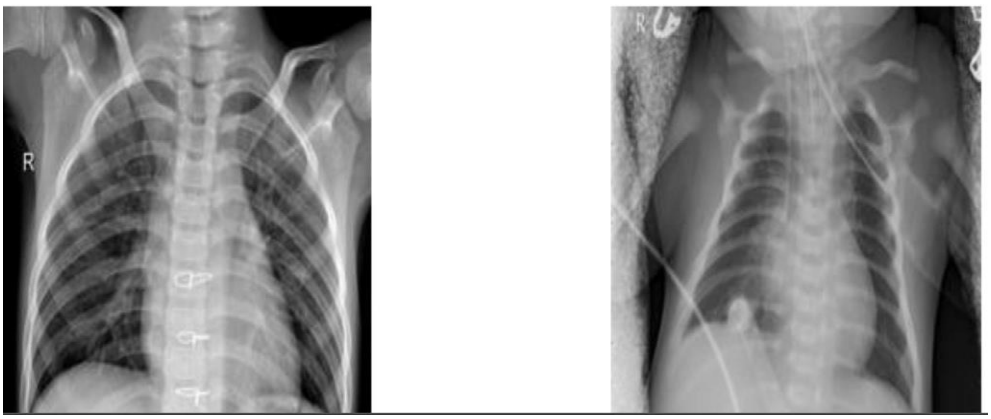
Deep Learning Project

Pneumonia Classification

Vida.zahedi@studion.unibo.it

Sample Images - Class: PNEUMONIA



Sample Images - Class: NORMAL

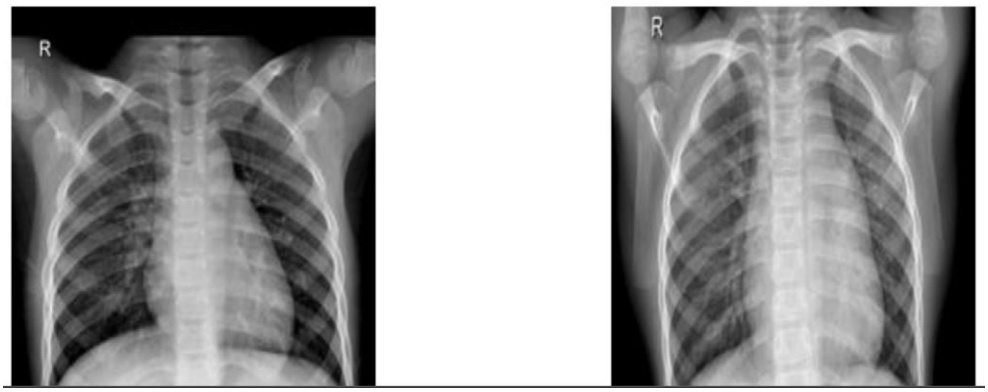


Table of Contents

Introduction

Pneumonia classification refers to the process of categorizing different types of pneumonia based on various factors, such as the underlying causes. In this dataset there are 2 types of images normal and pneumonia.

And we use EfficientNet and MobileNet Modesl models for classifying them , EfficientNet and MobileNet are both popular deep learning models that have been successfully used for image classification tasks, including pneumonia classification.

Problem

The pneumonia classification problem using the "Chest X-Ray Images (Pneumonia)" dataset from Kaggle involves the classification of chest X-ray images into two categories: normal and pneumonia. The dataset contains X-ray images of pediatric and adult patients

