UNIVERSITÀ DI PISA

Master Degree in
Artificial Intelligence and Data Engineering

Progetto
Computational Intelligence and Deep Learning

Chest X-Ray Image Classification for Covid-19 Detection Using Synthetic Data Augmentation

Pietro CALABRESE
Francesco MARABOTTO

Contents

1 Introduction

Coronavirus is a viral disease that affects in various ways and can cause respiratory problems, even acute ones. A positive chest X-ray of infected patients is a crucial step in the battle against COVID-19. To this end, approaches based on Deep Learning techniques have been introduced ([4], [6], [12], [20]) and they have shown good accuracy.

One of the biggest challenges in the medical imaging field is to be able to work, and therefore to train the models, only on small datasets; in fact, there are not many chest X-ray images that are publicly accessible. We report a description of the dataset we used in our work in section 2. To overcome the problem of image scarcity, we used data augmentation. In the first part of our work we used standard data augmentation techniques, that use simple modifications to incorporate affinity like image transformations and color adjustments, such as scaling, flipping, converting, improving contrast or brightness, blurring, and sharpening, white balance, etc. This classical data augmentation is fast, reliable, and easy. However, the changes are limited because it is structured to turn an existing sample into a slightly altered sample. In the second part of our work we used synthetic data augmentation instead. One of the most innovative techniques to generate synthetic images consists in the use of a Generative Adversarial Network (GAN). In section 7 we illustrate the idea behind the GANs and the procedure we followed to generate the images.

As mentioned above, we have structured our work in two parts. In the first part we trained a VGG16 and a ResNet50 using transfer learning, briefly described in the section 2, and a custom made from scratch CNN. In the second part, as described in the previous paragraph, we generated X-ray images of the chest using a GAN and we added these images to the initial dataset in order to be able to train the models again.

Going into more detail, let's briefly introduce the content of the various sections. In the section 2 we show the dataset we used, through the description of its properties and the preprocessing operations we carried out on the images; furthermore, we present the Transfer Learning technique used in training, through the feature extraction technique, the VGG16 network and the ResNet50 network. In section 3 we describe the metrics used to evaluate the results. In section 4 and 5 we present the VGG16 and ResNet50 models, respectively. In each of the two sections we have described the architecture of the two networks and the design of the two models; finally, we present and briefly discuss the training results of these two pre-trained models. We present the custom-made CNN in section 6; as for the other two previous sections, we described the CNN architecture, the model design and presented and discussed the results obtained. In section 7 we discuss the approach based on the GAN and previously introduced. In section 8 we show the results of the second training, i.e. after adding the images generated by the GAN, and section 9 concludes the work by providing a summary of what has been done.

1.1 Related Works

This section presents the most recent Deep Learning techniques used in the detection of Covid19 from X-rays.

Matias Cam Arellano and Oscar E Ramos [2] used the DenseNet121 pre-trained CNN model whose last layer alone is retrained for detecting COVID-19 from chest radiographs using open databases. As the model is already trained for detecting different

lung diseases, the network provided distinctive features with an accuracy of 94.7%. In [4] COVID-19 and pneumonia are detected from chest X-ray images using a three-step process. The Conditional Generative Adversarial Network (C-GAN) is used at the first step to segment the lung region from the CXR images. In the second step, the feature extraction network which is a combination of traditional feature extraction algorithms and deep CNNs extracts the features from the segmented lung images. Various machine learning classifiers were used at the final step to classify the CXR images based on the extracted features. Binary Robust Invariant Scale Key-points (BRISK) in combination with VGG-19 produced the highest classification accuracy of 96.6%.

Khandaker Foysal Haque et al [8] developed a CNN model that obtained an accuracy of 97.56% in detecting COVID-19 from CXR images. The proposed model that is trained with four convolutional layers performed better when compared with the one trained with three and five convolutional layers. In [10] a combined CNN-LSTM (Convolutional Neural Network – Long Short Term Memory) was introduced to automatically diagnose COVID-19 from chest X-ray images. The model produced an accuracy of 99.4% which used CNN for extracting the deep features and LSTM for classifying the abnormality.

In [18] COVID-CheXNet was proposed to detect COVID-19 from chest X-ray images. The model yielded a very good accuracy of 99.99% by fusing the results generated by two pre-trained CNN models ResNet34 and HRNet. Linda Wang et al [19] introduced a DCNN named COVID-Net which is one of the first open-source architectures for detecting COVID-19 from CXR images. The authors combined five different open-access datasets and created the open-access dataset COVIDx which is tested on COVID-Net, VGG-19 and ResNet 50. COVID-Net performed well with an accuracy of 93.3%. In [15] a DCNN was proposed that used multiple patches of the lungs with random cropping to avoid overfitting. FC-DenseNet-103 performed the segmentation of the lung region and ResNet-18 performed the classification of the disease which produced 91.9% accuracy.

Julian et al [3] evaluated the performance of a model using three different preprocessing schemes and obtained a classification accuracy of 91.5%. In [7] Khalid El Asnaoui et al compared seven deep learning models for detecting and classifying COVID-19 pneumonia. The input X-ray and CT images are pre-processed to improve the quality and Inception-ResNet V2 obtained the highest accuracy of 92.18% with data augmentation and transfer learning. In [13] a classification method based on Advanced Squirrel Search Optimization Algorithm (ASSOA) was proposed that used two stages to classify different abnormalities from X-ray images. Pretrained ResNet 50 model is used at the first stage for feature extraction and at the second stage, the proposed ASSOA algorithm is applied for feature selection. Multilayer Perceptron Neural Network (MLP) is used for the classification of infected cases. The accuracy attained using the Kaggle dataset is 99.26% and using the chest X-ray COVID-19 GitHub images is 99.7%.

Md Manjural Ahsan et al [1] analyzed the performance of six modified pre-trained models. The obtained accuracy was up to 100% for VGG-16 and MobileNet V2 in identifying the COVID-19 patients. Pradeep Kumar Chaudhary and Ram Bilas Pachori [5] introduced the Fourier-Bessel series expansion-based dyadic decomposition (FBD) where an X-ray image is decomposed into sub-band images. Each sub-band image is fed to the pre-trained ResNet50 model where the deep features are extracted. The extracted features are ensembled and fed to the softmax classifier which classified pneumonia caused by COVID-19 versus other pneumonia with an accuracy of 98.66%.

Afshar Shamsi et al [11] proposed a transfer learning based uncertainty aware system for identifying COVID-19 infected cases from X-ray and CT images. Four pre-trained models are used for feature extraction and these features are passed through deep learning models to perform the classification task. The ResNet 50 model along with the SVM (Support Vector Machine) classifier achieved the best results with an accuracy of 87.9%. In [20] a multi-input deep convolutional attention network (MIDCAN) was proposed that was able to handle 3D chest Computed Tomography and 2D chest X-ray images simultaneously. A new convolutional block attention module (CBAM) is included in the model to improve the accuracy of the model to 98.02±1.35%. Chaimae Ouchicha et al [16] developed an advanced tool for the detection of COVID-19 from chest X-ray images named CVDNet. It consists of two parallel columns that have the same structures but different kernel sizes to detect the local and the global features. The output from the two columns is concatenated and the model produced an accuracy of 96.69%.

Table 1 summarizes some results achieved in the state of the art of this domain:

work	**network**	**accuracy[%]**	**precision[%]**	**recall[%]**	**f1-score[%]**	**classes**
[1]	VGG16	100	100	100	100	2
[1]	ResNet50	93	96	93	93	2
[19]	ResNet50	93.3	-	-	-	3
[17]	VGG16	95	-	-	-	2
[6]	VGG16	86	86	86	86	3

Table 1: Related works

2 Dataset Presentation and Preprocessing Operations

This chapter describes the dataset used and the preprocessing operations that we carried out on the images of this dataset, necessary to be able to input them to the various networks used in this work.

2.1 Properties of the Dataset

The dataset we intend to use in our project is public and available for academic purposes at the following link: https://www.kaggle.com/tawsifurrahman/covid19-radiography-database.

This dataset is divided in four different classes:

- **COVID-19 data**: COVID data are collected from different publicly accessible dataset, online sources and published papers:
  - 2473 CXR images are collected from padchest dataset link
  - 183 CXR images from a Germany medical school link
  - 559 CXR image from SIRM, Github, Kaggle & Tweeter link1 link2 link3 link4
  - 400 CXR images from another Github source link
- **Normal images**: 10192 Normal data are collected from from three different dataset.
  - 8851 RSNA link
  - 1341 Kaggle link
- **Lung opacity images**: 6012 Lung opacity CXR images are collected from Radiological Society of North America (RSNA) CXR dataset link
- **Viral Pneumonia images**: 1345 Viral Pneumonia data are collected from the Chest X-Ray Images (pneumonia) database link

We have analyzed the dataset and we have found that the images of the Lung Opacity class could give problems because, from the research we have carried out, they are easily confused for a machine with the other classes. Lung opacities are vague, fuzzy clouds of white in the darkness of the lungs, which makes detecting them a real challenge. In order not to risk altering the learning we have therefore decided to remove them.

