Pneumonia Classification on X-rays using Convolutional Neural Networks

Artificial Intelligence Applications - Prof: Petia Georgieva (petia@ua.pt) - Academic Year 2021/22

André Reis Fernandes
Number: 97977
MSc. Computacional Engineering
Department of Electronics,
Telecommunications and Informatics
andre.fernandes16@ua.pt
Workload : 50%

Gonçalo Jorge Loureiro de Freitas
Number: 98012
MSc. Computacional Engineering
Department of Electronics,
Telecommunications and Informatics
goncalojfreitas@ua.pt
Workload : 50%

***Abstract*—The article focuses on the research of image classification algorithms, namely the images indicate pathology of pneumonia caused by bacteria and viruses. We study various approaches and different CNN architectures to maximize the results of this study. In this project, the algorithms used give relatively positive classification results with an F1-Score of approximately 87%.**

***Index Terms*—Convulutional Neural Networks, Data Balancing, F1-Score, ROC, AUC**

I. Introduction

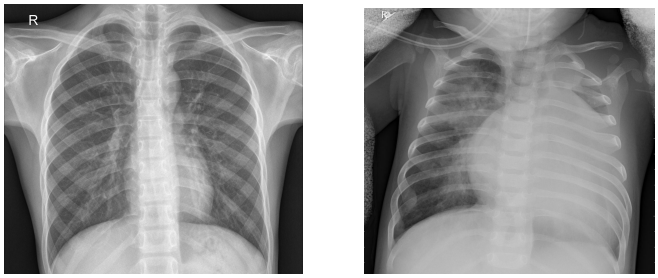
Pneumonia is an infection that affects one or both lungs. It causes the air sacs, or alveoli, of the lungs to fill up with fluid or pus. This disease impact in public health is now more severe than HIV infection, malaria, cancer, and myocardial coloration. Due to this, the identification of the causative agents of pneumonia is of utmost importance.[11]

To diagnose pneumonia, your healthcare provider will review your medical history, perform a physical exam, and order diagnostic tests such as a chest X-ray. It's here where the application of artificial intelligence and image processing technologies takes a part of this medical problem. From the chest X-ray image, artificial intelligence technology can help to classify diseases quickly and bring high accuracy in disease diagnosis.

In this project we build machine learning models based on Convolutional Neural Networks to classify if a person has Pneumonia or not, using the "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification dataset"[5]. We'll also be comparing our model to other models to see how effective it actually is.

II. The data set

The data set consists in chests X-Ray's images from persons with and without Pneumonia.The data set is already divided in training and test and, each one is also divided with normal and pneumonia images. To get the validation set we use a 80:20 split in the training set, where, from the total 100% of the original training set, 80% will go to the 'real' training set used for this model and 20% will go to the validation set. It is important to refer that we set a random state value for the train and validation split, this makes the result from the split always be the same.



(a) (b)

Fig. 1: Samples of Chest X-Ray images. (a) Normal and (b) Pneumonia

Since this is a binary data set, to label the data we've classified images with 0 as Normal chest X-ray and Pneumonia chest X-ray as 1:

0 - Normal Chest x-Ray
1 - Pneumonia Chest x-Ray

A. Data Overview

Let's first take a look at the histograms of the training, validation and test data.

Comparing all histograms, it's possible to notice that the Training set, Fig.2, is the one with more data, approximately 4200 data, about 1100 normal and 3100 pneumonia. In the validation data histogram, represented in Fig.3, there are about 1050 cases with approximately 270 normal cases and 770 pneumonia. Finally, in the Fig.4, the testing data histogram, there is about 600 data, 230 normal cases and 390 pneumonia cases.

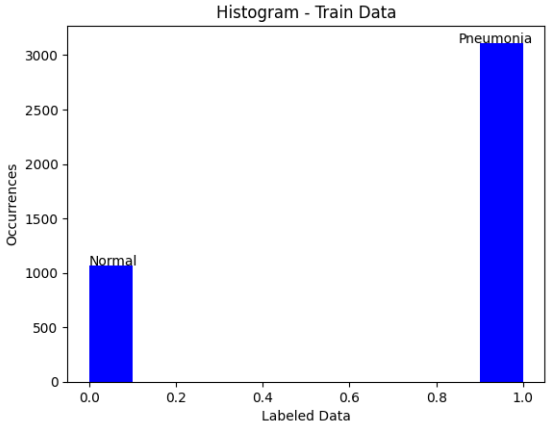


Fig. 2: Training data distribution.