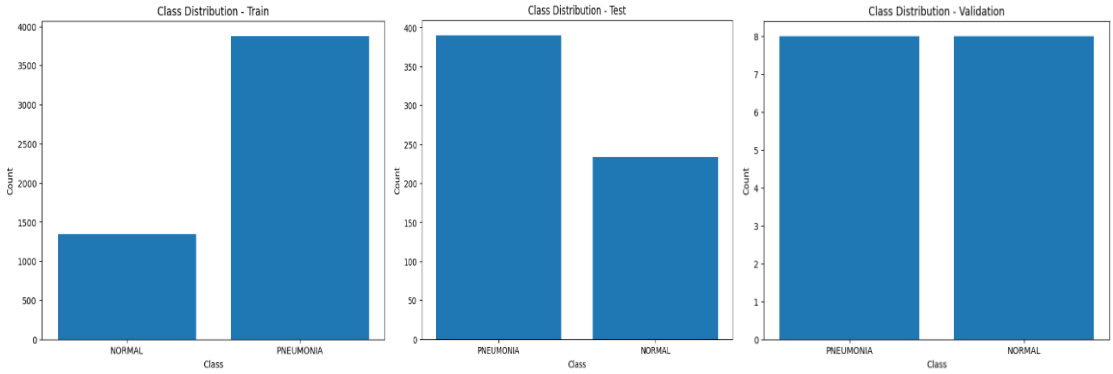
with pneumonia, as well as images of healthy individuals.

The goal of this classification task is to develop deep learning model that can accurately differentiate between normal and pneumonia cases based on the X-ray images. This problem is important for the early and accurate diagnosis of pneumonia, which can aid in timely treatment and management of the condition.

Data Set

The "Chest X-Ray Images (Pneumonia)" dataset, available on Kaggle, is a widely used dataset for pneumonia classification tasks, The dataset was created by Paul Mooney and is sourced from the National Institutes of Health (NIH), USA. The dataset contains a collection of chest X-ray images in JPEG

format, it consists of two main categories: "normal" and "pneumonia ,Each image is typically stored as a separate JPEG file with a unique filename. Below we show the class distribution for train , test and validation sets.



Preprocessing

preprocessing of data prior to training is an important task, which should be done carefully, this project also has performed many preprocessing steps to further improve the model performance. Such as removing any corrupted images or irrelevant files, resizing all images in each class folder to a target size in our case

(224,224), normalizing the pixel values of the images to scale them between 0 and 1, and fine-tuning as well.

Which we will discuss in more detail later.

Approaches

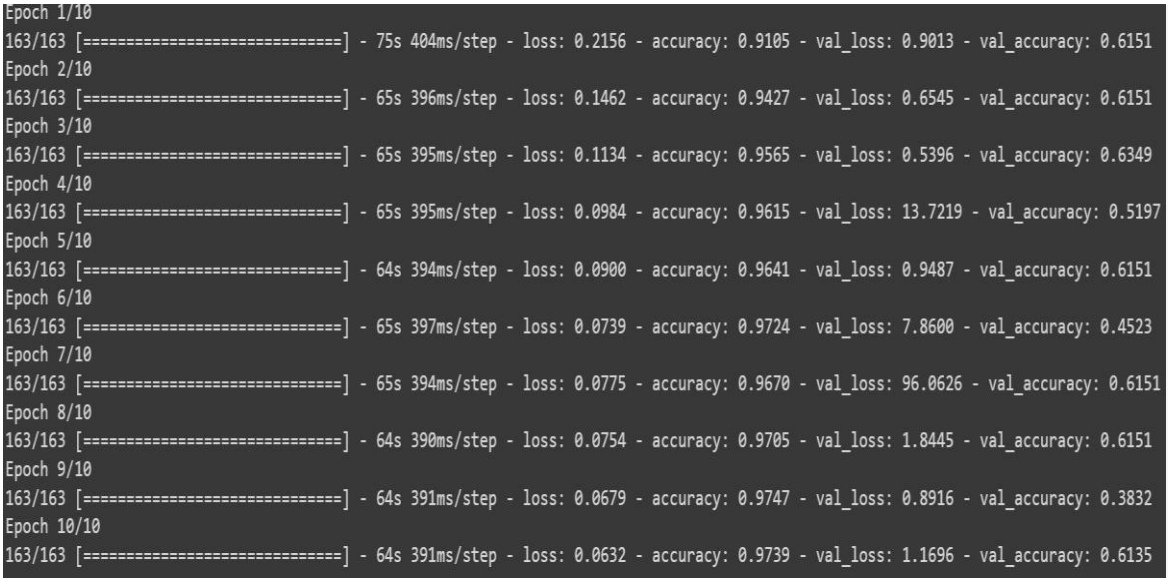
1.EfficientNet

EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a *compound coefficient*. Unlike conventional practice that arbitrarily scales these factors. Here we have several modifications of this model and their respective performance evaluation.

