data\Val"

def calculate_f1_score(clf, val_gen, total_val_samples, batch_size):
    steps = total_val_samples // batch_size

    y_true = []
    y_pred = []

    for i in range(steps):
        (batch_x, batch_y), _ = next(val_gen)
        y_true.extend(batch_y)
        predictions = clf.predict(batch_x)
        y_pred.extend(predictions)

    f1 = f1_score(y_true, y_pred, average='weighted')
    return f1

def train_classifier(gen, total_samples, batch_size):
    # Define and compile your model
    class_weights = [
        {0: 1.8, 1: 1, 2: 1.},
        {0: 2, 1: 1, 2: 1.},
        {0: 2.5, 1: 1, 2: 1.},
        {0: 3, 1: 1, 2: 1.},
        # Add more class weights here if you want to try them
    ]

    best_f1_score = 0
    best_clf = None
    best_weights = None

    # Define steps_per_epoch
    steps_per_epoch = total_samples // batch_size

    # Calculate the total number of images in the validation set
    total_val_samples = sum([len(files) for r, d, files in os.walk(
        VALIDATION_DATA_FOLDER)])

    # Instantiate a generator for validation data
    val_gen = image_generator(VALIDATION_DATA_FOLDER, batch_size)

    for weights in class_weights:
        clf = SGDClassifier(loss='log_loss', class_weight=weights, n_jobs=-1)

        # Loop over each batch from the generator
        for i in tqdm(range(steps_per_epoch), desc="Training␣progress"):
            (batch_x, batch_y), label_to_class = next(gen)
            clf.partial_fit(batch_x, batch_y, classes=np.unique(batch_y))

        # Test the classifier here and calculate the F1 score
        f1_score = calculate_f1_score(clf, val_gen, total_val_samples,
            batch_size)

        if f1_score > best_f1_score:
            best_f1_score = f1_score
            best_clf = clf
            best_weights = weights

    print("Best␣weights:␣", best_weights)
    print("Best␣F1␣score:␣", best_f1_score)

    return best_clf, label_to_class
```

7.3 image_generator.py

```
import os
import glob
from PIL import Image
import numpy as np
from sklearn import preprocessing

# Your directory
TRAIN_DATA_FOLDER = r"Lung␣Segmentation␣Data\Train"

allowed_extensions = [".jpeg", ".png", ".jpg"]

def image_generator(input_dir, batch_size):
    """
    A generator that yields batches of images and labels.
    """
    image_files = []
    labels = []

    for class_folder_name in os.listdir(input_dir):
        class_folder_path = os.path.join(input_dir, class_folder_name)
        for image_path in glob.glob(os.path.join(class_folder_path, "*")):
            if os.path.splitext(image_path)[1] in allowed_extensions:
                image_files.append(image_path)
                labels.append(class_folder_name)

    # Perform encoding on labels
    le = preprocessing.LabelEncoder()
    encoded_labels = le.fit_transform(labels)

    # Create a mapping of encoded labels to class names
    label_to_class = dict(zip(le.transform(le.classes_), le.classes_))

    while True:
        # Shuffle the indices of the images
        indices = np.arange(len(image_files))
        np.random.shuffle(indices)

        for start_idx in range(0, len(indices) - batch_size + 1, batch_size):
            excerpt = indices[start_idx:start_idx + batch_size]

            batch_input = []
            batch_output = []

            for index in excerpt:
                input = Image.open(image_files[index]).convert(’L’)
                if input.size != (256, 256):
                    input = input.resize((256, 256))
                # Normalize the pixel values (scale them between 0 and 1)
                input = np.array(input.getdata()) / 255.0
                output = encoded_labels[index]

                batch_input.append(input)
                batch_output.append(output)

            batch_x = np.array(batch_input)
            batch_y = np.array(batch_output)

            yield (batch_x, batch_y), label_to_class
```

7.4 augment.py

```
import Augmentor
import os

# Your directory
TRAIN_DATA_FOLDER = r"Lung␣Segmentation␣Data\Train"

allowed_extensions = [".jpeg", ".png", ".jpg"]

# List of augmentations
augmentations = ["rotate", "translate", "rescale", "brightness_contrast"]

def augment_images(input_dir, output_dir, augment_type):
    """
    This function takes an input directory of images, applies specified
        augmentation, and saves the images in the output directory.
    """

    # Initialize the pipeline (don't load standard augmentation operations)
    p = Augmentor.Pipeline(input_dir, output_dir, save_format="JPEG")

    if augment_type == "rotate":
        p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    elif augment_type == "translate":
        p.random_distortion(probability=0.5, grid_width=2, grid_height=2,
            magnitude=2)
    elif augment_type == "rescale":
        p.zoom(probability=0.5, min_factor=1.1, max_factor=1.3)
    elif augment_type == "brightness_contrast":
        p.random_brightness(probability=0.5, min_factor=0.7, max_factor=1.3)
        p.random_contrast(probability=0.5, min_factor=0.8, max_factor=1.2)

    p.sample(len(p.augmentor_images))

# Process all classes
for class_folder_name in os.listdir(TRAIN_DATA_FOLDER):
    class_folder_path = os.path.join(TRAIN_DATA_FOLDER, class_folder_name)

    for augment in augmentations:
        output_dir = os.path.join(TRAIN_DATA_FOLDER, f"{class_folder_name}_{
            augment}")

        # Create output directory if it doesn't exist
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        # Augment the images
        augment_images(class_folder_path, output_dir, augment)
```