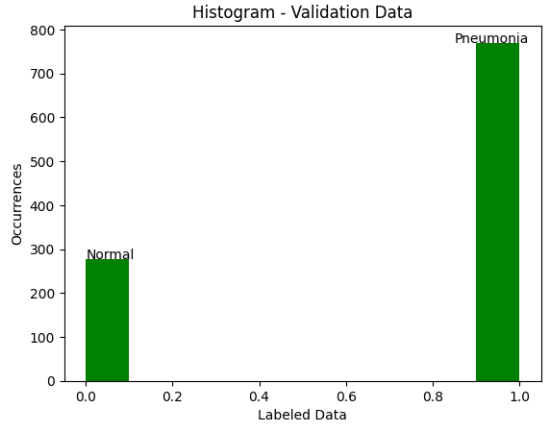


Fig. 3: Validation data distribution.

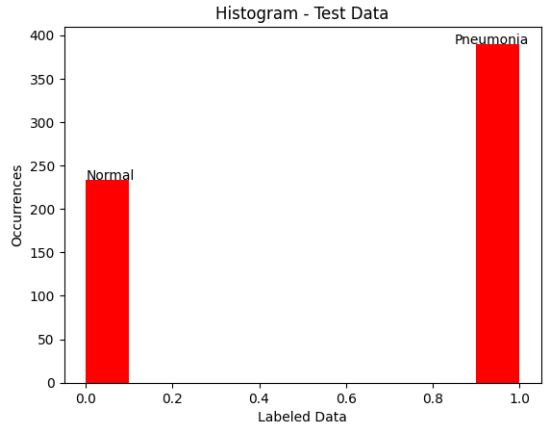


Fig. 4: Testing data distribution.

Analyzing all three histograms we can confirm that there are more pneumonia cases then normal cases which tells us a data balance is going to be needed.

Data balance can be obtained by one of the following methods:

1) Over-sampling
2) Under-sampling
3) Class weight
4) Threshold

For this problem, we are going to balanced the data set by adding weights to the classes (0 and 1). To calculate the weight of each class we do as following:

$$Weight_0 = \frac{1}{Normal\_Training} \times \frac{Total\_Train}{2.0}$$

$$Weight_1 = \frac{1}{Pneumonia\_Training} \times \frac{Total\_Train}{2.0}$$

Where $Weigth_0$ and $Weigth_1$ are the weights of the class 0 (Normal) and 1 (Pneumonia), respectively. Normal_Training and Pneumonia_Training represent the number of Normal and Pneumonia cases in the training set, respectively. As for Total_Train this represents the total number of cases in the training set. You can find more about balancing data in [6].

III. Slight Introduction to Convolutional Neural Networks

A Convolutional neural network (CNN or ConvNet), is a network architecture for deep learning which learns directly from data, which is very helpful as it eliminates the need for manual feature extraction.

CNNs are particularly useful for finding patterns, features or specifications in images. They can also be quite effective for classifying non-image data such as audio, time series, and signal data.

A basic CNN architecture consists of two key components:

1. A convolution tool that separates and identifies the distinct features of an image for analysis in a process known as Feature Extraction.
2. A fully connected layer that takes the output of the convolution process and predicts the image's class based on the features retrieved earlier.

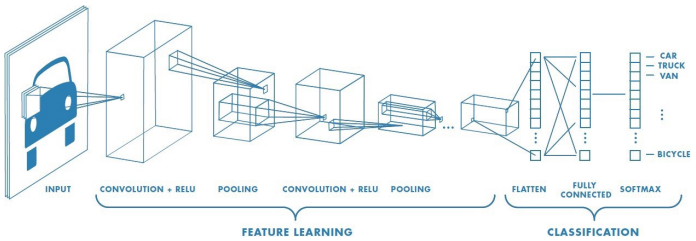


Fig. 5: Example of a CNN architecture. Source: [9]

When an image is given as input into a CNN, each of its inner layers generates various activation maps. Activation maps point out the relevant features of the given input image. Each of the CNN neurons generally takes the input in the form of a group/patch of the pixel, multiplies their values(colours) by the value of its weights, adds them up, and input them through the respective activation function.[10]

A more deep explanation of how CNNs work can be found in [9]

IV. Approach

We divide our work in 2 mini projects being the first mini project divided in 3 experiments, where we are going to compare the results of the model using unbalanced and balanced data, while fixing the number of epochs to 25, with the goal of seeing the difference of working with balanced and unbalanced data. For the final experiment we are going to implement the k-fold Cross-Validation method, with 10 folds, on the balanced approach in order to improve the performance of our model, using also 2 of the keras callback, this being,

1) Model Checkpoint - Save a copy of the best performing model only when an epoch that improves the metrics ends.
2) Early Stopping - Stop training when a monitored metric has stopped improving

For the Early Stopping callback we define a value of $patience = 5$, this is the number of epochs with no improvement after which the training will stop. More about k-fold Cross-Validation can be found here [2].