`train_model`

In this version we first, Create two ImageDataGenerator instances: train_datagen for data augmentation during training and test_datagen for

normalization during testing, As there was a huge class imbalance in the training we Calculate class weights to address the class imbalance in the training data. Class weights will help the model give more importance to the underrepresented class during the training phase, after this, we Load the pre-trained EfficientNetB0 model (without the top classification layers) that has been pre-trained on the ImageNet dataset. This base model serves as the feature extractor then we add Add a global average pooling layer and a dense layer with a sigmoid activation for binary classification (normal or pneumonia), and finally Compile the model using binary cross-entropy as the loss function, the Adam optimizer, and accuracy as the evaluation metric. Here are the results of this model. The trainng was done for 10epochs.



Based on the results, it seems that The training accuracy is consistently increasing across epochs, indicating that the model is learning from the training data and improving its predictions on the training set,

The validation accuracy, on the other hand, is fluctuating. This fluctuation may be due to the class imbalance in the dataset, which affects the model's ability to generalize to the validation set. here is a significant gap between the training accuracy and validation accuracy in some epochs (e.g., Epoch 4 and Epoch 6). This suggests that the model might be overfitting to the training data, resulting in poorer

performance on the unseen validation set. In the next version, we tried to address some of these issues to further improve the model performance.

train_model1

In this version, we added a fine-tuning step after freezing some layers in the base model. This is a common technique to improve the model's performance by allowing it to fine-tune on the specific dataset, Unfreezes the last few layers of the base model (adjustable with unfreeze_layers). Then we compile the model with a smaller learning rate for fine-tuning. By freezing the majority of the base model layers in the beginning and then unfreezing and fine-tuning only the last few layers, you can allow the model to adapt to the specifics of the dataset while retaining the pre-trained knowledge from ImageNet. Here are the results of this version.

```
Epoch 1/10
163/163 [==============================] - 70s 383ms/step - loss: 0.6968 - accuracy: 0.5606 - val_loss: 0.6967 - val_accuracy: 0.3849
Epoch 2/10
163/163 [==============================] - 60s 369ms/step - loss: 0.6935 - accuracy: 0.3292 - val_loss: 0.6914 - val_accuracy: 0.6151
Epoch 3/10
163/163 [==============================] - 60s 370ms/step - loss: 0.6933 - accuracy: 0.6451 - val_loss: 0.6937 - val_accuracy: 0.3849
Epoch 4/10
163/163 [==============================] - 60s 368ms/step - loss: 0.6932 - accuracy: 0.4810 - val_loss: 0.6930 - val_accuracy: 0.6151
Epoch 5/10
163/163 [==============================] - 60s 370ms/step - loss: 0.6932 - accuracy: 0.6463 - val_loss: 0.6930 - val_accuracy: 0.6151
Epoch 6/10
163/163 [==============================] - 60s 368ms/step - loss: 0.6932 - accuracy: 0.5596 - val_loss: 0.6932 - val_accuracy: 0.3849
Epoch 7/10
163/163 [==============================] - 60s 370ms/step - loss: 0.6932 - accuracy: 0.4189 - val_loss: 0.6932 - val_accuracy: 0.3849
Epoch 8/10
163/163 [==============================] - 61s 373ms/step - loss: 0.6932 - accuracy: 0.3326 - val_loss: 0.6933 - val_accuracy: 0.3849
Epoch 9/10
163/163 [==============================] - 61s 373ms/step - loss: 0.6931 - accuracy: 0.2598 - val_loss: 0.6933 - val_accuracy: 0.3849
Epoch 10/10
163/163 [==============================] - 60s 367ms/step - loss: 0.6932 - accuracy: 0.2786 - val_loss: 0.6933 - val_accuracy: 0.3849
```

the updated training process didn't produce the desired improvements. The model's accuracy and loss metrics show that the model is not learning effectively, and the validation accuracy remains stuck at around 38-39%.