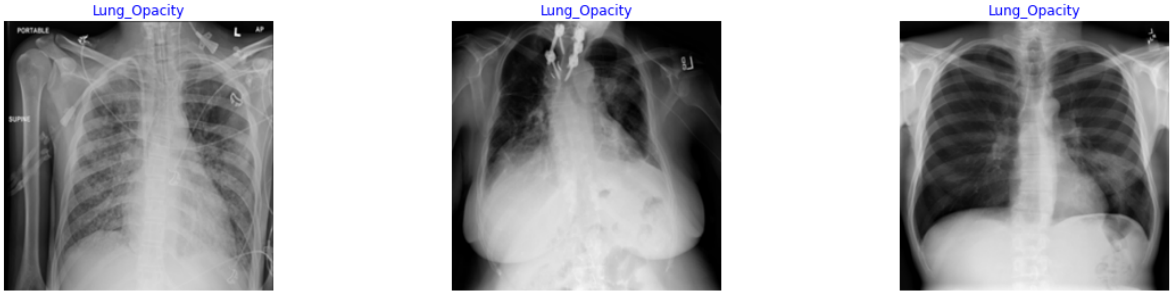


Figure 1: Lung Opacity samples

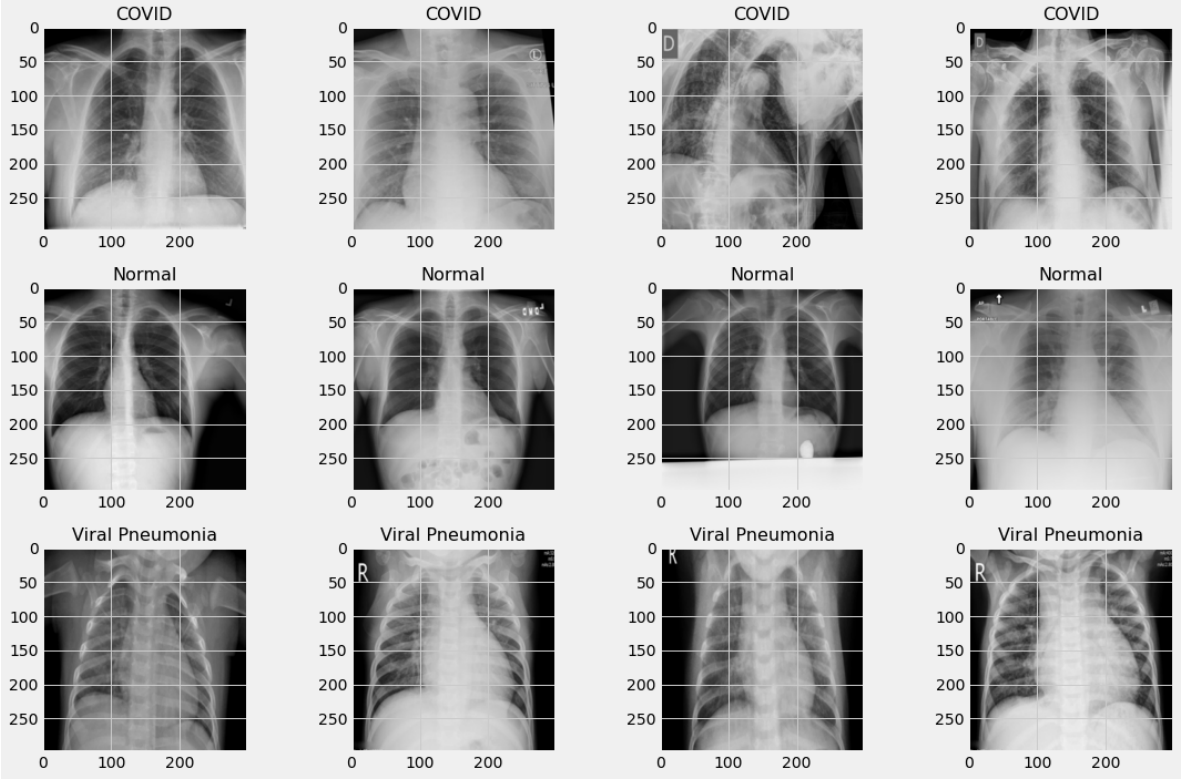


Figure 2: Samples of images

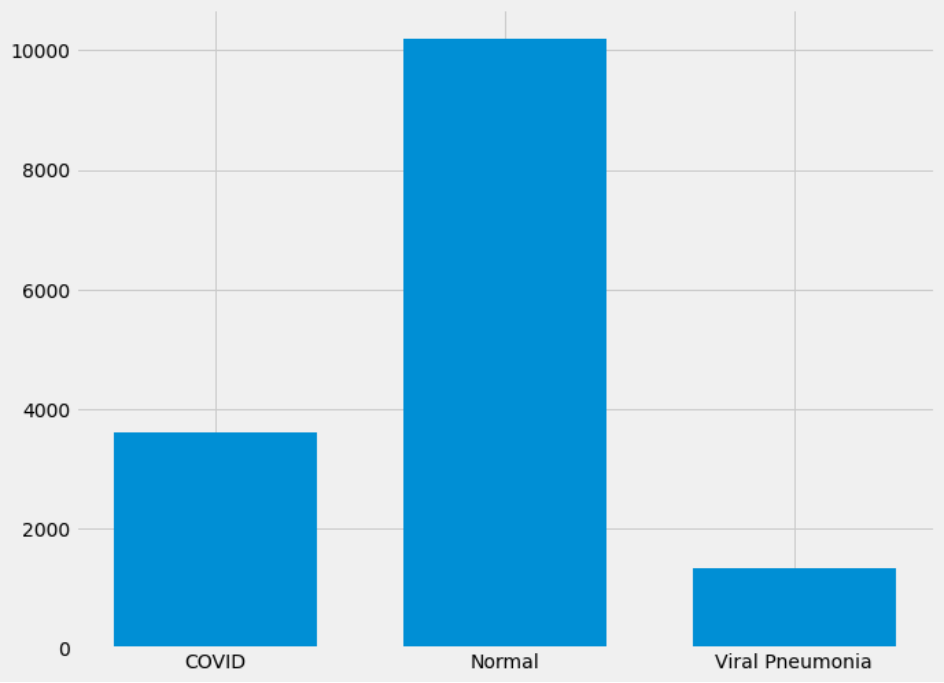


Figure 3: Distribution of samples

2.2 Preprocessing Operations

First of all, we carried out some preliminary preprocessing operations.

In the first step we have standardized the size of the images to that of the default template size. This way we were able to give the images in input to the various networks used. For greater clarity, the images in our dataset were initially 299×299 in size and they were in black and white; after this preprocessing operation they have been resized to the standard size of the model, that is $224 \times 224 \times 3$.

Second, to generalize the model, we applied data augmentation, as mentioned in the introduction. We added this step to generate more images and ensure that the model can understand that each image with its rotations was the same image. Indeed, data augmentation is a strategy that overcomes the data scarcity problem. It increases the number of samples in the dataset by making slight variations in the already existing samples. As for the data augmentation, we have carried out the following operations for each image: $rotation\_range$, $rescale$, $shear\_range$, $zoom\_range$, $horizontal\_flip$, $width\_shift\_range$, $height\_shift\_range$.

Finally, the dataset was split into 12122 training images, 1515 validation images and 1516 testing images.

2.3 Transfer Learning

Transfer Learning is an approach where a neural network model trained for a particular task is reused for a model on another task. For the new task, only the layers that are very close to the output units are retrained. The pre-trained model has to be trained with a sufficient amount of data because it gains knowledge about feature extraction of the image which is going to be transferred to another model. The main application of transfer learning is the classification of medical images for emerging diseases due to the limited availability of samples. Transfer learning has the benefit that the training time of the model decreases and is computationally less expensive as only a few layers are retrained. Since the models are already trained, it does not require a vast amount of data. Figure 4 illustrates the concept of transfer learning. Here the knowledge gained by *Model 1*, which is trained with a large amount of data is transferred to *Model 2* to perform a related task when the amount of data available to train *Model 2* is limited [14].

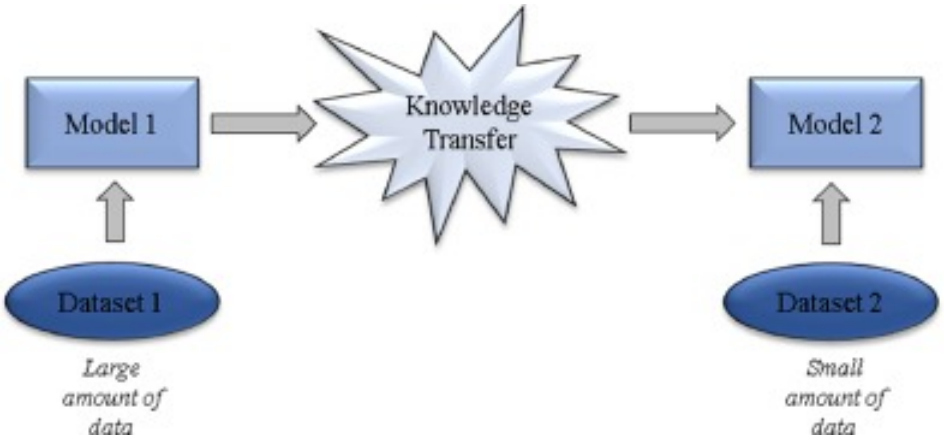


Figure 4: Concept of Transfer Learning

3 Metrics

The parameters used to evaluate the models shown so far are the **confusion matrix** and a **classification report**, which contains the *accuracy*, *recall*, *precision*, and *f-measure*. In particular, since the dataset is unbalanced, reference has been made mainly to the F1-measure metric.

The metrics used have the following structure:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

TP is the true positives, which is the correctly classified patients. TN is the true negatives that do not belong to a specific class and are classified correctly. The FP is the false positives, which are data wrongly classified as positives of a specific class. The FN is false negatives that belong to a specific class and is classified as does not belong to that class.

As accuracy is not a good metric to use when dealing with unbalanced datasets, we focused on f1-measure in evaluating model performance. Indeed, the problem is that an high accuracy value sounds like a great result, whereas your model performs very poorly. We can solve the problem in two ways: one through resampling and the other through metrics, that is, as mentioned before, considering the f1-measure and not the accuracy. We have used both methods in our work. In fact, we proceeded to use undersampling techniques to balance the dataset but, given the scarce presence of samples of the "Viral Pneumonia" class, this made it very difficult to train the model as the general dataset did not assume adequate dimensions; this approach is therefore only mentioned. Instead, we used the technique of using alternative metrics; in particular, the f1-measure instead of accuracy. In fact, f1-measure takes into account not only the number of prediction errors the model makes, but also looks at the type of errors that have been made.

