Report Chest X-Ray Images (Pneumonia)

Noms des étudiants

- Bouhnine Salaheddine (195159)
- Saïdi Rayane (195048)

Nom du professeur associé

- Ken Hasselman (HSL)

Contents

1 Introduction

Pneumonia is a prevalent infectious disease that affects millions of people worldwide, with particularly high mortality rates in vulnerable populations such as children and the elderly. Rapid and accurate detection of pneumonia is essential to provide timely and effective treatment, potentially reducing its impact on global health.

Chest X-ray imaging is a common and valuable diagnostic tool for detecting pneumonia. However, the interpretation of these images relies on trained radiologists and can be both time-consuming and subjective, with diagnoses potentially varying between practitioners. In addition, in many areas of the world, access to trained radiologists is limited.

In recent years, advances in artificial intelligence (AI) and, more specifically, convolutional neural networks (CNNs), have shown significant potential in medical imaging, providing a means to automate and standardize the analysis process. CNNs are a class of deep learning algorithms that have proven particularly effective for image analysis tasks, making them an excellent tool for interpreting chest X-ray images.

In this project, we aim to leverage the power of CNNs for the automated detection of pneumonia from chest X-ray images. We develop a CNN model using TensorFlow and Keras, popular and powerful libraries for deep learning in Python. Our objective is to train a model that can accurately detect the presence of pneumonia, offering a tool that can aid in the diagnostic process, enhance the speed of diagnosis, and potentially improve patient outcomes.

2 Dataset

The dataset used in this project is the Chest X-ray Images (Pneumonia) dataset available on Kaggle. The dataset is divided into training and test sets.

The dataset used in this study is a collection of chest X-ray images, specifically curated for pneumonia detection. It contains a total of 5856 X-ray images, which are labeled and categorized into two classes: "Normal" and "Pneumonia". The images are grayscale and vary in size.

The dataset is organized into a hierarchical directory structure as shown below:

```
chest_xray/
|-- test/
|   |-- NORMAL/
|   |   |-- IM-0001-0001.jpeg
|   |   |-- IM-0003-0001.jpeg
|   |   |-- ...
|   |-- PNEUMONIA/
|   |   |-- person1_virus_6.jpeg
|   |   |-- person1_virus_7.jpeg
|   |   |-- ...
|-- train/
|   |-- NORMAL/
|   |   |-- IM-0115-0001.jpeg
|   |   |-- IM-0117-0001.jpeg
|   |   |-- ...
|   |-- PNEUMONIA/
|   |   |-- person1_bacteria_1.jpeg
|   |   |-- person1_bacteria_2.jpeg
|   |   |-- ...
|-- val/
|   |-- NORMAL/
|   |   |-- NORMAL2-IM-1427-0001.jpeg
|   |   |-- NORMAL2-IM-1430-0001.jpeg
|   |   |-- ...
|   |-- PNEUMONIA/
|   |   |-- person1946_bacteria_4874.jpeg
|   |   |-- person1946_bacteria_4875.jpeg
|   |   |-- ...
```

Here is a sample of some images from the dataset :