As for the second mini project, we are also going to use the balanced approach with 25 epochs but now he are going to compare this approach to 3 others. In the first one, using the balanced training set, we implement Data-Augmentation, inspired from [4], to test if this method is helpful when trying to get better results. Then we are going to compare our model with two well known CNN architectures, VGG19 and InceptionV3, both with Data Augmentation applied. In order to learn more about this architectures we have consulted [7] and [1], respectively. The summary's of this models can be found in the Appendix in Fig.20 and 24.

It's important to refer that, as we are going to work with unbalanced train and test set, the metrics used in all CNN models are the Precision and Recall. As for the loss we use the option *'binary_crossentropy'* and for the optimizer we use a manually defined learning rate of 0.0000001, remembering that the lower the learning rate is, the smoother the curve of the loss is, but more time, i.e., more epochs, is needed for the model to converge. To prepare our data, as CNNs work better with smaller numbers and our images values range and are not fixed, we normalize and scale down all the images to [180,180].

A. Project 1

1) The model: We have used the CNN architecture inspired by the one developed by Abhinav Sagar that can be found in [8]. This model can be described as following:

1. Five convolutional blocks comprised of convolutional layer, max-pooling and batch-normalization.
2. On top of it, a flatten layer followed by four fully connected layers
3. In between, some of the blocks have dropouts to reduce over-fitting.
4. Activation function was Relu throughout except for the last layer where it was Sigmoid as this is a binary classification problem.

In Figure 6 we have represented a summary of the parameters of our model. The rest of the model summary can be found in the Appendix in Fig.19.

```
=================================
Total params: 3,493,985
Trainable params: 3,493,025
Non-trainable params: 960
_________________________________
```

Fig. 6: CNN model parameters summary from [8]

Then, the next step is to compile and fit the model saving its history. For the balanced and unbalanced experiment we use, as refer before, 25 epochs with 200 steps per epoch. The higher number of steps, the better results that can be obtained, but the execution time of the code is also bigger. This leads to the following graphics for the Unbalanced, Fig.7, and Balanced, Fig.8, approaches.

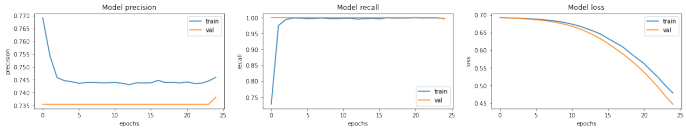


Fig. 7: Graphics for Unbalanced data of Mini Project 1 using 25 epochs.

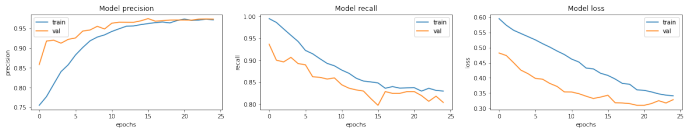


Fig. 8: Graphics for Balanced data of Mini Project 1 using 25 epochs.

It's possible to see in Fig.7 and 8 that the model hasn't converge at 25 epochs so, we can conclude that the number of epochs was too low to transform the input data into accurate output.

As for the k-fold Cross Validation experiment we use 10 folds and 25 epochs with 200 steps which this leads to the graphic, represented in Fig.9, for the loss for each fold and for each epoch were we can see, as excepted, the loss lowers in every fold.

B. Project 2

For the second project, we also use 25 epochs but now with 131 steps per epoch. As mention before, this project is divided in 4 experiments.

The first step, like in the first project, was prepare both of the train and test sets. For the Balanced and Balanced +

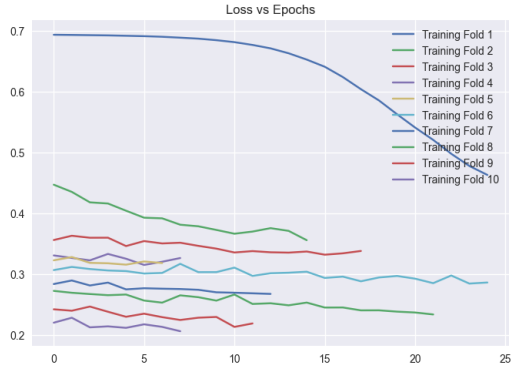


Fig. 9: Graphics for the loss data of the k-fold CV experiment of Mini Project 1 using 25 epochs and 10 folds for cross validation.