4 VGG16

This section proposes an approach based on the VGG16 architecture, developed by Karen Simonyan and Andrew Zisserman in 2014. VGG16 is a pre-trained network on ImageNet, a dataset that contains 14 million tagged images and 1000 classes. To exploit this pre-trained network, we have chosen to use the *feature extraction* technique. We finally evaluated the model through the confusion matrix and a classification report, which contained precision, recall and f1-score.

4.1 Feature Extraction Technique

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier, which is trained from scratch. The feature extraction technique is particularly useful when dealing with autoencoders, bag-of-words and, as in our case, with image processing.

4.2 Visual Geometry Group (VGG16)

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.17% top-5 test accuracy in ImageNet. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's. Figure 5 presents VGG16 architecture.

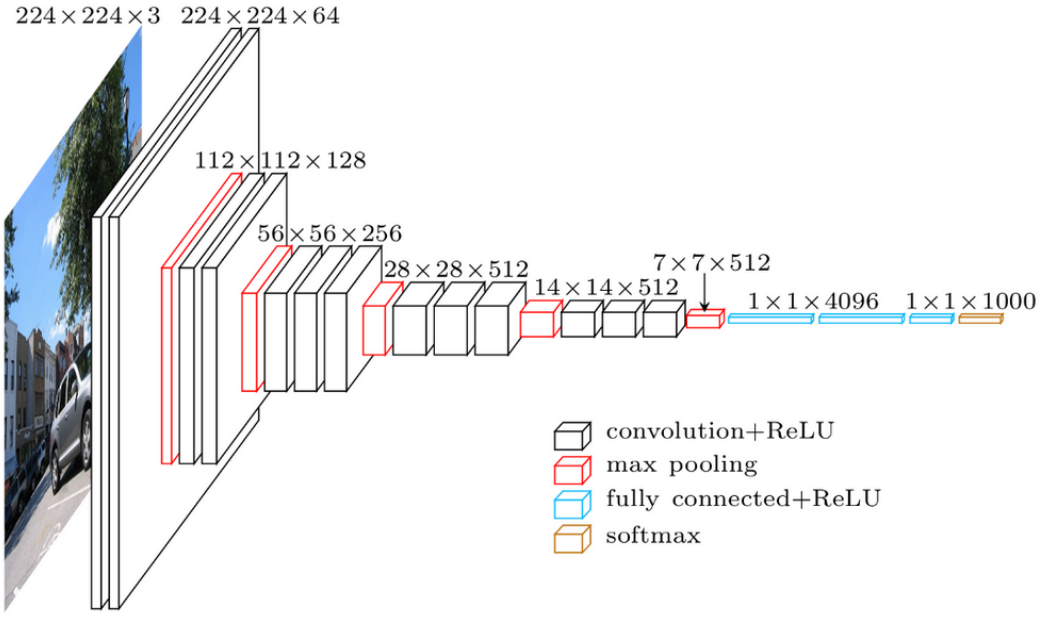


Figure 5: VGG16 Architecture

The input to *conv1* layer is of fixed size 224×224 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with a 3×3 dimension (which is the smallest size to capture the notion of left/right, up/down, center). The convolution stride is fixed to 1 pixel; the spatial padding of convolutional layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 convolutional layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers (not all the convolutional layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

4.3 Model Design

This section presents the building steps for the modified VGG16 architecture. This architecture can be used in medical diagnosis.

The model was trained on the dataset that was explained in Section 2. We set the model training to 100 epochs. However, thanks to an *EarlyStop* condition, the model has been training for 21 epochs. The criterion chosen within this function would have stopped the training of the model as soon as there had not been a lowering of the validation loss for 5 consecutive epochs.

In our modified VGG16 model, we removed the top fully connected layers and output layers from the original architecture and we added other fully connected layers and an output layer. This modified VGG16 is divided into two parts, the frozen part (i.e. not trainable part), which presents the original first layers and weights, and the trainable part, which presents the last added four layers that should have been trained. The four layers with dropout techniques are used to avoid overfitting and they are added after the flattening step.

The four added levels are:

- The first layer, *flatten_3*.
- The second layer, *dense_6*.
- The third layer, *dropout_3*.
- Finally, the fourth layer, *dense_7*.

The second layer uses rectified linear function ($ReLU$) as an activation function. The fourth layer, i.e. the output one, has three neurons as it is a categorical classification problem and it uses the *softmax* activation function. The loss function used is *categorical cross-entropy* loss as we have three classes, and the used optimizer is the Adam one, which takes a learning rate parameter of 0.0001.

In summary, the model has the structure shown on the next page:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 128)               3211392
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 17,926,467
Trainable params: 3,211,779
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 6: VGG16 Model

4.4 Presentation and Discussion of the Results

The modified VGG16 has been trained and tested on the dataset presented in the section 2. The total number of X-rays for training is 12122, for validation is 1515, and for testing 1516. The confusion matrix presents the classification error used to describe the model performance on the testing dataset. The figure 7 presents the confusion matrix of our model:

	TP	**TN**	**FP**	**FN**
COVID	333	1125	11	47
Normal	998	446	58	14
Viral Pneumonia	113	1389	3	11

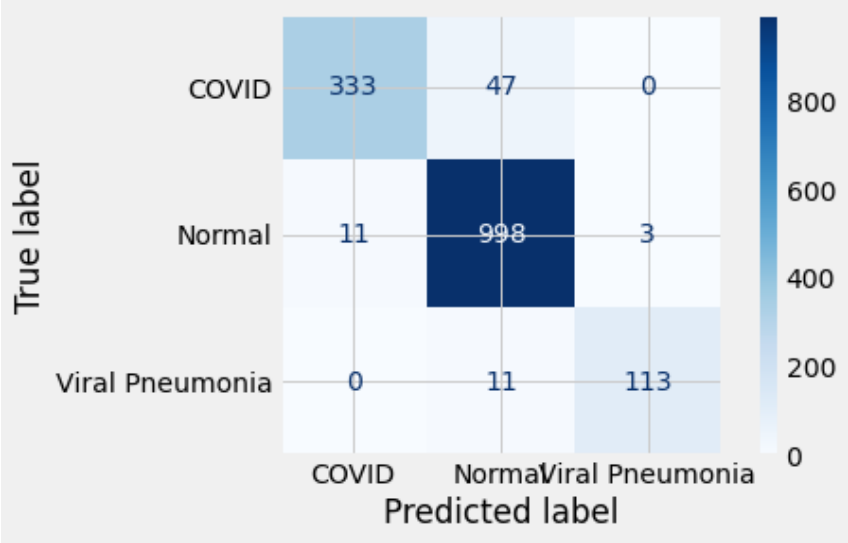


Figure 7: VGG16 Confusion Matrix

The classification report is presented in table 2, which contains the performance metrics precision, recall, f1-score with relative supports.

	precision	**recall**	**f1-score**	**support**
COVID	0.97	0.88	0.92	380
Normal	0.95	0.99	0.97	1012
Viral Pneumonia	0.97	0.91	0.94	124
accuracy			0.95	1516
macro avg	0.96	0.92	0.94	1516
weighted avg	0.95	0.95	0.95	1516

Table 2: VGG16 Classification Report

Figure 8 reports the performance of the modified VGG16 model in terms of loss and accuracy.

The below graph shows that the training accuracy is 92.21% and the training loss is 0.20. The validation accuracy is 93.15% and validation loss is 0.19. As training and validation accuracy values are close and the loss values for training and validation are close, our model presents trains well, and no overfitting occurs.

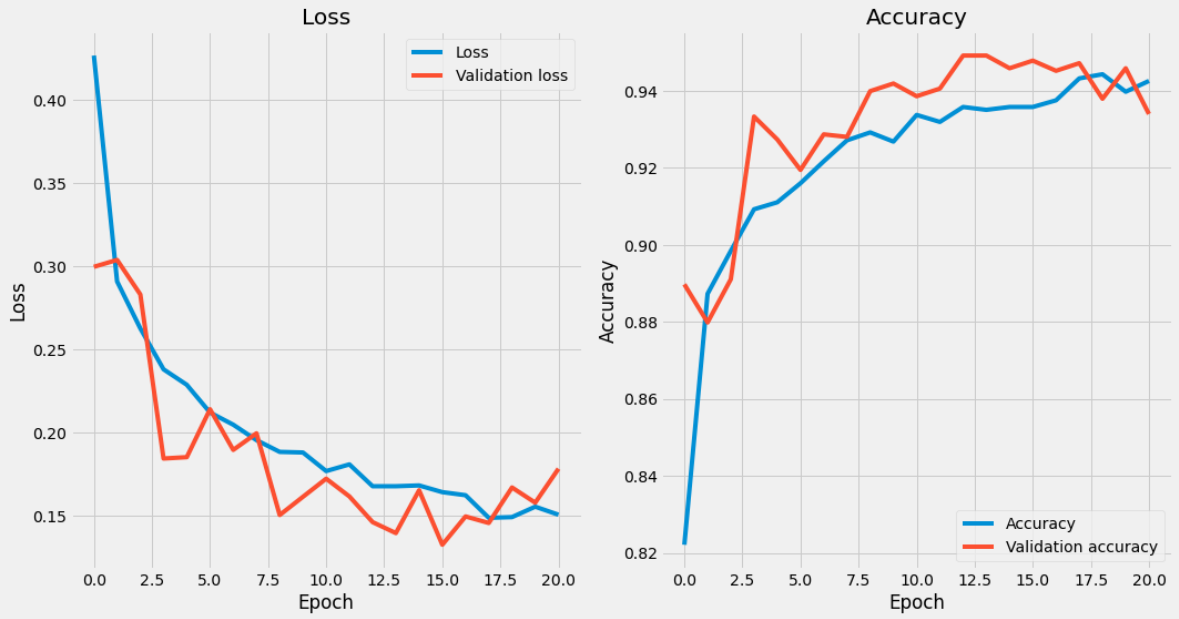


Figure 8: VGG16 Loss and Accuracy

5 ResNet50

In this section we present the approach based on the *ResNet50* model.

