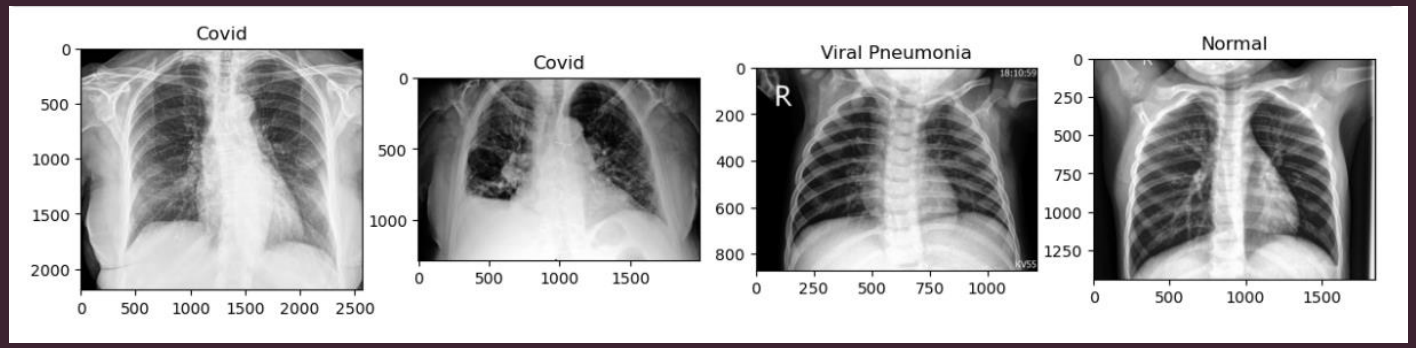
Covid-19 chest Xray classification

Using machine learning and Deep Learning Methods



Submitters:

Israel Gitler 208580076

Ohad Wolfman 316552496

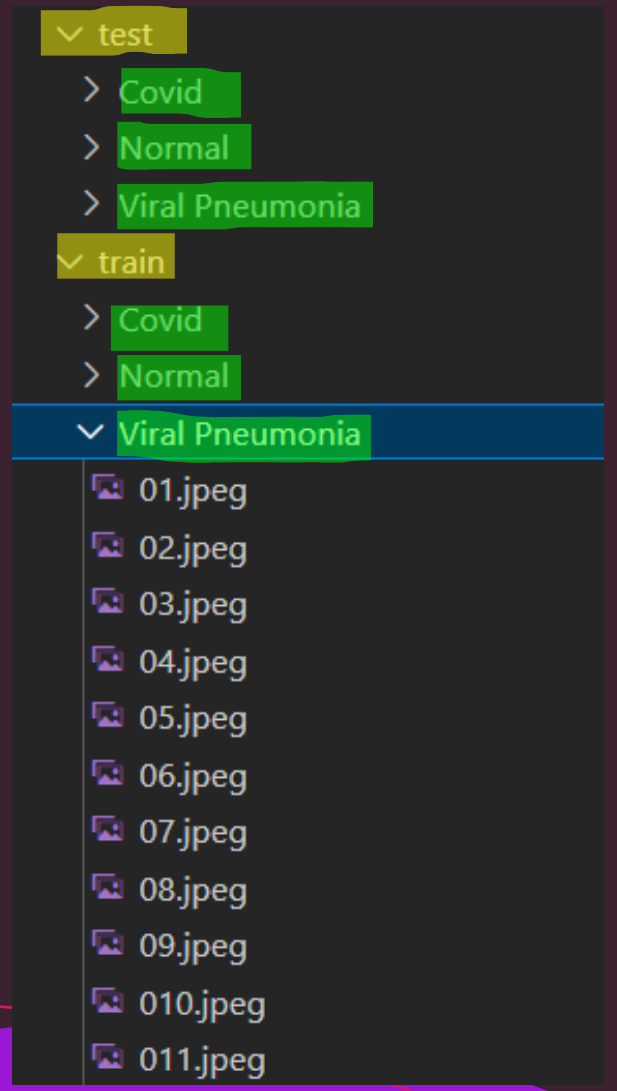
INTRODUCTION

- This project focuses on classifying chest X-ray images into three categories:
  1. COVID-19
  2. viral pneumonia
  3. healthy people
- Chest X-ray classification plays a vital role in diagnosing respiratory diseases, especially during pandemics like COVID-19.
- This project aims to develop a reliable CNN model that will overcome ML model scores and will be capable of accurately categorizing chest X-ray images into relevant classes.