Data-Augmentation approach, the first two experiments, we have also used the model inspired by the one developed by Abhinav Sagar[8], described above, and for the two others we've use models available by keras which are based in pre-made architectures with pre-trained weights.
It's worth to mention that, different from the model from [8], the VGG19 and InceptionV3 architectures were implemented in linearly stacked layers i.e. there is no branching, every layer has one input and output and the output is the input of the next layer.
For the precision, recall and loss of each approach, we get the following graphics for the Balanced approach in Fig.10, for the Data-Augmentation approach in Fig.11 and, for the VGG19 and InceptionV3 approaches in Fig.12 Fig.13, respectively.

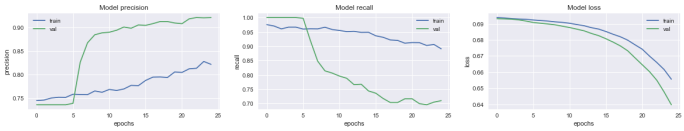


Fig. 10: Graphics for Balanced data of Project 2 using 25 epochs.

Fig. 11: Graphics for Data-Augmentation of Project 2 using 25 epochs.

Observing all four figures it is possible to confirm that the InceptionV3 is the only one converging when using 25 epochs. We can also see that the VGG19 was close to converge, probably needing only a few more epochs to converge. As for the Data-Augmentatio, Fig.11, we see that this approach got a lot of variatiom in the validation loss.

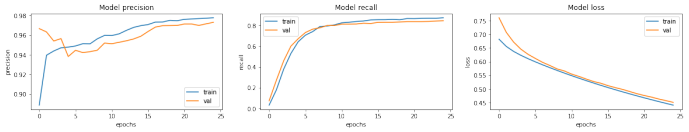


Fig. 12: Graphics for VGG19 of Project 2 using 25 epochs.

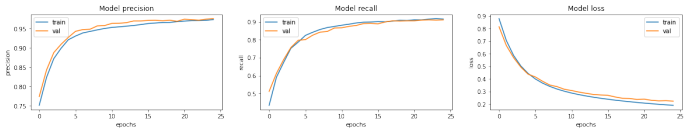


Fig. 13: Graphics for InceptionV3 of Project 2 using 25 epochs.

V. Results

Since the test set is unbalanced, the accuracy metric is not suitable so, we opted to use the F1 - Score metric to evaluate the performance of the model, calculated as following.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The metrics obtained for the first project are represented in the TableI and, for the second project, in the TableII.

Model	Loss	Precision(%)	Recall(%)	F1-Score(%)
Unbalanced	0.54	62.70	100.00	77.08
Balanced	0.41	87.47	84.10	85.75
k-fold CV	0.42	82.21	93.59	87.53

TABLE I: Results obtained for the first mini project - Number of epochs = 25.

Analysing the TableI we can see that, in fact, working with balanced data gets better results, in this case, a minor loss and a better F1-Score. As for the k-fold Cross Validation we see that this method improves the performance of our model, being the experiment with the higher F1-Score.

Model	Loss	Precision(%)	Recall(%)	F1-Score(%)
Balanced	0.64	83.78	79.49	81.58
Data-Augmentation	0.41	84.80	88.72	86.72
VGG19	0.50	84.63	88.97	86.75
InceptionV3	0.41	81.33	93.85	87.14

TABLE II: Results obtained for the second project - Number of epochs = 25.

Looking at the TableII we see that the InceptionV3 architecture is the one with a lower loss and higher F1 - Score. In terms of execution time, it's important to refer that the execution time of 1 VGG19 epoch is much higher than the execution time of 1 epoch of InceptionV3 as it is possible to see in Figure14.

As for the Data-Augmentation, this method also improves the performance of our model as we can see when comparing the first two experiments in the mini project 2, where the

```
Epoch 11/25
131/131 [==============================] - 245s 2s/step
- val_precision: 0.9513 - val_recall: 0.8117
Epoch 12/25
131/131 [==============================] - 236s 2s/step
```

(a)

```
Epoch 11/25
131/131 [==============================] - 51s 388ms/step
7 - val_precision: 0.9639 - val_recall: 0.8675
Epoch 12/25
131/131 [==============================] - 51s 389ms/step
```

(b)

Fig. 14: Execution time of (a) VGG19 and (b) InceptionV3 architectures

Balanced + Data Augmentation got lower loss and higher F1-Score than the Balanced approach.

If we look at the Balanced approaches for each project we see that the results in the second one were worse, this has to do with the number of steps per epoch used as we use 200 for the first project but only 131 for the second, the higher the number of steps the better the results will be.

Comparing the TablesI and II we see that the Balanced for the mini project 1, the InceptionV3 and Data-Augmentation from the mini project 2 got the lower loss, 0.41. As for the F1-Score, the higher value obtained was from the k-fold Cross Validation experiment on TableI. From this comparison and from the graphs of the loss of each approach, where we see that the InceptionV3 is the one converging faster so, we can conclude that an implementation of the InceptionV3 architecture with k-fold Cross Validation would the best approach.

