DETECTION OF PNEUMONIA IN CHEST X-RAY IMAGES USING NEURAL NETWORKS

Savvas Nikolaidis, Symeon Papadopoulos
snikolaidis@ihu.edu.gr
spapadopoulos@ihu.edu.gr

ABSTRACT

This report presents a study concerning the detection of pneumonia in chest x-ray images with the use of machine learning. In particular, three different approaches are analyzed, a custom 15-layer CNN model, the VGG19 pretrained model and an ensemble model consisting of the aforementioned custom CNN model and the VGG19 and Xception pretrained models. The data that are used come from the Detect Pneumonia (Spring 2023) dataset of the respective Kaggle competition. In the current study, a tuning approach is used in order to improve the results of each model. In addition, the results of the three models are visually depicted with the use of diagrams and a comparison among the three cases is conducted. Finally, the conclusions of the study are presented.

Index Terms— CNN, pneumonia, pretrained model, classification, ensemble

1. DATA & PROBLEM DESCRIPTION

Pneumonia is an infection in the lungs, occurring either in the left or right or both lungs at the same time, affecting the alveoli, which are the small air sacs in the lungs. The various symptoms of pneumonia are dry cough, chest pain, fever, and difficulty breathing [1]. Pneumonia affects a significant portion of the global population, leading to high morbidity and mortality rates if not detected and treated promptly.

With the increasing availability of medical imaging data, the development of automated systems for pneumonia detection has gained significant attention. In this study, a machine learning approach for pneumonia detection ia proposed using chest X-ray images, leveraging the power of convolutional neural networks (CNNs), including a 15-layer CNN, a VGG19 pretrained model and an ensemble model combining the 15-layer CNN model and VGG19 and Xception pretrained models.

Initially, a dataset consisting of 4672 train and 1168 test chest X-ray images, labeled as 0 (normal), 1 (bacterial pneumonia) or 2 (viral pneumonia), is collected. Preprocessing techniques such as image resizing, normalization, and augmentation are applied to enhance the quality and diversity of the dataset. Next, a CNN-based model is used as the first approach in this three-label image classification problem. As a second approach a VGG19 model, is employed. Last but not least, an ensemble model is constructed by combining the 15-layer CNN model, the VGG19 model and another powerful CNN architecture, Xception.

In order to evaluate the performance of the showcased approaches, extensive experiments are conducted using a diverse dataset of chest X-ray images. The evaluation metrics, including training accuracy - loss and validation accuracy - loss, are calculated to assess the system's effectiveness in pneumonia detection. The experimental results demonstrate that the 15-layer CNN achieves superior performance compared to the individual VGG19 and ensemble models.

In conclusion, this report presents machine learning-based approaches for the detection of pneumonia in chest X-ray images, utilizing convolutional neural networks. Among the showcased approaches the VGG19 pretrained model demonstrates improved performance and the most promising results compared to the 15-layer CNN and the ensemble model. It is evident that the correct use of hyperparameters and fine-tuning of the models as well as the use of the most suitable model can lead to a tremendous increase in the accuracy of the results and enhance greatly pneumonia detection.

The dataset that is used, is the Detect Pneumonia (Spring 2023) dataset taken from the respective Kaggle competition. It consists of two folders, namely train_images and test_images and a labels_train.csv file which contains the class labels of the images in the form of pairs, file_name, class_id. The train_images folder contains 4672 xhest x-ray images out of which, 1227 images belong to class 0 (subject without disease - normal), 2238 images belong to class 1 (patient with bacterial pneumonia) and 1207 images belong to class 2 (patient with viral pneumonia). The test_images folder contains 1168 images that are used for testing the model.

As far as the allocation of the classes is concerned, although the majority of images are classified as having bacterial pneumonia, it is evident that there is no significant imbalance in the data given the amount of images per class. Figures 1, 2 and 3 depict chest x-ray image samples belonging to each of the three classes.

The aim of this project is the classification of a particular x-ray as having viral, bacterial or no pneumonia. So the problem is defined as a three-class classification problem where the inputs are chest x-ray images and the output is one of the three aforementioned classes (viral, bacterial or no pneumonia).