GETTING KNOW THE DATA

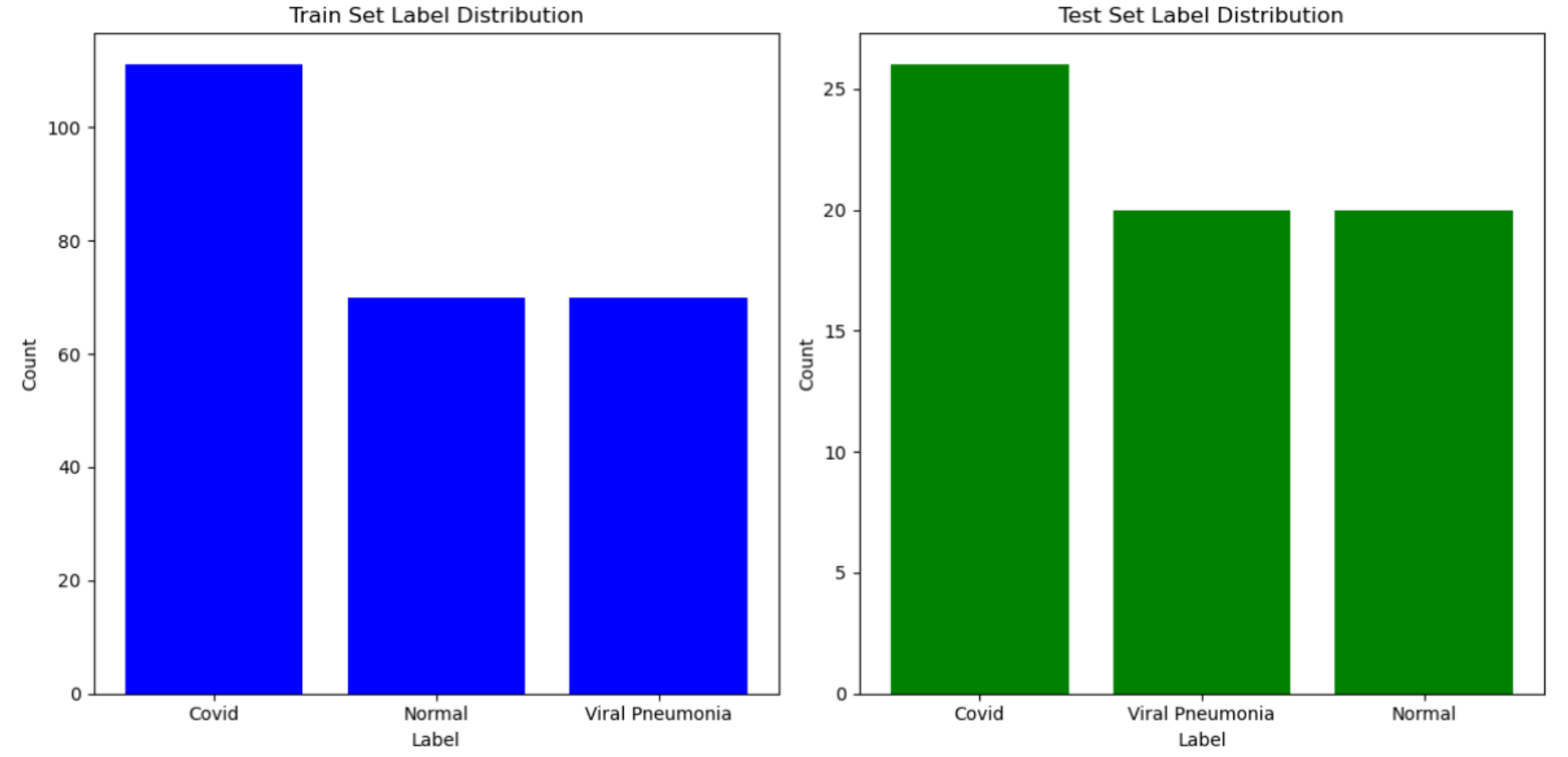
- This Data is organized in 2 folders which are separated to its label into different folders for each cluster.
- We extracted the images by iterating the folders and append it to lists.

```
def preprocess_data():
    # Load and preprocess train and test data
    train_dir = 'train'
    test_dir = 'test'

    train_filepaths = []
    train_labels = []
    for label in os.listdir(train_dir):
        label_dir = os.path.join(train_dir, label)
        for filename in os.listdir(label_dir):
            filepath = os.path.join(label_dir, filename)
            train_filepaths.append(filepath)
            train_labels.append(label)

    test_filepaths = []
    test_labels = []
    for label in os.listdir(test_dir):
        label_dir = os.path.join(test_dir, label)
        for filename in os.listdir(label_dir):
            filepath = os.path.join(label_dir, filename)
            test_filepaths.append(filepath)
            test_labels.append(label)
```