5.1 ResNet50 Architecture

ResNet Architecture consists of an input layer, 4 ensuing stages and an output layer, as shown in 9. Each stage represents a part of the process we are executing consecutively. It receives input from previous stages, executes one step of the CNN, and provides the output. ResNet is divided into 5 stages, where the structure of Stage 0 can be regarded as a preprocessing of input, and the last four stages have a more similar structure. We have an input stem that performs a 7×7 convolution, has an output channel of 64, and a stride of 2. Next, we have a 3×3 max pooling-layer, with a stride of 2. In this layer, we are effectively decreasing 4 times the input width and height, and we increase the channel size to 64. On Stage 2, and the subsequent ones, we have a down-sampling block and residual blocks. The residual blocks works in the same manner as the down-sampling one, the only difference would lie in the stride of the convolutions, which in this case would be 1. Changing the number of residual blocks we obtain different models, thus with ResNet50 we are just indicating the number of the convolutional layers we have in the network.

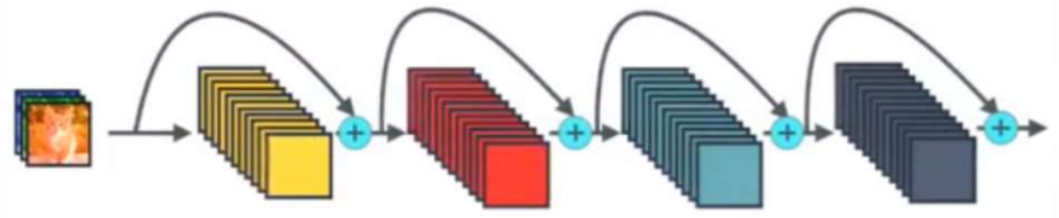


Figure 9: ResNet Building Blocks

5.2 Model Design

We chose to use the pre-trained ResNet50 model on the ImageNet dataset. The option include_top=False allows feature extraction by removing the last dense layers. Later, we need to connect our pretrained model with the new layers of our model. We used a *global pooling* to connect the dimensions of the pretrained layers with our new layers. Then we put 2 *dense* layers with *relu* activation function and a last *dense* layer with *softmax* as activation function to perform the final classification (i.e. "COVID", "Normal" and "Viral Pneumonia").
The final model takes the following form:

5.3 Presentation and Discussion of the Results

We trained and tested the ResNet50 on the dataset presented in the section 2. The total number of X-rays for training is 12122, for validation is 1515, and for testing 1516. The confusion matrix presents the classification error used to describe the model

Layer name	**Output size**	**Layer**
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3 x 3 max pool, stride 2 [1 x 1, 64 3 x 3, 64 1 x 1, 256] x 3
conv3_x	28×28	$[1 \times 1, 128$ $3 \times 3, 128$ $1 \times 1, 512] \times 4$
conv4_x	14×14	$[1 \times 1, 256$ $3 \times 3, 256$ $1 \times 1, 1024] \times 6$
conv5_x	7×7	$[1 \times 1, 512$ $3 \times 3, 512$ $1 \times 1, 2048] \times 3$
fc1	1×1	global average pooling in_features = 2048, out_features = 2048
dense	none,128	relu out_features = 128
dense_1	none,64	relu out_features = 64
dropout	1 x 1	dropout 0.5
dense_2	none,3	Softmax out_features = 3

Table 3: ResNet50 Model

performance on the testing dataset. The figure 7 presents the confusion matrix of our model:

	TP	TN	FP	FN
COVID	341	1128	8	39
Normal	999	458	46	13
Viral Pneumonia	117	1387	5	7

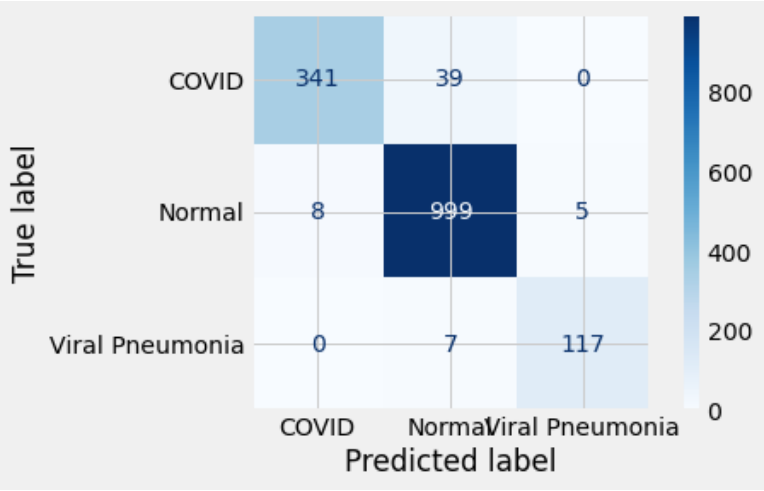


Figure 10: ResNet50 Confusion Matrix

The classification report is presented in table 4, which contains the performance metrics precision, recall, f1-score with relative supports.

	precision	recall	f1-score	support
COVID	0.98	0.90	0.94	380
Normal	0.96	0.99	0.97	1012
Viral Pneumonia	0.96	0.94	0.95	124
accuracy			0.96	1516
macro avg	0.96	0.94	0.95	1516
weighted avg	0.96	0.96	0.96	1516

Table 4: ResNet50 Classification report

Figure 11 reports the performance of the ResNet50 model in terms of loss and accuracy. The below graph shows that the training accuracy is 93.03% and the training loss is 0.19. The validation accuracy is 92.98% and validation loss is 0.19. As training and validation accuracy values are close and the loss values for training and validation are close, our model presents trains well, and no overfitting occurs.

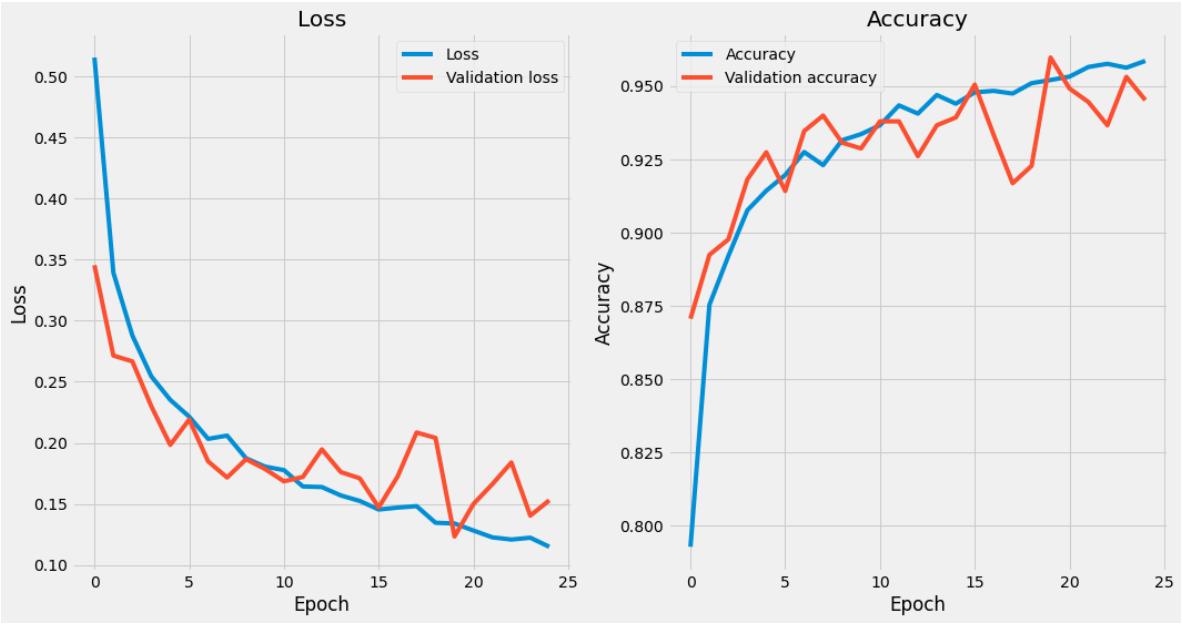


Figure 11: ResNet50 Loss and Accuracy

6 CNN custom-made

The **Convolutional Neural Network** (CNN) is a type of Deep Neural Network (DNN) that is frequently employed in a variety of applications. In this chapter we present our own custom CNN (Convolutional Neural Network) model, built from scratch and later trained and tested with our dataset.

6.1 CNN Architecture

CNN architecture is stacked with three primary layers namely i) Convolutional layer ii) Pooling layer and iii) Fully Connected layer. The basic CNN architecture along with the layers is shown in Figure 12. The responsibility of convolutional and pooling layers are feature extraction and the fully connected layer is for classification purposes. There are also two more parameters, dropout layer and activation function, that are defined apart from these layers.

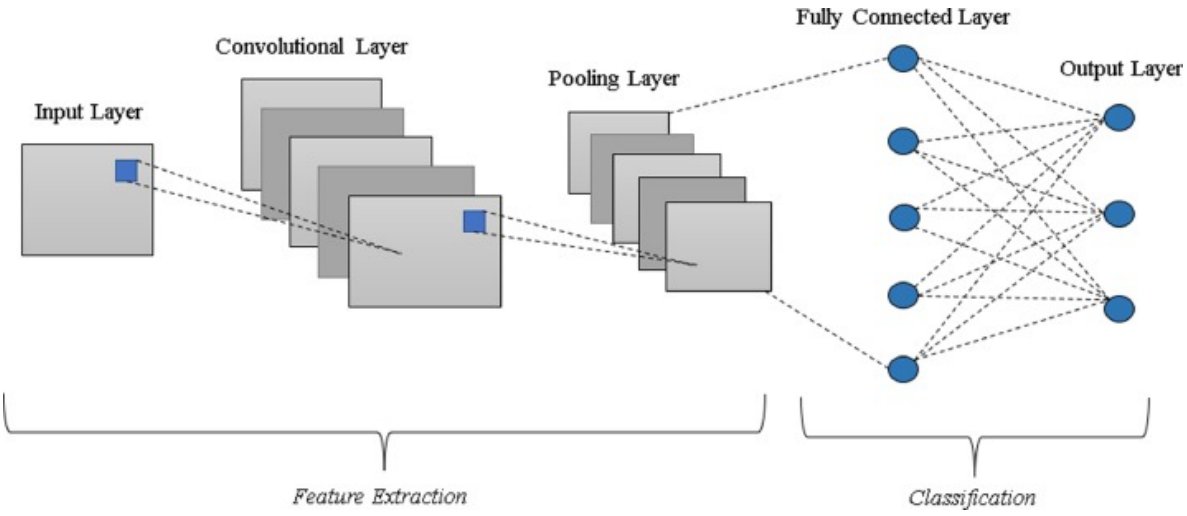


Figure 12: Architecture of a standard Convolutional Neural Network

We now briefly describe the building blocks of a CNN.