Train_model2

In this version we increased the number of layers to be unfrozen for fine-tuning from 100 to 150 layers. By unfreezing more layers, the model can learn from the dataset in a more customized manner, while still leveraging the pre-trained knowledge from ImageNet, we also decreased the learning rate for the optimizer during fine-tuning. A smaller learning rate during fine-tuning

allows for more precise updates to the weights, which can be beneficial when training the later layers of the model.

Also, I added an early stopping callback to monitor the validation performance and stop training when the validation loss stops improving. This helps prevent overfitting and ensures the model is not trained unnecessarily for the specified number of epochs, and finally included a check to ensure that the Batch-Normalization layers in the unfrozen layers are not set to trainable. This can help avoid issues with batch statistics during fine-tuning phase. Here are the results of the model.

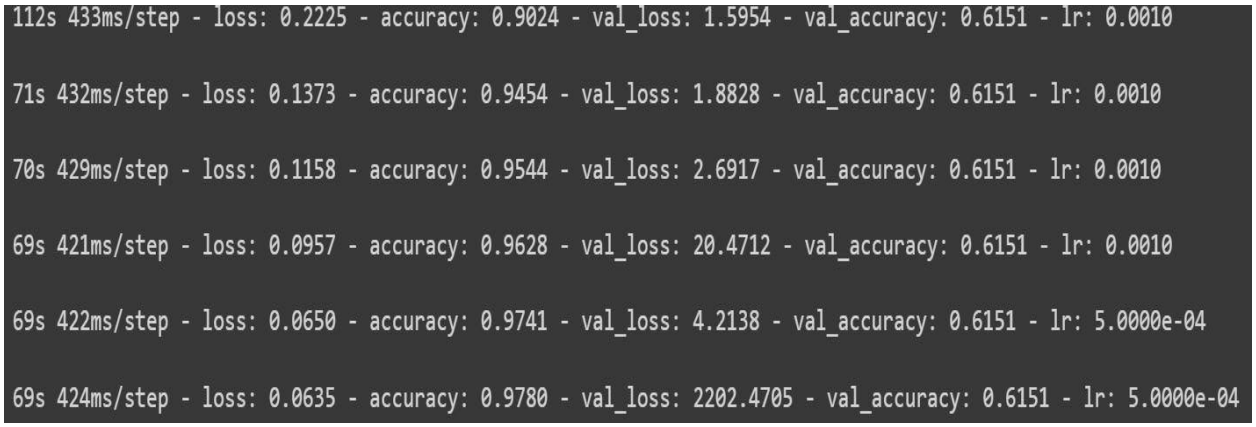
```
Found 624 images belonging to 2 classes.
Epoch 1/20
163/163 [==============================] - 76s 408ms/step - loss: 0.2473 - accuracy: 0.8992 - val_loss: 0.6759 - val_accuracy: 0.6217
Epoch 2/20
163/163 [==============================] - 65s 395ms/step - loss: 0.1399 - accuracy: 0.9454 - val_loss: 0.6669 - val_accuracy: 0.6151
Epoch 3/20
163/163 [==============================] - 64s 390ms/step - loss: 0.1224 - accuracy: 0.9553 - val_loss: 0.7094 - val_accuracy: 0.6151
Epoch 4/20
163/163 [==============================] - 64s 390ms/step - loss: 0.0982 - accuracy: 0.9615 - val_loss: 0.9043 - val_accuracy: 0.6151
Epoch 5/20
163/163 [==============================] - 63s 387ms/step - loss: 0.0856 - accuracy: 0.9666 - val_loss: 12.1659 - val_accuracy: 0.6168
```

The results indicates that these changes improved the model, however there is still room for improvement.