A. ROC & AUC

Since our date set it's a binary classification problem we can calculate the ROC (Receiver Operating Characteristic) and AUC (Area Under the Curve). ROC is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'. The AUC is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve and higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.[3]

In a deeper view, when AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly; if the AUC = 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives; when 0.5<AUC<1 there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives. Finally, in case of AUC = 0.5 the classifier is not able to distinguish between Positive and Negative class points meaning either the classifier is predicting random class or constant class for all the data points.

For a better comparison, all the ROC's obtained were all plotted in the same Figure, except the k-fold Cross Validation, has it was done in a separate code. For the first project the ROC's are illustrated in Fig15 and Fig16, where we see that the k-fold CV has the best ROC curve and the Balanced and Unbalanced curves are very close, what can be misleading.

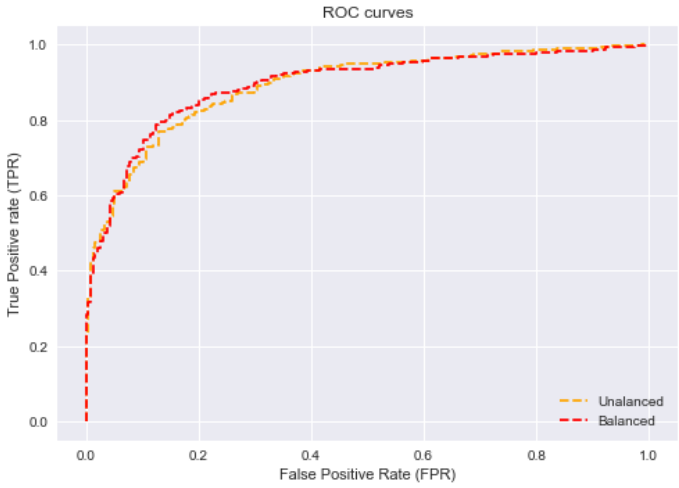


Fig. 15: ROC for mini Project 1 - 25 epochs with 200 steps per epoch.

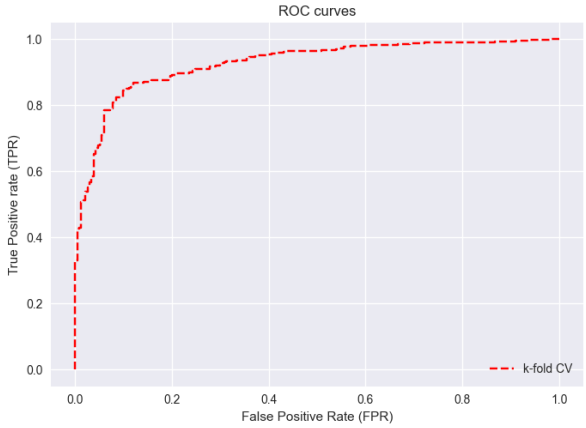


Fig. 16: ROC for mini Project 1 k-fold Cross Validation approach - 1o folds, 25 epochs with 200 steps per epoch.

The AUC values obtained for the first project are represented in the TableIII.

Model	AUC
Unbalanced	0.8915
Balanced	0.8946
k-fold CV	0.926

TABLE III: AUC for 1st Project - 25 epochs with 200 steps per epoch.

Analysing the values from TableIII, we see that, between the 2 first experiments, the balanced experiment has a higher value but the difference is almost null. This leads us to conclude that

the AUC is not a good metric to have in mind when working with unbalanced data. As for the AUC value from the k-fold CV experiment we see that this value is very close to 1, which means that here is a high chance that the classifier will be able to distinguish the positive class values from the negative class values.

The ROC for the second project is represented in Fig.17, where we can see that the Data Augmentation has the best ROC curve and the Balanced approach the worst. As for the AUC values, this are represented in TableIV.

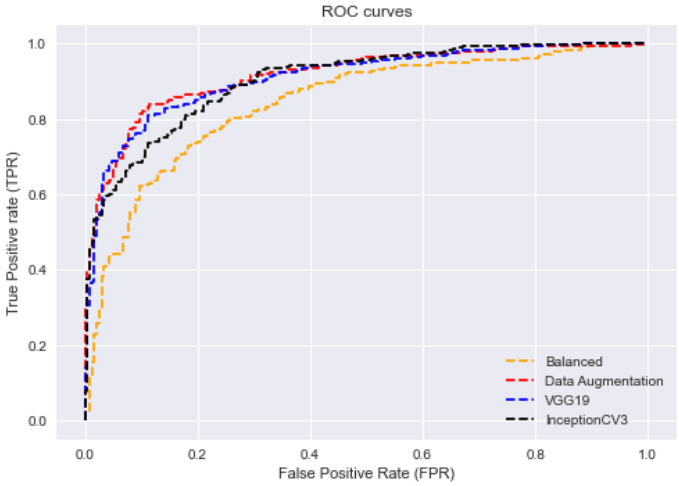


Fig. 17: ROC for mini Project 2 - 25 epochs with 200 steps per epoch.