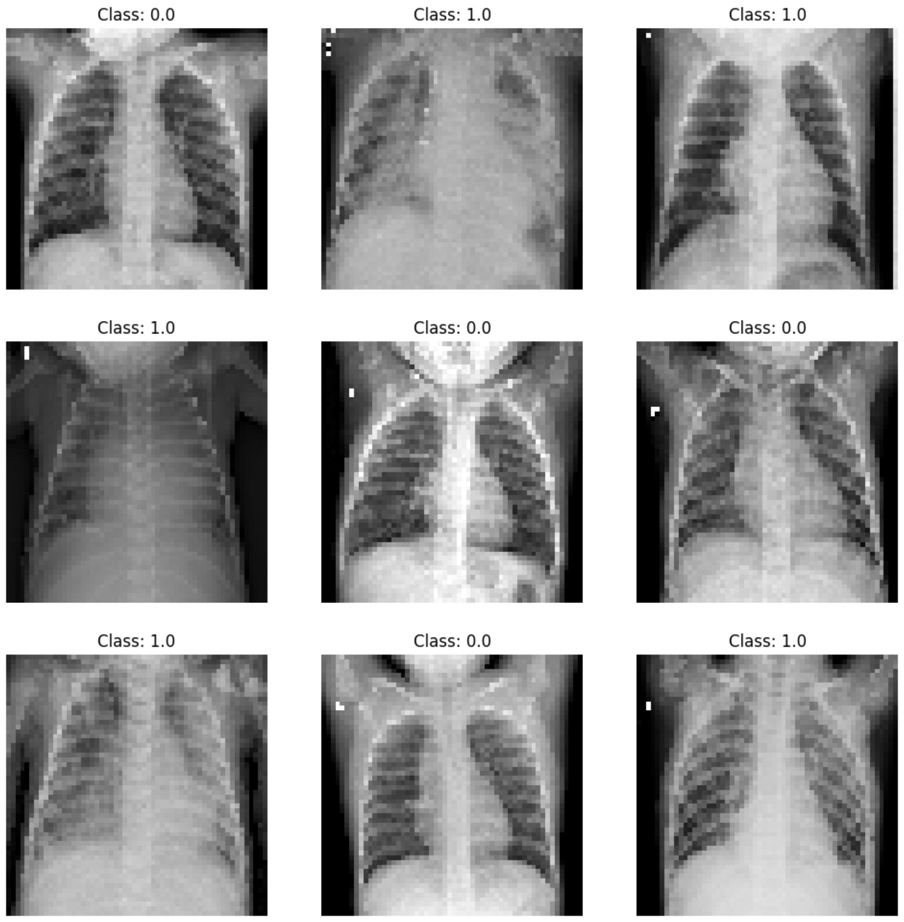

Figure 1: Some images from the dataset

The directory **chest_xray/** is the main directory, which contains three subdirectories: **test/**, **train/**, and **val/**. These directories represent the test, training, and validation sets, respectively. Each of these directories further contains two subdirectories: **NORMAL/** and **PNEUMONIA/**, denoting the class of the images inside them.

3 Data Preprocessing

In any machine learning project, one of the most crucial steps is the preprocessing of data. This step ensures that the data fed into the model is in the right form and quality, leading to better performance and results.

In this project, the ImageDataGenerator class from Keras is utilized for data loading and preprocessing. This class not only helps to efficiently load and read the data from the directory but also provides capabilities to perform on-the-fly data augmentation, an essential aspect in dealing with image datasets. However, for the scope of this project, only rescaling of the pixel values was performed using ImageDataGenerator.

The images in the dataset are in varying sizes and scales. Neural networks, however, require a fixed size for all input data points. Hence, the images were resized to 56 x 56 pixels. The **target_size** argument in the **flow_from_directory** method ensures this resizing.

Further, neural networks perform better when the input data are normalized. Normalization ensures that the pixel values of the images, which originally range from 0 to 255, get rescaled to fall in the range of 0 to 1. This is achieved by setting the **rescale** argument of ImageDataGenerator to 1./255.

Lastly, to deal with the large dataset and avoid memory errors, the concept of data batching is applied. This allows the model to train on a subset of the entire dataset at a time, making the process memory-efficient. In this project, two different batch sizes (32 and 64) to compare them and find the best between them.

Here is the code segment that performs these preprocessing tasks:

```
# Set up the ImageDataGenerator
data_gen = ImageDataGenerator(rescale=1./255)

# Set the random seed
np.random.seed(42)
tf.random.set_seed(42)

# Hyperparameters to test
batch_sizes = [32, 64]
epochs_list = [3, 5]

best_accuracy = 0
```

```
best_batch_size = 0
best_epochs = 0

for batch_size in batch_sizes:
    for epochs in epochs_list:
        # Load the datasets
        train_generator = data_gen.flow_from_directory(
            data_dir+"train/",
            target_size=(56, 56),
            batch_size=batch_size,
            class_mode='binary'
        )

        test_generator = data_gen.flow_from_directory(
            data_dir+"test/",
            target_size=(56, 56),
            batch_size=batch_size,
            class_mode='binary'
        )
```

4 Model Building and Training

In this study, we built a Convolutional Neural Network (CNN) model using Keras to classify chest X-ray images into two classes: normal and pneumonia. The model was kept intentionally simple to allow for rapid testing and iteration.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential([
    Flatten(input_shape=(56, 56, 3)),
    Dense(512, activation='relu'),
    Dense(2, activation='softmax')
])
```

The architecture of the model consists of an initial Flatten layer, which reshapes the 2D 56x56 input images into a 1D array. This is followed by a Dense layer with 512 units and ReLU activation function. The final layer is another Dense layer with two units, which corresponds to the two classes. This layer uses the softmax activation function to output a probability distribution.

To optimize the model, we employed the Adam optimizer with a learning rate of 0.001 and the cross-entropy loss function, which is suitable for binary classification problems.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

The impact of batch size and number of training epochs on model performance was also evaluated. We trained the model using two different batch sizes (32 and 64) and two different numbers of epochs (3 and 5). For each combination, we instantiated a new model and trained it from scratch on the training dataset.

```
history = model.fit(train_data_gen,
                    steps_per_epoch=total_train // batch_size,
                    epochs=epochs,
                    validation_data=val_data_gen,
                    validation_steps=total_val // batch_size)
```

The ”history” object records training loss and accuracy values at the end of each epoch. This allows us to track the progress of our model during training and adjust hyperparameters as needed.

5 Results and discussion

We used this code to display each iteration’s result:

```
# Evaluate the model on the test set
        loss, accuracy = model.evaluate(test_generator)
        print(f"Accuracy on the test set with batch size {batch_size} and
            {epochs} epochs: {accuracy:.2%}")

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_batch_size = batch_size
            best_epochs = epochs
```