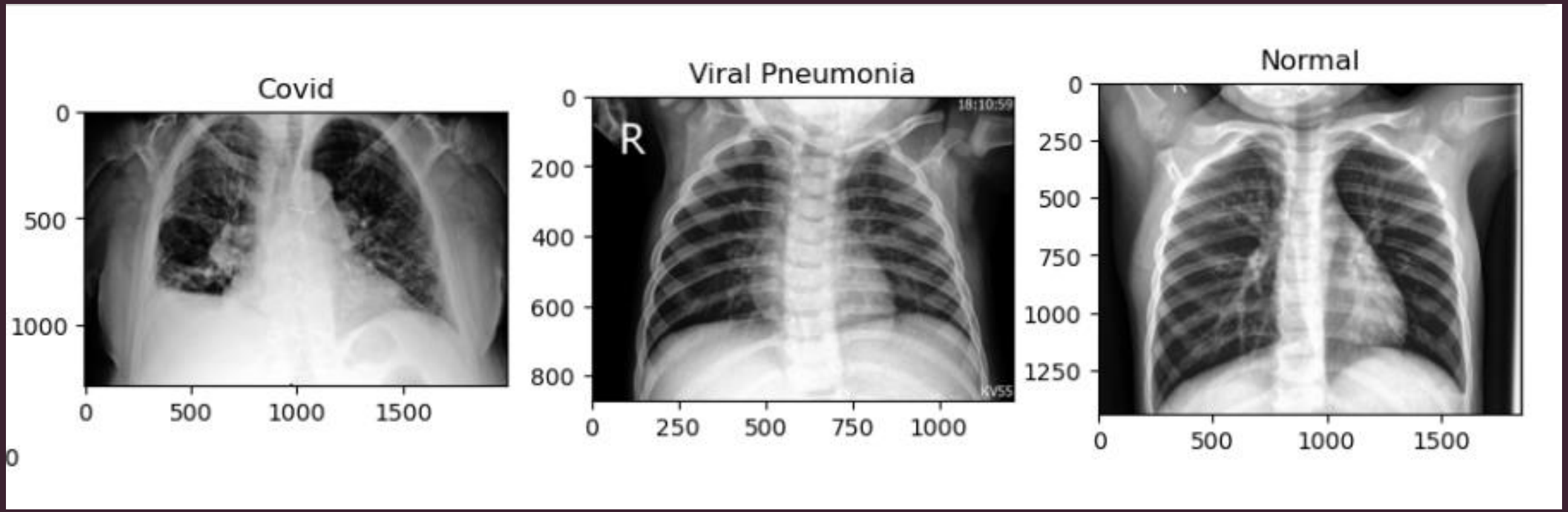
Getting know the data

- Although The data is a bit biased to the “Covid” cluster, we have enough data such that the model will be able to learn all the labels well