Model	AUC
Balanced	0.8427
Data-Augmentation	0.9163
VGG19	0.9099
InceptionV3	0.9032

TABLE IV: AUC for 2st Project - 25 epochs with 131 steps per epoch.

When seeing the AUC values of TableIV, besides for the balanced, they are all higher when compared to the values of the 1st mini project on TableIII, except the k-fold CV experiment. The higher value that was gotten was from the k-fold Cross Validation in mini project 1.

B. Confusion Matrix

For the Confusion Matrix, the generic representation is illustrated on TableV. This matrix give us the True Positives, TP, True Negatives, TN, False Positives, FP, and False Negative, FN, metrics.

	True 0	True 1
Pred 0	TP	FN
Pred 1	FP	TN

TABLE V: Confusion Matrix.

For the first mini project, the confusion matrix obtained for the balanced data is represented on TableVI, for the unbalanced on TableVII and for the k-fold CV experiment on TableVIII.

	True 0	True 1
Pred 0	187	47
Pred 1	62	328

TABLE VI: Confusion Matrix of mini project 1 - Balanced.

	True 0	True 1
Pred 0	2	232
Pred 1	0	390

TABLE VII: Confusion Matrix of mini project 1 - Unbalanced.

	True 0	True 1
Pred 0	155	79
Pred 1	25	365

TABLE VIII: Confusion Matrix of mini project 1 - k-fold Cross Validation.

As we can see, there is a big difference between the confusion matrices from mini project 1. It is possible to see that the confusion matrix for the unbalanced data gets super low values for TP and FP which helps us conclude that working with unbalanced data is not a good idea and can be, in other metrics, misleading. As we got more Pneumonia cases than Normal cases this is the behaviour expected for an unbalanced approach, if it was the other way around it would be except to get super low values for FP and TN. When comparing the balanced approach with the k-fold CV one we see that we got more correct predicted values for the latter, which leads to better performance from the k-fold CV experiment, as seen before.

For the second mini project, for each experiment, the confusion matrix is represented, respectively, in the TablesIX, X, XI and XII.

	True 0	True 1
Pred 0	174	60
Pred 1	80	310

TABLE IX: Confusion Matrix of mini project 2 - Balanced

	True 0	True 1
Pred 0	172	62
Pred 1	44	346

TABLE X: Confusion Matrix of mini project 2 - Data-Augmentation.

	True 0	True 1
Pred 0	171	63
Pred 1	43	347

TABLE XI: Confusion Matrix of mini project 2 - VGG19.

From the two mini projects, we can conclude that our model is working well when working with balanced data as the

	True 0	True 1
Pred 0	150	84
Pred 1	24	366

TABLE XII: Confusion Matrix of mini project 2 - InceptionV3.

number of TP and TN is bigger than the number of FP and FN. It is worth to mention that the number of TN is higher than the number of TP but, it is expected as there are more Pneumonia (1) cases in the test than Normal (0) cases.

C. Comparing with other Articles

We can also compare our results with other people works, per example, [11], where the authors got the following results.

Model	Precision(%)	Recall(%)	F1-Score(%)
Desenet 169	78.19	78.65	78.42
VGG16	79.88	82.17	81.01
VGG19	83.14	83.05	83.09

TABLE XIII: Results obtained from [11]

For both the mini projects we were able to get better results (besides for the unbalanced experiment) using approximately the same number of epochs.

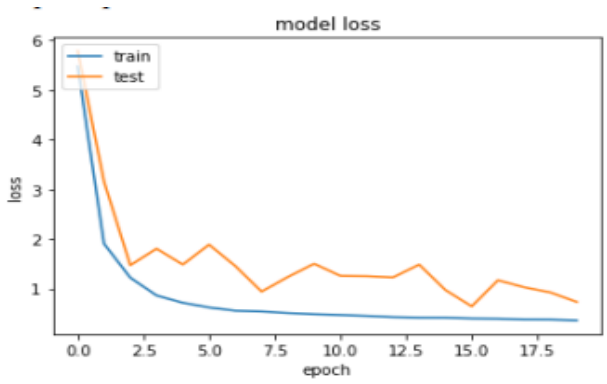


Fig. 18: VGG19 model loss from [11]

Comparing the VGG19 results in [11] with our implementation of the VGG19 model we see that our model got better results although in our implementation the model didn't full converge, Fig,12, but you can see in Fig.18 has converge. This difference of convergence has to do with the learning rate used. As we use a smaller learning rate, it's expected for the model to take more to converge. As for the difference in the % of F1-Score this was probably due to the number steps per of epochs used, although it wasn't possible to determine how many steps the authors use in this article.