6.1.1 Convolutional Layer

This is the basic layer responsible for extracting features from the input image. The convolutional layer has a set of kernels that basically are filters. The kernel is a matrix, which is smaller than the input data, that slides over the input from left to right and from top to bottom and performs a dot function with the input data. The result is the generation of a feature map. The feature maps of the later layers are built by combining the feature maps of the earlier layers.

6.1.2 Pooling Layer

The pooling layer is the second layer which follows the convolutional layer. This layer performs downsampling and thereby reduces the number of parameters and computa-

tions. The most commonly used pooling operation is *max pooling*, that produces the maximum element from the feature map.

6.1.3 Fully Connected Layer

The last layer in the architecture is the fully connected layer (FC layer). Its dimension is equal to the number of output classes. The input from the previous stages is fed into the fully connected layer, which then classifies the images.

6.1.4 Dropout Layer

A dropout layer is used before the output layers. It randomly discards a few neurons from the neural network, and it helps prevent overfitting. The Dropout layer only applies during the training phase; on the contrary, no values are dropped during inference.

6.1.5 Activation Function

Activation functions get the output from the previous layer and convert it to a form suitable for the next layer to consider that as its input. They can be used at any part of the network. It adds nonlinearity to the network. *Rectified Linear Unit* (ReLu), *softmax*, *sigmoid* and *tanh* are some of the commonly used activation units and ReLu is the most widely used in deep learning models.

6.2 Model Building

We can now move on to describe how the model was built from scratch. For this, we just need to keep adding layers, mostly *Conv2D* to extract features, *MaxPooling2D* to perform downsampling of the image and a *BatchNormalization* layer in order to improve the performance of the model.

The model has the following structure:

```
Model: "CustomCNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_2 (Batch (None, 224, 224, 3)       12
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 54, 54, 32)        18464
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 27, 27, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 27, 27, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 23328)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               2986112
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 387
=================================================================
Total params: 3,043,695
Trainable params: 3,043,689
Non-trainable params: 6
_________________________________________________________________
```

Figure 13: CustomCNN Model

All of the fully connected layers use rectified linear function (*relu*) as an activation function. The output layer has three neurons as it is a categorical classification problem and it used the *softmax* activation function. The loss function used is *categorical cross-entropy* loss as we have three classes, and the used optimizer is the Adam one, which takes a learning rate parameter of 0.0001.

The final *dense* layer is what provides the output of the network (i.e. "*COVID*", "*Normal*", "*Viral Pneumonia*").

6.3 Presentation and Discussion of the Results

Our CNN built from scratch, like the other networks in this paper, has been trained and tested on the dataset presented in the section 2. The total number of X-rays for training is 12122, for validation is 1515, and for testing 1516. The confusion matrix presents the classification error used to describe the model performance on the testing dataset. The figure 14 presents the confusion matrix of our model:

	TP	**TN**	**FP**	**FN**
COVID	331	1114	22	49
Normal	987	440	64	25
Viral Pneumonia	105	1385	7	19

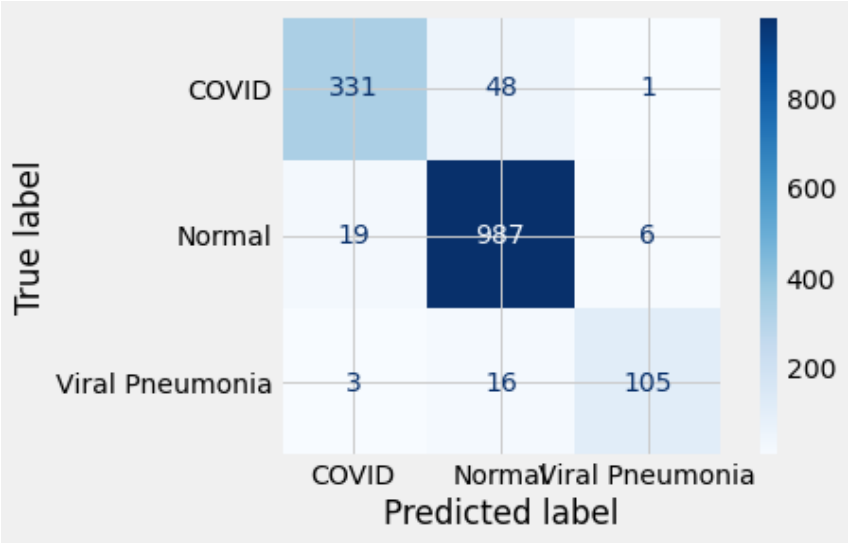


Figure 14: Confusion Matrix

The classification report is presented in table 5, which contains the performance metrics precision, recall, f1-score with relative supports.

	precision	**recall**	**f1-score**	**support**
COVID	0.94	0.87	0.90	380
Normal	0.94	0.98	0.96	1012
Viral Pneumonia	0.94	0.85	0.89	124
accuracy			0.94	1516
macro avg	0.94	0.90	0.92	1516
weighted avg	0.94	0.94	0.94	1516

Table 5: Classification report

Figure 15 reports the performance of the custom CNN model in terms of loss and accuracy.

The below graph shows that the training accuracy is 85.53% and the training loss is 0.34. The validation accuracy is 89.89% and validation loss is 0.27. As training and validation accuracy values are close and the loss values for training and validation are close, our model presents trains well, and no overfitting occurs.

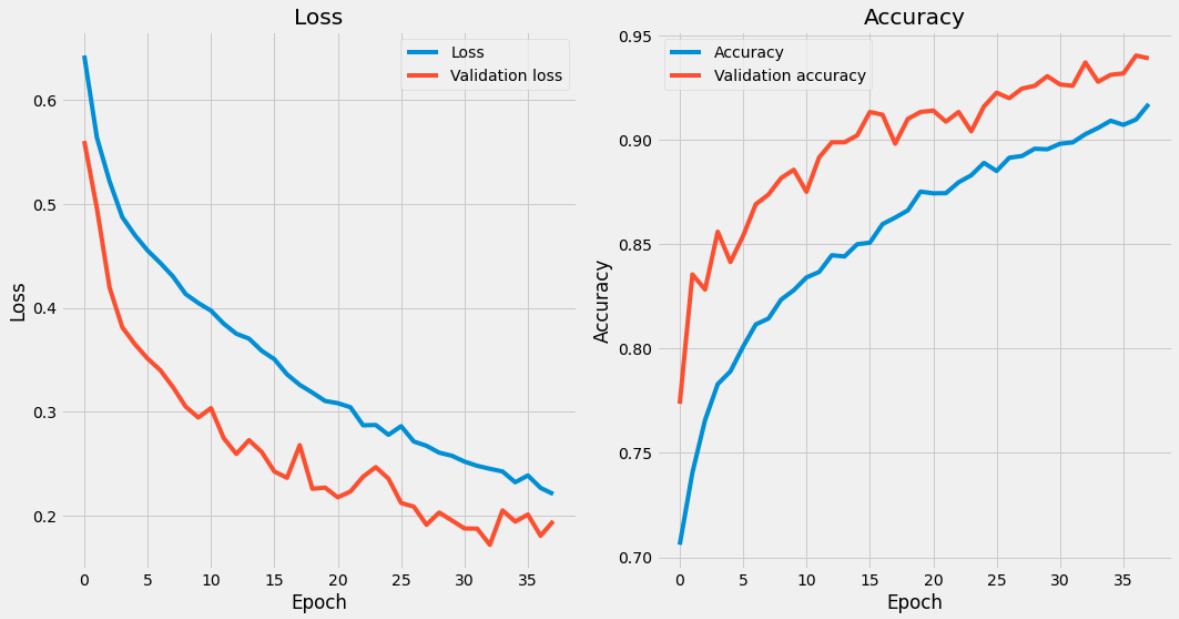


Figure 15: Loss and Accuracy

7 GAN

The biggest challenge in the medical imaging domain is small datasets. The medical image collection is a very expensive and tedious process that requires the participation of radiologists and researchers. Also, since the COVID-19 outbreak is recent, sufficient data of chest X-ray (CXR) images is difficult to gather. We propose to alleviate the drawbacks by using synthetic data augmentation.

Data augmentation methods are employed to extend the training dataset artificially. Current data augmentation techniques use simple modifications to incorporate affinity like image transformations and color adjustments, such as scaling, flipping, converting, improving contrast or brightness, blurring, and sharpening, white balance, etc. This classical data augmentation is fast, reliable, and easy. However, in this augmentation, the changes are limited because it is structured to turn an existing sample into a slightly altered sample. In other words, classical data augmentation does not produce completely unseen data. A modern, advanced form of augmentation is synthetic data augmentation which overcomes the limitations of classical data augmentation. Generative Adversarial Network (GAN) is one such innovative model that generates synthetic images. It is a powerful method to generate unseen samples with a min-max game without supervision. The general concept of the GANs is to use two opposing networks (G(z) and D(x)), where one (G(z) generator) produces a realistic image to trick the other net that is equipped to better discriminate between the true and false images (D(z) discriminator).