GETTING KNOW THE DATA

- In addition we noticed the sizes of the images are different



PRE-PROCESSING

- During the loading of the images, we implemented a preprocessing function to convert all the images to the same size and the same scale

```python
def preprocess_image(image_path, target_size=(256, 256)):
    img = cv2.imread(image_path)
    img_resized = cv2.resize(img, target_size)
    img_normalized = img_resized.astype("float32") / 255.0
    return img_normalized
```

- Afterwards we separated them into Numpy correspondence arrays

```python
x_train = np.array([preprocess_image(filepath) for filepath in train_filepaths])
y_train = np.array(train_labels)
x_test = np.array([preprocess_image(filepath) for filepath in test_filepaths])
y_test = np.array(test_labels)

# Convert labels to numerical values
label_to_index = {label: i for i, label in enumerate(np.unique(y_train))}
y_train = np.array([label_to_index[label] for label in y_train])
y_test = np.array([label_to_index[label] for label in y_test])

return x_train, y_train, x_test, y_test, train_filepaths, train_labels, test_filepaths, test_labels
```

PRE-PROCESSING

- And made sure the training data would shuffle

```
x_train, y_train, x_test, y_test, train_filepaths, train_labels, test_filepaths, test_labels = preprocess_data()

# Shuffle indices
shuffled_indices = np.random.permutation(len(x_train))

# Shuffle x_train and y_train accordingly
x_train = x_train[shuffled_indices]
y_train = y_train[shuffled_indices]
```

- Creating a Pandas Data Frames – (251 samples in the training set, and 66 in test set)

	filepath	label
0	test\Covid\0100.jpeg	Covid
1	test\Covid\0102.jpeg	Covid
2	test\Covid\0105.png	Covid
3	test\Covid\0106.jpeg	Covid
4	test\Covid\0108.jpeg	Covid

	filepath	label
0	train\Covid\01.jpeg	Covid
1	train\Covid\010.png	Covid
2	train\Covid\012.jpeg	Covid
3	train\Covid\015.jpg	Covid
4	train\Covid\019.png	Covid

- We also split the training set into a validation set

```
Num of samples in train set:  200
Num of samples in validation set:  51
Num of samples in test set:  66
```

SOFTMAX MODEL

- We used Softmax which is commonly preferred when dealing with multi-class classification problems.
- Softmax assumes your classes are mutually exclusive. This means a data point can only belong to one class at a time.
- In addition, Softmax outputs a probability distribution for each class. The sum of these probabilities for all classes will always be 1. This makes it ideal for scenarios where you want the model to predict the most likely class along with its confidence level

SOFTMAX MODEL

- We developed our SoftmaxModel class using TensorFlow and Keras. After experimenting with various configurations, we achieved optimal performance with a learning rate of 0.001 and the Adam optimizer

```
Epoch 48/50
7/7 [==============================] - 0s 22ms/step - loss: 3.5642e-07 - accuracy: 1.0000 - val_loss: 0.9532 - val_accura
cy: 0.9804
Epoch 49/50
7/7 [==============================] - 0s 23ms/step - loss: 3.5225e-07 - accuracy: 1.0000 - val_loss: 0.9532 - val_accura
cy: 0.9804
Epoch 50/50
7/7 [==============================] - 0s 22ms/step - loss: 3.5106e-07 - accuracy: 1.0000 - val_loss: 0.9532 - val_accura
cy: 0.9804
3/3 [==============================] - 0s 5ms/step - loss: 2.5155 - accuracy: 0.8485
Test Accuracy: 0.8484848737716675
```

'ADAM' OPTIMIZER

- Adam (Adaptive Moment Estimation) helps minimize the loss function by iteratively adjusting the learning rate for each parameter in the model based on past gradients by fast convergence (if the loss is high – the adjustment in the learning rate will be much more significant than if the loss is low)

SIMPLE NN MODEL

- Using TensorFlow and Keras, we built a simple Neural network of Sequential model, with ReLU activation function in hidden layers and softmax activation function in output layer.

```
model = Sequential([
    Flatten(input_shape=(256, 256, 3)),  # Flatten input images
    Dense(256, activation='relu'),  # First dense layer with ReLU activation
    Dense(128, activation='relu'),  # Second dense layer with ReLU activation
    Dense(64, activation='relu'),  # third dense layer with ReLU activation
    Dense(num_classes, activation='softmax')  # Output layer with softmax activation
])
```