VI. Conclusion & Future Work

With this project we were able to conclude that using unbalanced data is not the best approach when using a machine learning model, being better to use one of the various balancing methods. It's also possible to conclude that is a good method to use Data Augmentation and/or k-fold Cross Validation to help us improve our results, as well as working with an higher number of steps per epoch, keeping in mind that the bigger this number is the more time the model will take to fully execute.

As for future work, this could focus on raising the number of epochs, for the model to fully converge, and raising the number of steps per epoch for the models already developed as well as exploring other CNN models and pre-defined architectures. Another good proposal for future work would be implementation of k-fold Cross Validation for all models here presented, especially the InceptionV3 as this is excepted to be the best model. Finally, another good subject for future work will be studding the difference the number of layers causes in a model, i.e, if a higher number of layers will always lead to better results or if there is a point where raising the number of layers will only make the results worst.

References

[1] Luqman Ali et al. "Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures". In: *Sensors* 21 (Mar. 2021), p. 1688. DOI: 10.3390/s21051688.

[2] Jason Brownlee. *A gentle introduction to k-fold cross-validation*. Aug. 2020. URL: https://machinelearningmastery.com/k-fold-cross-validation/.

[3] Jason Brownlee. *How to Configure the Learning Rate When Training Deep Learning Neural Networks*. URL: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

[4] Mohamed Elgendi et al. "The Effectiveness of Image Augmentation in Deep Learning Networks for Detecting COVID-19: A Geometric Transformation Perspective". In: *Frontiers in Medicine* 8 (2021). ISSN: 2296-858X. DOI: 10.3389/fmed.2021.629134. URL: https://www.frontiersin.org/articles/10.3389/fmed.2021.629134.

[5] Daniel Kermany. *Labeled optical coherence tomography (OCT) and Chest X-Ray images for classification*. 2018.

[6] Sotiris Kotsiantis, D. Kanellopoulos, and P. Pintelas. "Handling imbalanced datasets: A review". In: *GESTS International Transactions on Computer Science and Engineering* 30 (Nov. 2005), pp. 25–36.

[7] Rafid Mostafiz et al. "machine learning & knowledge extraction Focal Liver Lesion Detection in Ultrasound Image Using Deep Feature Fusions and Super Resolution". In: *Machine Learning and Knowledge Extraction* 2 (July 2020). DOI: 10.3390/make2030010.

[8] Abhinav Sagar. *Deep learning for detecting pneumonia from X-ray images*. Nov. 2019. URL: https://towardsdatascience.com/deep-learning-for-detecting-pneumonia-from-x-ray-images-fc9a3d9fdba8.

[9] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[10] Pranshu Sharma. *Basic introduction to convolutional neural network in Deep Learning*. Mar. 2022. URL: https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/.

[11] Han Trong Thanh, Pham Huong Yen, and Trinh Bich Ngoc. "Pneumonia Classification in X-ray Images Using Artificial Intelligence Technology". In: *2020 Applying New Technology in Green Buildings (ATiGB)*. 2021, pp. 25–30. DOI: 10.1109/ATiGB50996.2021.9423017.

Appendix

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 180, 180, 3)]	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
conv2d_1 (Conv2D)	(None, 180, 180, 16)	2320
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
separable_conv2d (SeparableConv2D)	(None, 90, 90, 32)	688
separable_conv2d_1 (SeparableConv2D)	(None, 90, 90, 32)	1344
batch_normalization (BatchNormalization)	(None, 90, 90, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
separable_conv2d_2 (SeparableConv2D)	(None, 45, 45, 64)	2400
separable_conv2d_3 (SeparableConv2D)	(None, 45, 45, 64)	4736
batch_normalization_1 (BatchNormalization)	(None, 45, 45, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
separable_conv2d_4 (SeparableConv2D)	(None, 22, 22, 128)	8896
separable_conv2d_5 (SeparableConv2D)	(None, 22, 22, 128)	17664
batch_normalization_2 (BatchNormalization)	(None, 22, 22, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 11, 11, 128)	0
dropout (Dropout)	(None, 11, 11, 128)	0
separable_conv2d_6 (SeparableConv2D)	(None, 11, 11, 256)	34176
separable_conv2d_7 (SeparableConv2D)	(None, 11, 11, 256)	68096
batch_normalization_3 (BatchNormalization)	(None, 11, 11, 256)	1024
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 256)	0
dropout_1 (Dropout)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 512)	3277312
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Total params: 3,493,985
Trainable params: 3,493,025
Non-trainable params: 960