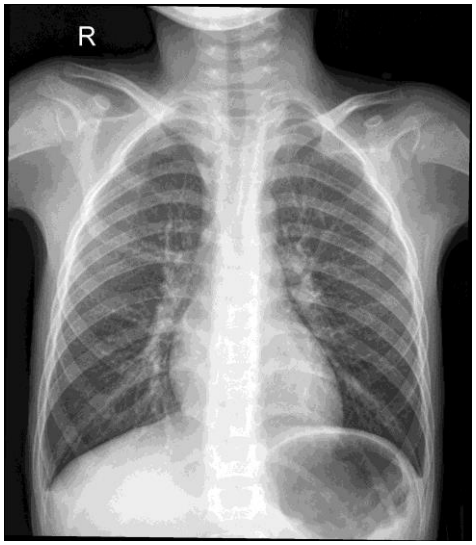

Figure 1 Sample from class 0

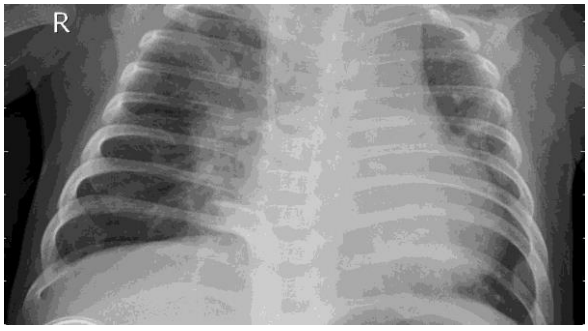

Figure 2 Sample from class 1

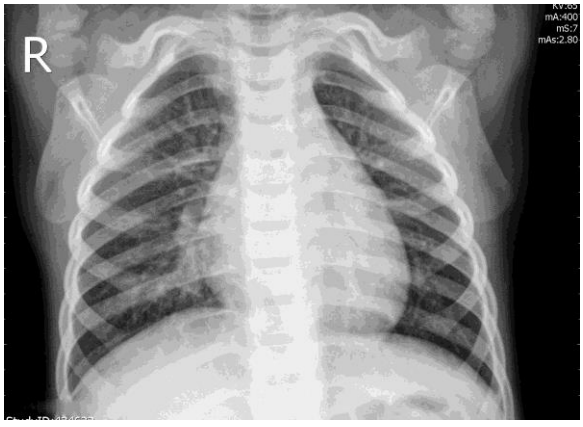

Figure 3 Sample from class 2

The methodology that is followed can be summarized in the following steps. Initially, the appropriate libraries and functions are imported. In particular, keras, tensorflow, scikitlearn, numpy, cv2 and pandas are used. Subsequently, the train and test folder paths as well as the path to the labels_train.csv file are introduced, in order to gain access to the images that are used as inputs. The next step concerns the image augmentation strategy that is used in order to avoid overfitting and limited data availability problems, since the total amount of images for training are extremely small compared to the parameters of the studied models. In addition, the images in the train_images folder are split into train and validation sets allocating 80% of the images to the train set and the rest 20% of the images to the validation set, in order to train and then evaluate the performance of the studied models. Subsequently, the model that will be trained is constructed and the parameters of the models are determined. The following step is the training of the models. Last but not least, the best model weights are saved and the model is loaded with the best weights in order to predict the classes of the images of the test_images folder and a test_prediction.csv file is generated.

In order to write and execute the code for the task, extensive use of the Google colab pro platform and the Kaggle notebook feature is made. In both cases, the code is executed with the use of GPUs in order to minimize the time needed for execution.

2. MODEL DESCRIPTION

2.1. 15-layer CNN

The first model that is studied is a custom 15-layer CNN. In specific, the model consists of 5 Conv2D layers, 5 MaxPooling2D layers, 1 Flatten layer, 4 Dense layers and 2 Dropout layers. The input images are expected to have dimensions of 224x224 pixels with 3 color channels (RGB). The architecture of the model is analyzed below. The model starts with a Conv2D layer with 64 units, each of size 3x3, and a ReLU activation function. This layer performs the initial convolution operation on the input image. Subsequently, a MaxPooling2D layer with a pool size of 2x2 is applied, which reduces the spatial dimensions of the features obtained from the previous layer. The aforementioned sequence of applying Conv2D and MaxPooling2D layers is repeated two more times, gradually increasing the number of units to 128 and then 256. This allows the model to capture more complex features as the network deepens. Furthermore, an additional Conv2D layer with 512 units and a 3x3 kernel size is applied, followed by a ReLU activation and MaxPooling2D layer. This provides further depth and complexity to the model. Similarly, another Conv2D layer with 1024 units and a 3x3 kernel size is added, followed by ReLU activation and MaxPooling2D layer. This further increases the depth and complexity of the

model. With the use of the Flatten layer the feature maps obtained from the previous layers are flattened into a 1-dimensional vector. Then, a Dense layer with 512 neurons and a ReLU activation function is added. This layer learns high-level representations from the flattened features. In addition, in order to reduce the risk of overfitting, a Dropout layer with a dropout rate of 0.5 is applied, randomly disabling half of the neurons during training. Then, another Dense layer with 256 neurons and a ReLU activation function is added followed another Dropout layer with a dropout rate of 0.5 to further prevent overfitting. Finally, a Dense layer with 3 neurons and a softmax activation function is added (since there are 3 classes for classification). This layer outputs the class probabilities for each input image.