- We compiled the model with the previous configurations, and fit the model with 50 epochs, with batch size of 32

```
Epoch 48/50
7/7 [==============================] - 3s 380ms/step - loss: 1.6700e-06 - accuracy: 1.0000 - val_loss: 0.9931 - val_accur
acy: 0.9608
Epoch 49/50
7/7 [==============================] - 3s 377ms/step - loss: 1.5735e-06 - accuracy: 1.0000 - val_loss: 0.9872 - val_accur
acy: 0.9608
Epoch 50/50
7/7 [==============================] - 3s 376ms/step - loss: 1.4948e-06 - accuracy: 1.0000 - val_loss: 0.9843 - val_accur
acy: 0.9608
3/3 [==============================] - 0s 22ms/step - loss: 1.4562 - accuracy: 0.8636
Test Accuracy: 0.8636363744735718
```

SIMPLE NN MODEL

- The current model's performance might be suffering because it's a fully connected neural network with several dense layers. This architecture might be too complex for this specific task, especially if the images contain a lot of irrelevant pixels.

CNN MODEL

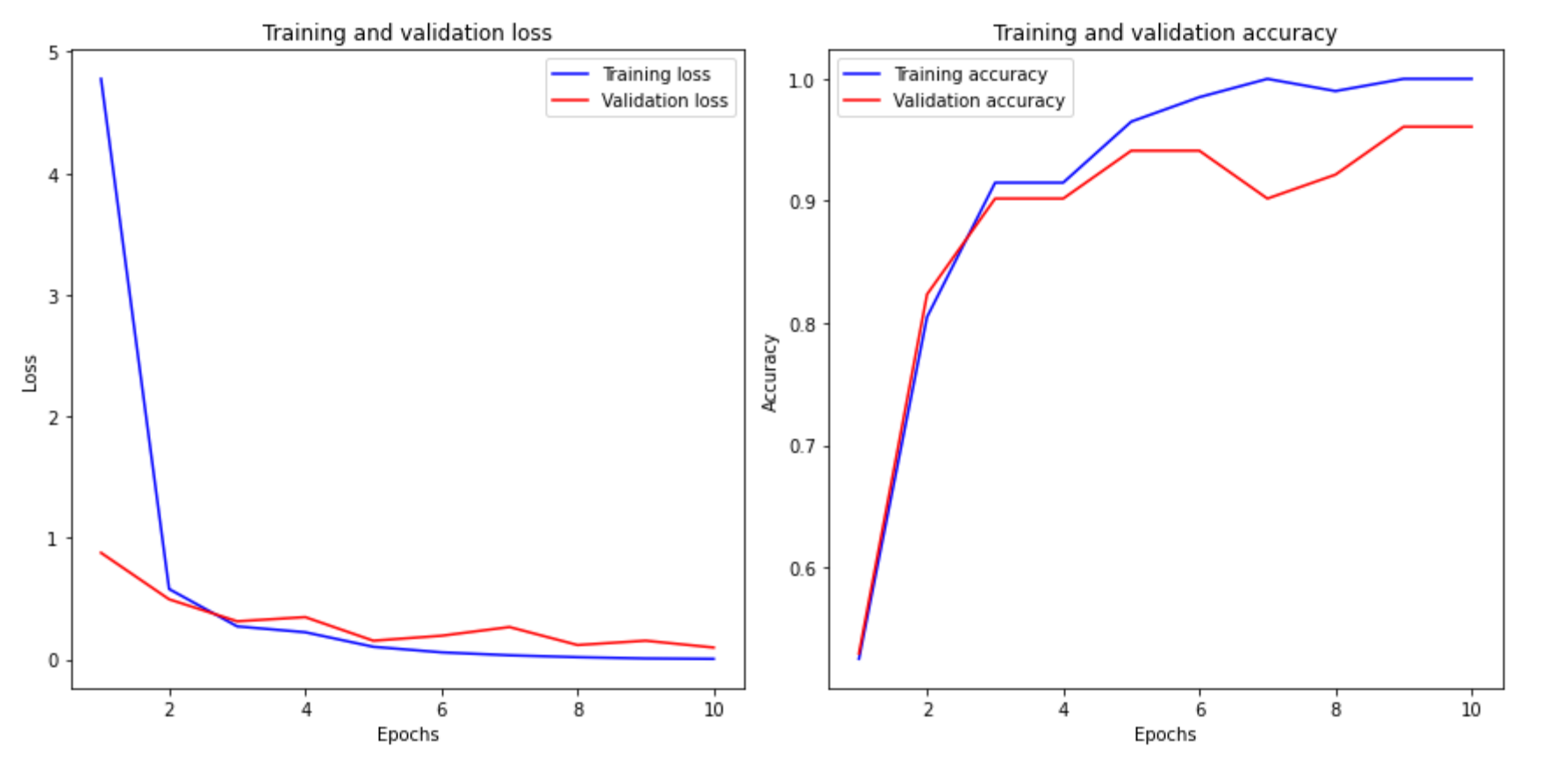
- Using a convolutional neural network, we achieved a much better score
- We convolved the data and reduced it dimension by MaxPooling2D and the dropout method
- This process required a few attempts and adjustments with the parameters.
- We compiled the model with almost same previous configuration (Adam optimizer, loss - categorical_crossentropy, and accuracy metric), changed only the batch size to 16 and epochs to 10.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 254, 254, 32)      896