```
print(f"\nBest accuracy on the test set is {best_accuracy:.2%} with batch size
    {best_batch_size} and {best_epochs} epochs.")
```

We observed varied performance of the model across different batch sizes and epochs. The model performance for each combination of hyperparameters is tabulated below:

Batch size	Epochs	Validation Accuracy
32	3	84.29%
32	5	77.40%
64	3	71.96%
64	5	72.60%

Table 1: Model performance with different batch sizes and epochs.

The model achieved the highest validation accuracy of 84.29% with a batch size of 32 and after 3 epochs of training.

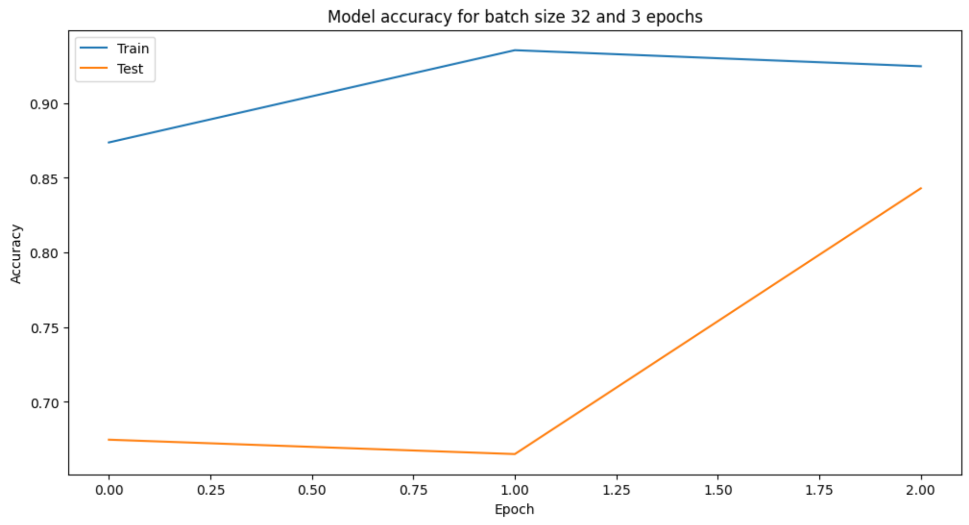

Figure 2: Training and validation accuracy

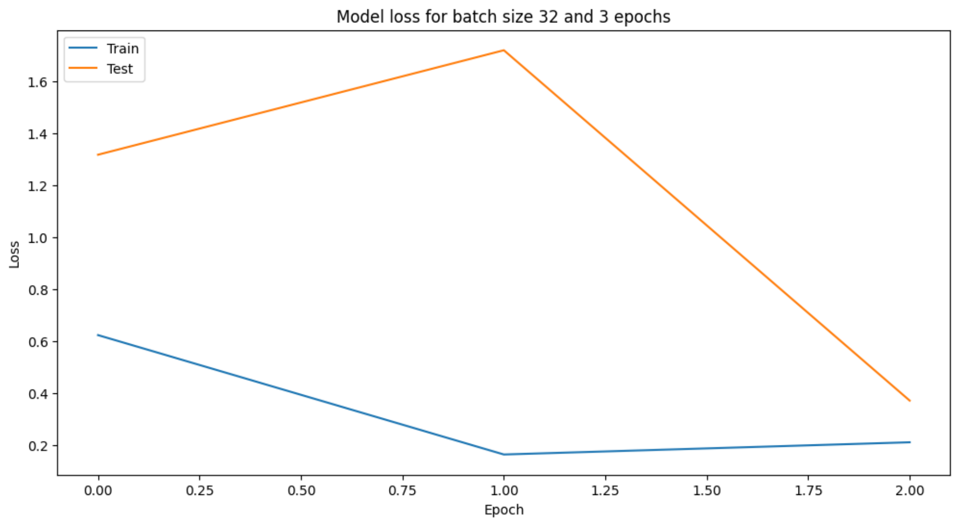

Figure 3: Training and validation loss

The results indicate that a smaller batch size and fewer epochs yield better model performance for this specific dataset and model architecture. This might be attributed to the fact that smaller batch sizes and fewer epochs can provide a regularizing effect, helping to mitigate overfitting.

6 Conclusion

In this study, we have developed and evaluated a basic Convolutional Neural Network model for detecting pneumonia in chest X-ray images using Keras. The model was trained and validated using different batch sizes and epochs to determine the optimal parameters. The best results were achieved with a batch size of 32 and 3 training epochs, yielding a validation accuracy of 84.29%.

Our findings suggest that smaller batch sizes and fewer training epochs may be beneficial for this type of task, potentially due to the regularizing effect they provide, which can help mitigate overfitting.

However, it is important to note that these results may not generalize to other datasets or model architectures. The best combination of batch size and number of epochs will depend on a variety of factors, including the complexity and size of the model, the variability of the data, and the computational resources available.

This study represents an initial step towards the development of automated diagnostic tools for pneumonia detection. Future work could include a more extensive hyperparameter search, potentially considering different model architectures, learning rates, or optimization algorithms. Additionally, implementing regularization techniques or data augmentation could further improve model performance.