7.1 Auxiliary Classifier Generative Adversarial Network (AC-GAN)

Generative Adversarial Networks (GANs) utilizes two neural networks that compete with one another to create new virtual instances of data which can be transmitted as real data. GANs are extensively used for image generation. In this paper, we use a version of GAN called Auxiliary Classifier GAN to perform data augmentation. GANs find it difficult to generate high-resolution samples from highly variable data sets. Conditional GAN (CGAN) is a variant of GAN which allows the model to rely on outside information to improve the sample quality. In CGAN, a latent space point and a class label are given as input to the generator and it attempts to generate an image for that class. The discriminator is provided with an image as well as a class label, and it decides if the image is true or false. AC-GAN is a type of CGAN that transforms the discriminator to predict a particular image's class label instead of receiving it as an input. It stabilizes the training process and allows the generation of high-quality images while learning a representation that is independent of the class label. ACGAN architecture is shown in Figure 16.

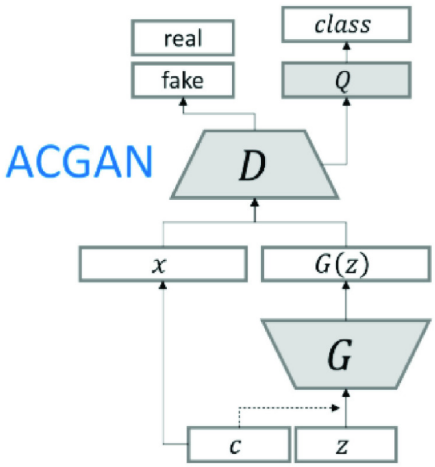


Figure 16: ACGAN Architecture

ACGAN applies the associated class label c and noise z to each produced sample. The generator G utilizes both to produce $X_{\text{fake}} = G(c, z)$ images. The discriminator D gives a distribution of probability over class labels and sources.

7.2 CovidGAN

The GAN architecture we used to generate the images is now presented. We have taken over the CovidGAN architecture from [17] and making changes in order to make the classification multiclass and not just binary.

7.2.1 Generator Architecture

The generator takes a latent vector of noise (which is a random normal distribution with 0.02 standard deviation) and class label as input, to output a single $112 \times 112 \times 3$ image. The class label is passed through an embedding layer of 50 dimensions for categorical input. Then, it is further passed through a 7×7 node dense layer with linear activation to output a $7 \times 7 \times 1$ tensor. The point in latent space is interpreted by a $1024 \times 7 \times 7$ node dense layer to give activations that can be reshaped to $7 \times 7 \times 1024$ to get many copies of a low-resolution version of the output image. The tensors generated from class label and noise (that is $7 \times 7 \times 1$ and $7 \times 7 \times 1024$) are concatenated and passed through four transpose convolutional layers to upsample the $7 \times 7 \times 1024$ feature maps, first to $14 \times 14 \times 512$, then $28 \times 28 \times 256$, then $56 \times 56 \times 128$ and finally to $112 \times 112 \times 3$. Each transpose convolutional layer, except for the last one, is followed by a batch normalization layer and an activation layer. The model uses methodologies such as ReLU activation, a kernel of size $(5, 5)$, stride of $(2, 2)$ and a hyperbolic tangent (tanh) activation function in the output layer. The total parameters of the generator are approximately 22 million. The output of the generator is an image of shape $112 \times 112 \times 3$.

7.2.2 Discriminator Architecture

The discriminator model is a CNN architecture that has two output layers and takes one image of shape $112 \times 112 \times 3$ as input. The model outputs a prediction if the image is

real (class = 1) or fake (class = 0), and also outputs the class label that is COVID-CXR or Normal-CXR. Each block of discriminator represents a convolutional layer, which is followed by a batch normalization layer, an activation layer and a dropout layer with 0.5 probability. The input is downsampled from $112 \times 112 \times 32$ to $56 \times 56 \times 64$, then $28 \times 28 \times 128$, then $14 \times 14 \times 256$ and finally to $7 \times 7 \times 512$. The model uses a kernel of size $(3, 3)$, a stride that changes alternatively from $(1, 1)$ to $(2, 2)$ and LeakyReLU activation function with a slope of 0.2. Discriminator has approximately 2 million parameters. The final output is flattened and the probability of the image's reality and the probability of the image belonging to each class is estimated. The first output layer with *sigmoid function* predicts the realness of the image. The second output layer with *softmax function* predicts the label.

The complete CovidGAN architecture is shown in Figure 17.

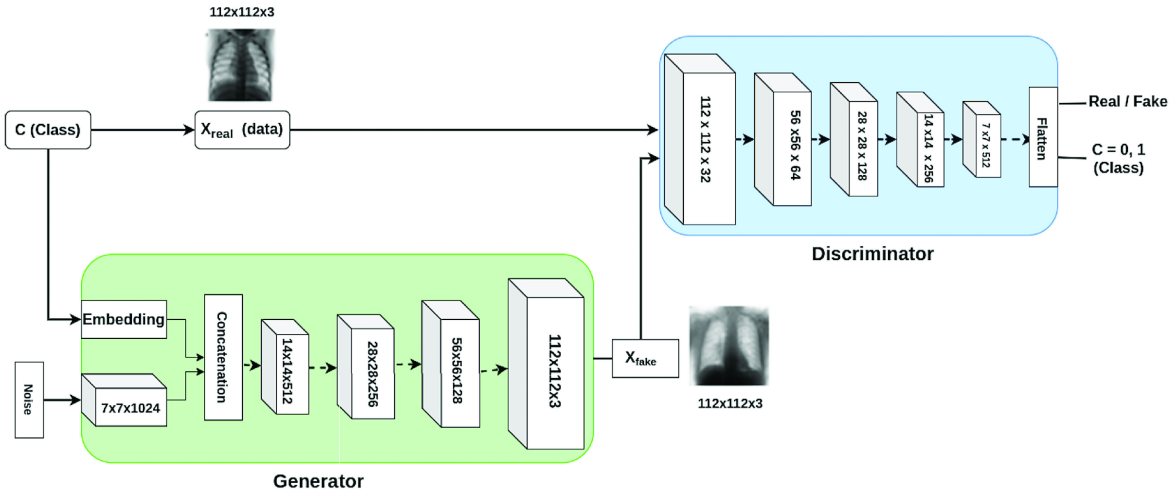


Figure 17: CovidGAN Architecture

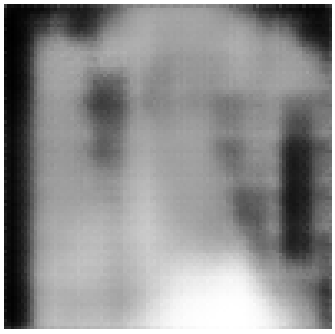
7.2.3 Training Process

The generator model is stacked on top of the discriminator model. Initially, the layers of the discriminator are set as non-trainable. Thus, only the generator gets updated via the discriminator. This forms a composite model of GAN, which we call CovidGAN. The CovidGAN is trained to synthesize CXR images for both COVID-CXR and Normal-CXR class. The image preprocessing step involved resizing $(112 \times 112 \times 3)$ and normalizing the images from $[0, 255]$ to $[-1, 1]$[1]. Adam is used as the optimizer function. The following hyperparameters are used for training CovidGAN: batch_size = 64, learning_rate = 0.0002, beta = 0.5 (beta is the momentum of Adam optimizer), number of epochs = 1500. The GAN gets optimized using two loss functions, one for each output layer of the discriminator. The first layer uses *binary_crossentropy* and second *sparse_categorical_crossentropy*. The complete architecture is trained using Keras deep learning library.

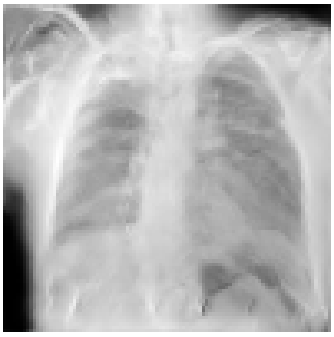
[1](Normalization is a process that changes the range of pixel values. Its purpose is to convert an input image into a range of pixel values that are more familiar or normal to the senses)

7.3 Generated Images

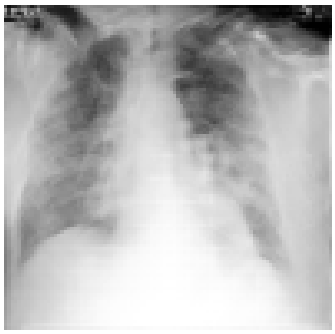
In this section we show a sample of the images that have been incrementally generated by GAN; indeed we have chosen to show the images generated in four different iterations. These images are shown in Figure 18.



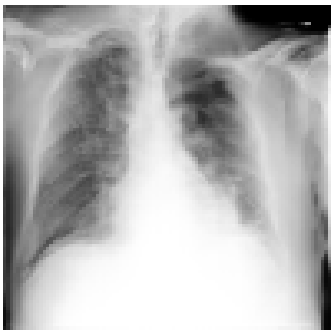
(a) GAN: image generated at step 1



(b) GAN: image generated at step 500



(c) GAN: image generated at step 1000



(d) GAN: image generated at step 1500

Figure 18: GAN: sample of generated images

8 Model Training with Synthetic Data Augmentation

This section relates to the training of the models following the addition of the images generated through the GAN to the dataset present in the section 2, i.e. the one with which the training presented in the sections 4, 5 and 6 were performed. As mentioned in the introduction, indeed, the difference between the first model training and the training that we present in this section lies in the different type of data augmentation management. In the first training we used a classic data augmentation, while in the second we used the synthetic data augmentation, through the images generated by the GAN.