 max_pooling2d_2 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 dropout_2 (Dropout)         (None, 127, 127, 32)      0

 conv2d_3 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_3 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 dropout_3 (Dropout)         (None, 62, 62, 64)        0

 flatten_1 (Flatten)         (None, 246016)            0

 dense_2 (Dense)             (None, 128)               31490176

 dense_3 (Dense)             (None, 64)                8256

 dense_4 (Dense)             (None, 3)                 195

=================================================================
Total params: 31518019 (120.23 MB)
Trainable params: 31518019 (120.23 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 10/10
13/13 [==============================] - 10s 809ms/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 0.0988 - val_accuracy:
0.9608
3/3 [==============================] - 1s 181ms/step - loss: 0.1515 - accuracy: 0.9697
Test Accuracy: 0.9696969985961914
```

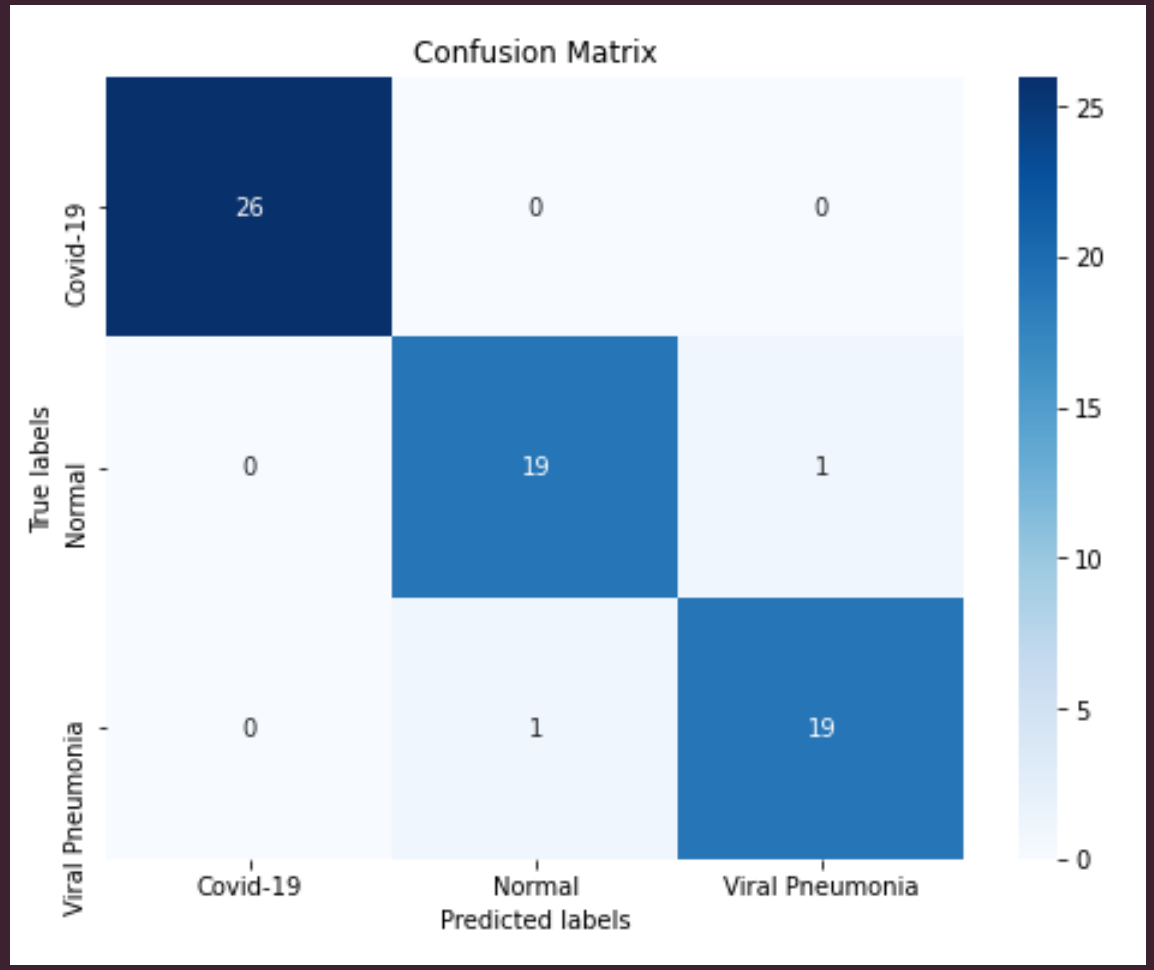
CNN MODEL

- Plotting the accuracy score relative to the loss score
- We would expect the loss and accuracy of the training and the validation sets to go together, and as we can see - it is.



CONFUSION MATRIX

- Following this confusion matrix we can see that the model predicted correctly 64 from 66 samples, with the following mistakes



METRICS COMPARING

- There are several considerations in the direction of the error (e.g. in cases of false negative/false negative)

- We wanted to see the results of the model also with the F1-score or Precision indices, which would be the recommended indices since it is important to identify positive cases accurately To be able to provide the necessary medical treatment.