train_model_improved

In this version increased the number of epochs to 20 in the initial training phase, I also added a learning rate scheduler that reduces the learning rate gradually during training. For addressing class imbalance I've added additional image transformations to the data augmentation process, making it more aggressive to increase data diversity increased the number of layers to be unfrozen during fine-tuning to 200 layers and finally

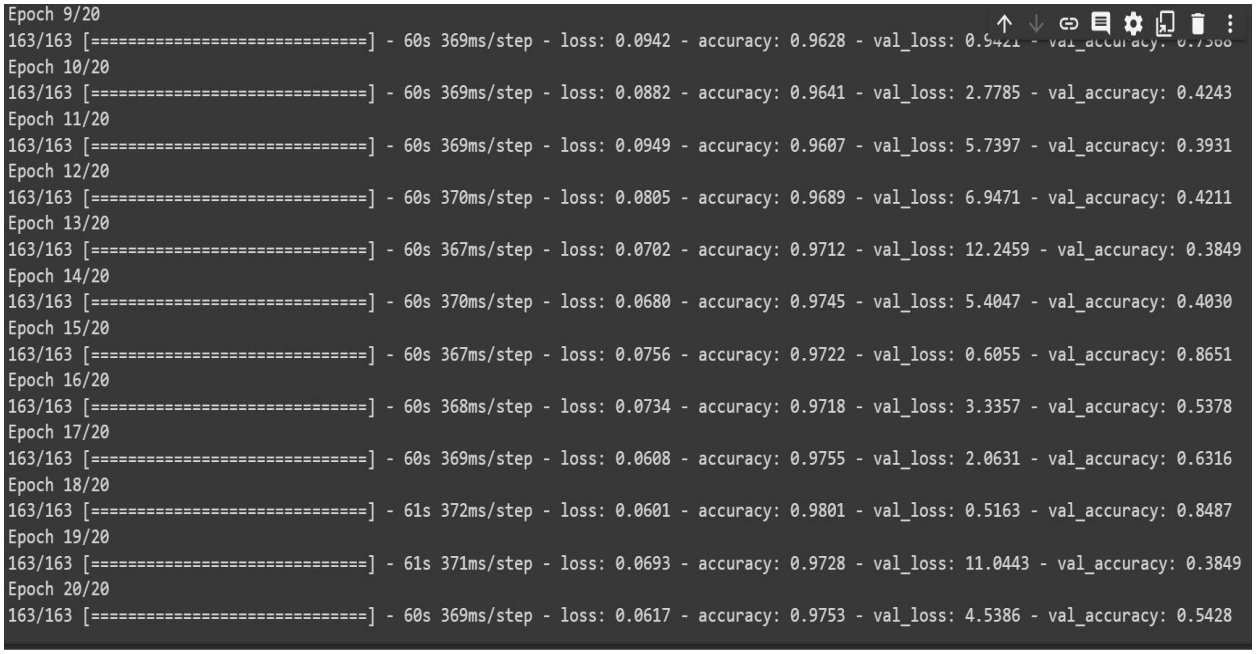
switched the optimizer to RMSprop, which can work well in some cases for fine-tuning pre-trained models



This final version has significant improvements compared to previous version.

MobileNet

MobileNet is a type of convolutional neural network designed for mobile and embedded vision applications. They are based on a streamlined architecture that uses depth-wise separable convolutions to build lightweight deep neural networks that can have low latency for mobile and embedded devices. In this part, we used MobileNetV2 pre-trained model with similar architecture to the previous model only changed the pre-trained from EfficientNetB0 to MobileNetV2. Bellow there is the result of this model.



Comparison and Conclusion

Based on the results, it seems that the MobileNet model is performing better than the EfficientNet models. The reasons for this difference in performance could be attributed to several factors such as MobileNetV2 is designed to be lightweight and efficient, making it suitable for mobile and resource-constrained environments. On the other hand, EfficientNet is a more complex and deeper architecture, which may require more data and computational resources to achieve its full potential, In

summary, the MobileNet model is currently performing better, but both models could benefit from further experimentation and tuning to achieve improved performance.