8.1 VGG16

In this subsection we report the results obtained with the VGG16. The confusion matrix presents the classification error used to describe the model performance on the testing dataset. The figure 19 presents the confusion matrix of our model:

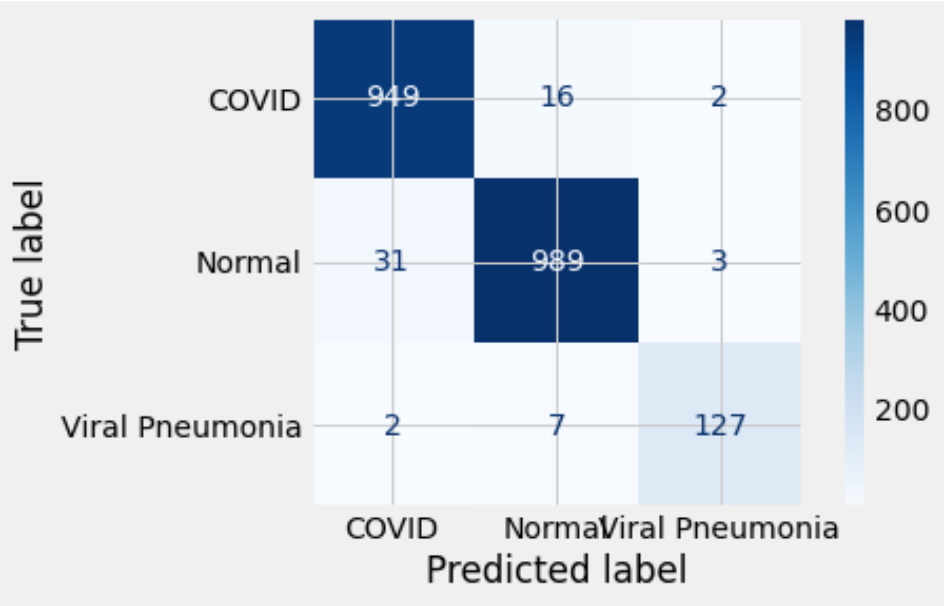


Figure 19: VGG16 2nd Training: Confusion Matrix

The classification report is presented in table 6, which contains the performance metrics precision, recall, f1-score with relative supports.

	precision	recall	f1-score	support
COVID	0.97	0.98	0.97	967
Normal	0.98	0.97	0.97	1023
Viral Pneumonia	0.96	0.93	0.95	136
accuracy			0.97	2126
macro avg	0.97	0.96	0.96	2126
weighted avg	0.97	0.97	0.97	2126

Table 6: VGG16 2nd Training: Classification report

Figure 20 reports the performance of the custom CNN model in terms of loss and accuracy. The below graph shows that the training accuracy is 97.30% and the training loss is 0.09. The validation accuracy is 97.37% and validation loss is 0.09. As training and validation accuracy values are close and the loss values for training and validation are close, our model presents trains well, and no overfitting occurs.

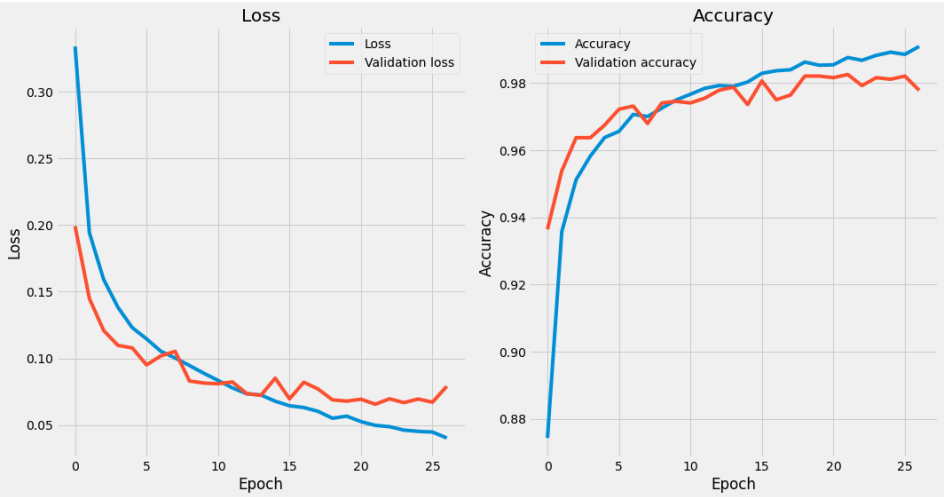


Figure 20: VGG16 2nd Training: Loss and Accuracy

8.2 ResNet50

In this subsection we report the results obtained with the ResNet50. The confusion matrix presents the classification error used to describe the model performance on the testing dataset. The figure 21 presents the confusion matrix of our model:

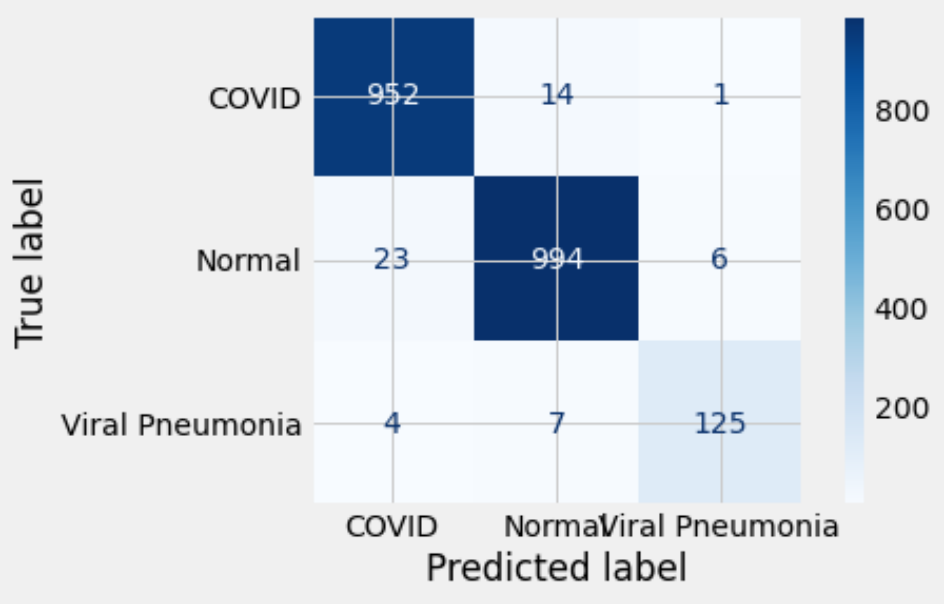


Figure 21: ResNet50 2nd Training: Confusion Matrix

The classification report is presented in table 7, which contains the performance metrics precision, recall, f1-score with relative supports.

	precision	**recall**	**f1-score**	**support**
COVID	0.97	0.98	0.98	967
Normal	0.98	0.97	0.98	1023
Viral Pneumonia	0.95	0.92	0.93	136
accuracy			0.97	2126
macro avg	0.97	0.96	0.96	2126
weighted avg	0.97	0.97	0.97	2126

Table 7: ResNet50 2nd Training: Classification report

Figure 22 reports the performance of the custom CNN model in terms of loss and accuracy. The below graph shows that the training accuracy is 96.89% and the training loss is 0.10. The validation accuracy is 97.07% and validation loss is 0.09. As training and validation accuracy values are close and the loss values for training and validation are close, our model presents trains well, and no overfitting occurs.

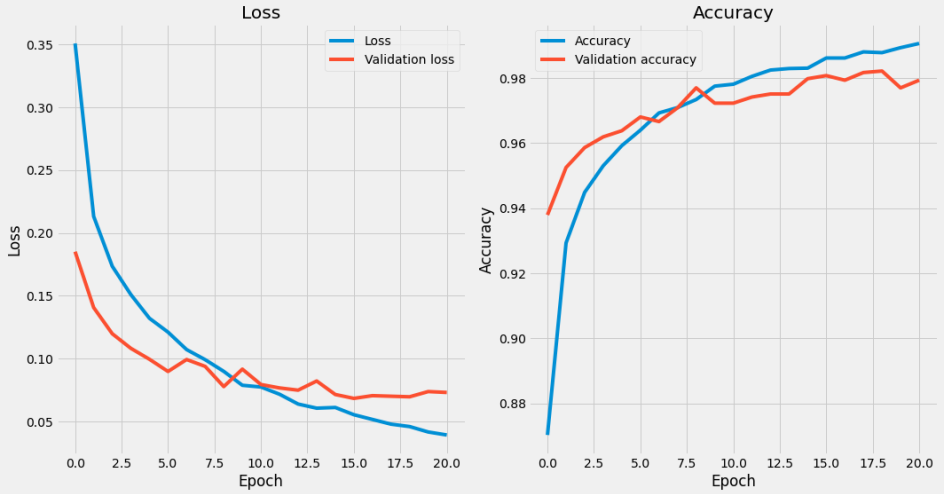


Figure 22: ResNet50 2nd Training: Loss and Accuracy

8.3 Custom CNN

In this subsection we report the results obtained with the Custom CNN. The confusion matrix presents the classification error used to describe the model performance on the testing dataset. The figure 23 presents the confusion matrix of our model:

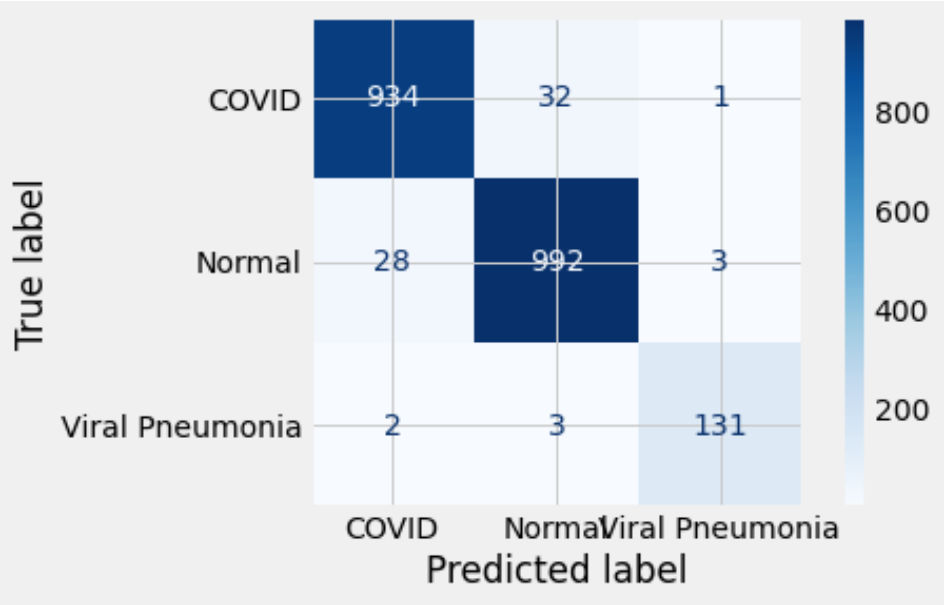


Figure 23: CNN 2nd Training: Confusion Matrix

The classification report is presented in table 8, which contains the performance metrics precision, recall, f1-score with relative supports.

	precision	recall	f1-score	support
COVID	0.97	0.97	0.97	967
Normal	0.97	0.97	0.97	1023
Viral Pneumonia	0.97	0.96	0.97	136
accuracy			0.97	2126
macro avg	0.97	0.97	0.97	2126
weighted avg	0.97	0.97	0.97	2126

Table 8: Classification report

Figure 24 reports the performance of the custom CNN model in terms of loss and accuracy. The below graph shows that the training accuracy is 96.37% and the training loss is 0.11. The validation accuracy is 96.04% and validation loss is 0.13. As training and validation accuracy values are close and the loss values for training and validation are close, our model presents trains well, and no overfitting occurs.