METRICS COMPARING

- As you can see when it comes to a COVID-19 patient there is even higher accuracy, which indicates the reliability and suitability of the model for the desired case

```
Covid-19:
  Precision: 1.0
  Recall: 1.0
  F1 Score: 1.0

Normal:
  Precision: 0.95
  Recall: 0.95
  F1 Score: 0.9500000000000001

Viral Pneumonia:
  Precision: 0.95
  Recall: 0.95
  F1 Score: 0.9500000000000001

Overall Accuracy: 0.9696969696969697
```

DATA AUGMENTATION

- In an attempt to improve the model's score, we implemented data augmentation techniques such as rescaling, rotation, shear, and zoom. This was motivated by the fact that the current dataset primarily consists of frontal chest X-ray images with similar appearances.

- While training with these augmented samples didn't lead to a performance increase, it's reassuring to know that, according to our understanding, X-ray institutes capture chest X-rays in a standardized position and orientation.
- This suggests that the original dataset, despite its lack of variety, possesses sufficient reliability.

CONCLUSIONS

- As you can see, the use of a convolutional network in our model provides an almost complete prediction of Covid-19 among patients who came for a chest x-ray.
- In this problem, and in many other problems, just as it is necessary to deepen the level of learning to several layers, it is necessary not to go too deep, thus not only to burden the complexity of the model, but inclining it to learn of data that does not help to solve the problem, and to use simple methods to reduce the dimensions required for the problem.