Subsequently, the model is compiled using the compile() function. The chosen loss function is 'sparse_categorical_crossentropy' which is commonly used for multi-class classification problems when the target labels are provided as integers. The optimizer chosen is 'adam' and the model's performance will be evaluated using the 'accuracy' metric.

In order to minimize the risk of overfitting, data augmentation is applied using the ImageDataGenerator class from Keras. In particular width_shift_range, height_shift_range, shear_range, and zoom_range augmentation parameters are used in order to apply geometric transformations to the images. Moreover, fill_mode is set to 'nearest' to fill in any pixels that may be created during the augmentation process. Finally, with the preprocessing_function parameter of ImageDataGenerator a custom_augmentation is introduced that adjusts contrast of the input image using the random_contrast() function. Figures 4 and 5 show the code snippet that concerns the data augmentation strategy that is followed. An additional part of the preprocessing stage is the normalization of the pixel values to the range of [0, 1]. This is a common practice for preprocessing images to ensure numerical stability during training.

```
train_datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip = False,
    preprocessing_function=custom_augmentation,
    fill_mode='nearest')
```

Figure 3 Data augmentation code snippet

```
def custom_augmentation(np_tensor):

  def random_contrast(np_tensor):
    return tf.image.random_contrast(np_tensor, 0.5, 2)

  augmnted_tensor = random_contrast(np_tensor)
  return np.array(augmnted_tensor)
```

Figure 4 Function for adjusting contrast

The next step concerns the training of the model with the use of .fit() function. The .fit() function uses a data generator for the training data (train generator) and a data generator for the validation data. In addition, the training and validation steps are determined according to batch size. It is determined that the most efficient batch size is 64. Furthermore, the checkpoint callback is added in order to save periodically those weights of the model that increase accuracy. Finally, the number of epochs that result in the best performance is 100.

The best model weights that are saved during training are loaded and then predictions on the images of the test_images folder are performed. Finally, the results are saved in the test_predictions CSV file and compared to the known results so that the performance of the model can be assessed.

In order to increase the performance of the studied model various measures are taken. The code has been executed with several different parameters regarding the number of epochs, batch size, ImageDataGenerator parameters and class weights. In specific, a number of cases is studied using a batch size of 32 and 64 and a number of epochs either 50 or 100. It is determined that a batch size of 64 and a number of 100 epochs give the best results. As far as the ImageDataGenerator parameters are concerned, several values have been tested for each parameter and the ones that give the best results are showed in figures 4 and 5. In table 1 the values of the parameters that have been used are presented. Finally, it is determined that adding class weights does not give any advantage on the model.

Table 1 ImageDataGenerator parameters

parameter	trial value 1	trial value 2	trial value 3
rotation_range	0	10	20
width_shift_range	0.1	0.2	-
height_shift_range	0.1	0.2	-
shear_range	0.1	0.2	-
zoom_range	0.1	0.2	-
horizontal_flip	True	False	-

2.2. VGG19 pretrained model

The second model of the study is a custom Convolutional Neural Network (CNN) built upon a pre-trained VGG19 model with added dense layers and dropout layers for regularization. The model architecture is based on the

VGG19 model, pre-trained on the ImageNet dataset. The architecture includes the base of VGG19, with the top (classifier) part removed. Given that the base model is not trainable, the learned weights will not be updated during training.

After the base model, the following layers are added:

- A Flatten layer, which converts the multidimensional tensors into a one-dimensional tensor, to be used for the fully connected layers.
- A Dense layer with 512 units and ReLU (Rectified Linear Unit) activation function.
- A BatchNormalization layer that normalizes the activations of the previous layer, improving the stability of the neural network.
- A Dropout layer with a rate of 0.7, which randomly sets 70% of the input units to 0 at each update during training, helping prevent overfitting.
- An additional Dense layer with 256 units and ReLU activation function.
- Another BatchNormalization layer.
- Another Dropout layer with a rate of 0.7.
- Finally, a Dense layer with 3 units and a softmax activation function. The softmax function outputs a vector representing the probability distributions of a list of potential outcomes.

Subsequently, the model is compiled with Adam optimizer and the sparse categorical crossentropy loss function as was the case with the 15-layer custom CNN that is studied first. The Adam optimization algorithm is an extension of stochastic gradient descent, and is one of the default optimizers in deep learning development. The metrics used to evaluate the model's performance is accuracy.