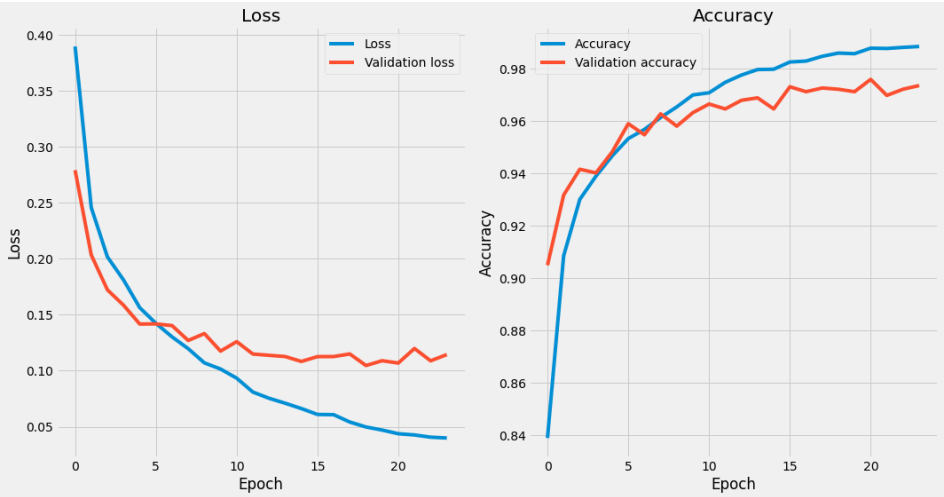


Figure 24: CNN 2nd Training: Loss and Accuracy

9 Conclusions

In this section, which concludes our work, we make some general considerations on what has been done. When dealing with diagnostic images, we must keep in mind that they are sensitive data and therefore not present in large quantities. The aim we set ourselves at the beginning of the work was to be able to demonstrate that the use of artificially generated images can significantly improve the classification of images. In our case, the models trained with the use of the samples generated through the GAN had a 5/6% increase in performance compared to the models that use the classic data augmentation. The latter, in the field of diagnostic images, very often produces artifact samples that are not reflected in the real world.

As regards the choice of the networks to be used, we carried out a preliminary study on the state of the art of the classification of thoracic X-ray images, especially those relating to Covid19 which is the most topical topic.

With regard to the CovidGAN network, we took advantage of the network present in [17] by adapting it to our domain; in particular, we have moved from a binary classification to a multiclass classification. At this stage, unfortunately, we cannot report a reliable estimate on the training times of a GAN as we have used the resources offered by Google Colaboratory which are time-limited and highly dependent on the usage load of all users. In fact, we proceeded in incremental steps through the use of checkpoints. Having noted that models trained with sythetic data augmentation have better performance than those trained with classic data augmentation, we can conclude that in this domain the use of artificially generated data can lead to an overall improvement of the classification system.

The possibility remains open, for future work, to use other pre-trained models or to carry out a much more accurate analysis on the time difference required for the training of a GAN compared to the direct use of the classic data augmentation. This work, compared to most of the works in the literature, has brought the classification towards a finer granularity using three classes and not a binary classification as often happens. Furthermore, through GAN we have brought about a general improvement in terms of performance.

Table 9 concludes our work by reporting a comparison between the results obtained with the classic data augmentation and those obtained with the synthetic data augmentation:

network	**accuracy[%]**	**precision[%]**	**recall[%]**	**f1-score[%]**
VGG16	93.15	93.19	93.02	93.10
VGG16 - GAN	97.37	97.45	97.33	97.39
ResNet50	92.98	93.06	92.87	92.96
ResNet50 - GAN	97.07	97.10	97.03	97.06
CNN	89.89	90.22	89.21	89.69
CNN - GAN	96.04	96.18	95.86	96.02

Table 9: Comparison of results using classical data augmentation and synthetic data augmentation

References

[1] Md Manjurul Ahsan et al. "Detecting SARS-CoV-2 from chest X-Ray using artificial intelligence". In: *Ieee Access* 9 (2021), pp. 35501–35513.

[2] Matías Cam Arellano and Oscar E Ramos. "Deep Learning Model to Identify COVID-19 Cases from Chest Radiographs". In: *2020 IEEE XXVII International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*. IEEE. 2020, pp. 1–4.

[3] Julián D Arias-Londoño et al. "Artificial Intelligence applied to chest X-Ray images for the automatic detection of COVID-19. A thoughtful evaluation approach". In: *Ieee Access* 8 (2020), pp. 226811–226827.

[4] Abhijit Bhattacharyya et al. "A deep learning based approach for automatic detection of COVID-19 cases using chest X-ray images". In: *Biomedical Signal Processing and Control* 71 (2022), p. 103182.

[5] Pradeep Kumar Chaudhary and Ram Bilas Pachori. "Automatic diagnosis of COVID-19 and pneumonia using FBD method". In: *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2020, pp. 2257–2263.

[6] Javier Civit-Masot et al. "Deep Learning System for COVID-19 Diagnosis Aid Using X-ray Pulmonary Images". In: *Applied Sciences* 10.13 (2020). ISSN: 2076-3417. DOI: 10.3390/app10134640. URL: https://www.mdpi.com/2076-3417/10/13/4640.

[7] Khalid El Asnaoui and Youness Chawki. "Using X-ray images and deep learning for automated detection of coronavirus disease". In: *Journal of Biomolecular Structure and Dynamics* 39.10 (2021), pp. 3615–3626.

[8] Khandaker Foysal Haque et al. "Automatic detection of COVID-19 from chest X-ray images with convolutional neural networks". In: *2020 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE. 2020, pp. 125–130.

[9] Muneeb ul Hassan. "Vgg16-convolutional network for classification and detection". In: *Neurohive. Dostopno na: https://neurohive. io/en/popular-networks/vgg16/[10. 4. 2019]* (2018).

[10] Md Zabirul Islam, Md Milon Islam, and Amanullah Asraf. "A combined deep CNN-LSTM network for the detection of novel coronavirus (COVID-19) using X-ray images". In: *Informatics in medicine unlocked* 20 (2020), p. 100412.

[11] AS Jokandan et al. "An Uncertainty-aware Transfer Learning-based Framework for Covid-19 Diagnosis. arXiv 2020". In: *arXiv preprint arXiv:2007.14846* ().

[12] Eman Naser Karajah and Mohammad Awad. "Covid-19 Detection From Chest X-Rays Using Modified VGG 16 Model". In: *2021 International Conference on Promising Electronic Technologies (ICPET)*. 2021, pp. 46–51. DOI: 10.1109/ICPET53277.2021.00015.

[13] El-Sayed M El-Kenawy et al. "Advanced meta-heuristics, convolutional neural networks, and feature selectors for efficient Covid-19 x-ray chest image classification". In: *Ieee Access* 9 (2021), pp. 36019–36037.

[14] H. Mary Shyni and E. Chitra. "A COMPARATIVE STUDY OF X-RAY AND CT IMAGES IN COVID-19 DETECTION USING IMAGE PROCESSING AND DEEP LEARNING TECHNIQUES". In: *Computer Methods and Programs in Biomedicine Update* 2 (2022), p. 100054. ISSN: 2666-9900. DOI: https://doi.org/10.1016/j.cmpbup.2022.100054. URL: https://www.sciencedirect.com/science/article/pii/S2666990022000064.

[15] Yujin Oh, Sangjoon Park, and Jong Chul Ye. "Deep learning COVID-19 features on CXR using limited training data sets". In: *IEEE transactions on medical imaging* 39.8 (2020), pp. 2688–2700.

[16] Chaimae Ouchicha, Ouafae Ammor, and Mohammed Meknassi. "CVDNet: A novel deep learning architecture for detection of coronavirus (Covid-19) from chest x-ray images". In: *Chaos, Solitons & Fractals* 140 (2020), p. 110245.

[17] Abdul Waheed et al. "CovidGAN: Data Augmentation Using Auxiliary Classifier GAN for Improved Covid-19 Detection". In: *IEEE Access* 8 (2020), pp. 91916–91923. DOI: 10.1109/ACCESS.2020.2994762.

[18] Alaa S Al-Waisy et al. "COVID-CheXNet: hybrid deep learning framework for identifying COVID-19 virus in chest X-rays images". In: *Soft computing* (2020), pp. 1–16.

[19] Linda Wang, Zhong Qiu Lin, and Alexander Wong. "Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images". In: *Scientific Reports* 10.1 (2020), pp. 1–12.

[20] Yu-Dong Zhang et al. "MIDCAN: A multiple input deep convolutional attention network for Covid-19 diagnosis based on chest CT and chest X-ray". In: *Pattern recognition letters* 150 (2021), pp. 8–16.