Fig. 19: CNN model summary from [8]

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 5, 5, 512)         20024384

 flatten (Flatten)           (None, 12800)             0

 dense (Dense)               (None, 1)                 12801

=================================================================
Total params: 20,037,185
Trainable params: 12,801
Non-trainable params: 20,024,384
_________________________________________________________________
```

Fig. 20: VGG19 model summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 4, 4, 2048)        21802784

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 1)                 32769

=================================================================
Total params: 21,835,553
Trainable params: 32,769
Non-trainable params: 21,802,784
_________________________________________________________________
```

Fig. 21: InceptionV3 model summary

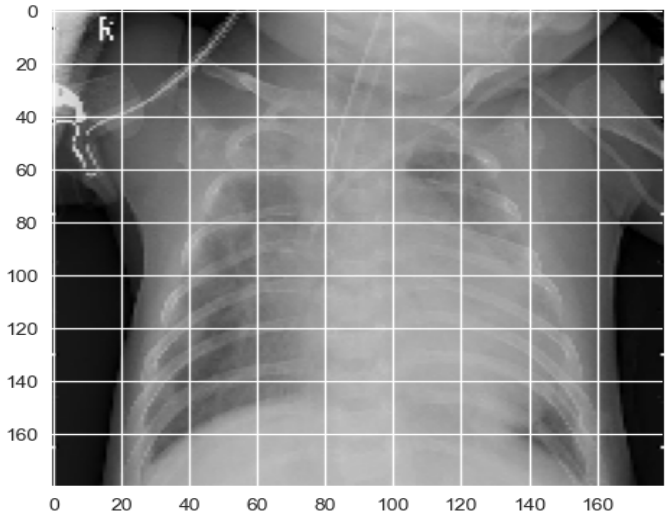


Fig. 22: Pneumonia Chest X-ray where all the models predicted correctly

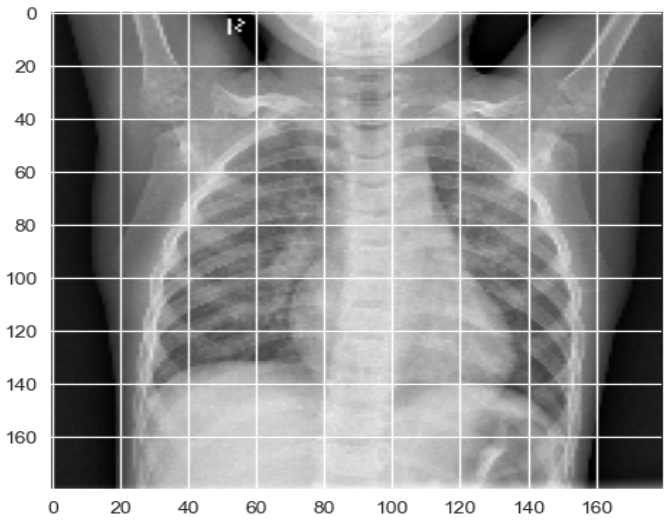


Fig. 23: Normal Chest X-ray where all the models predicted correctly

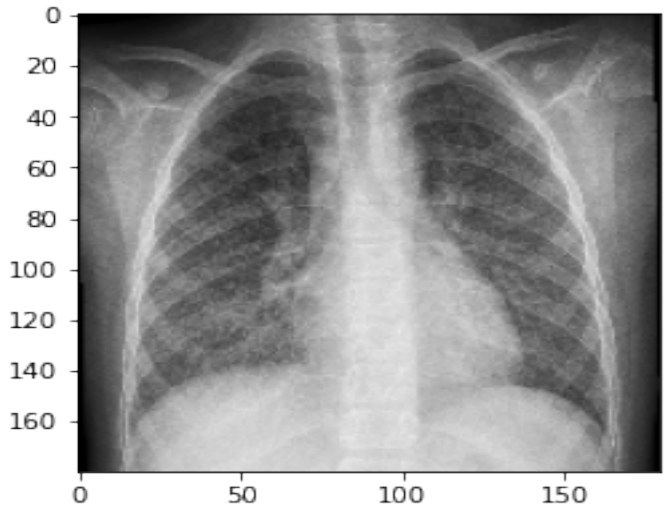


Fig. 24: Pneumonia Chest X-ray where the unbalanced model predicted wrong and all the other models predicted correctly