Data augmentation is used to artificially increase the size of the training set. This is done by applying random image transformations to the existing images in the training set. The transformations include shearing, zooming, and shifting in width and height. By doing this, the model is exposed to more diverse examples, which can help it generalize better to unseen data. Additionally, a custom contrast adjustment function is used for the preprocessing function.

The model is trained for a maximum of 50 epochs, with the EarlyStopping callback monitoring the validation loss. If the validation loss does not improve for 20 accumulative epochs, the training process is stopped early. The training process uses a batch size of 32 to have more weight updates per epoch and lead to a more robust model and faster convergence. Since there is a minor imbalance in the dataset the 'balanced' mode is used to automatically calculate class weights. This method will inversely adjust the weights according to the number of samples in each class in the training dataset. This can help to improve the model's performance on underrepresented classes and overall performance on imbalanced datasets. The model checkpoint callback is used to save the weights of the model at the epoch with the highest validation accuracy. The model is trained using a generator, which allows for data to be loaded in batches on-the-fly and can be more memory efficient.

Furthermore, the VGG19 pre-trained model was trained using the fine-tuning approach. The process started with loading the pre-trained model, and then freezing all the layers of this pre-trained model. This means that during the initial training process, the weights of the VGG19 layers were not updated, and only the weights of the newly added layers were trained. Next, the top 12 layers of the model were unfrozen, allowing these layers to update their weights during the training process. The reason for this was to allow the model to adapt more closely to our specific task. The model was then trained for an additional 50 epochs.

After training the model, the history object is used to store the loss and accuracy of the model for each epoch, for both the training and validation sets. This can be used later to plot the learning curves of the model.

Finally, after training the model many times it is concluded that the best hyperparameters about batch size and number of epochs are 32 and 50 respectively. As far as data augmentation is concerned, the best results are produced by the same parameters used in the **15-layer CNN** model. In contrast to the **15-layer CNN** model, the 'balanced' mode method helped to increase the performance a bit. What helped the model the most to be more stable and accurate is the implementation of BatchNormalization and Dropout layers with a rate of 0.7 as it is presented in the line charts below.

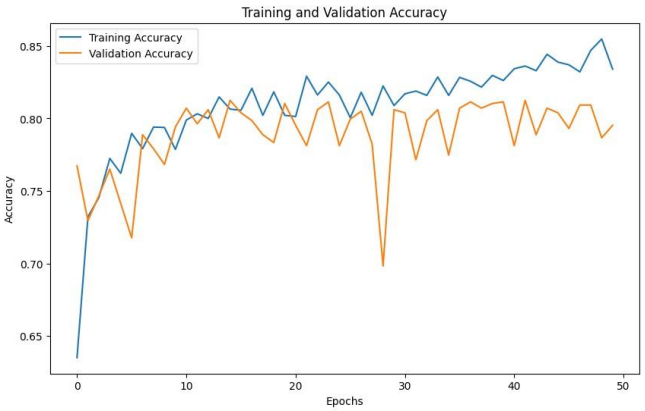

Figure 5 Plot1 without Dropout & BatchNormalization

Figure 6 Plot2 with Dropout & BatchNormalization

Figure 7 Training and validation loss of 15-layer CNN

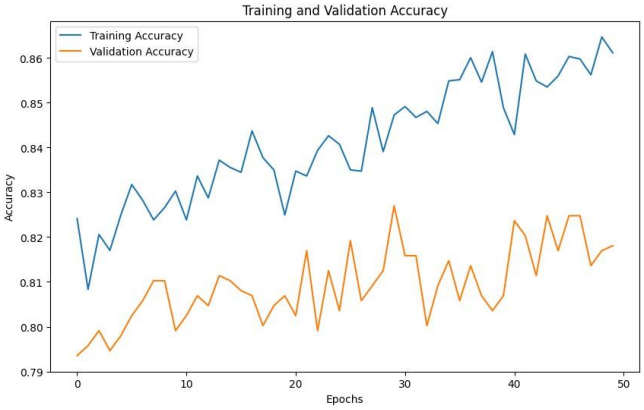

Figure 8 Training and validation accuracy of 15-layer CNN

2.3. Ensemble model

The third case that is examined is the implementation of an ensemble model that is constructed by combining the custom 15-layer CNN with two pretrained models, namely VGG19 and Xception models. In each of the pretrained the layers of the pre-trained base models are frozen to prevent their weights from being updated during training. As it is the case with the two previous studies each model is compiled with an Adam optimizer, sparse categorical cross-entropy loss function and accuracy as the evaluation metric. The number of epochs is 50 and the batch size 32. Class weights are computed and passed to the fit method to account for class imbalance in the training data. Initially each model is trained separately.

After training the individual models, the best-performing model checkpoints (best_model1.h5, best_model2.h5, best_model3.h5) are loaded and converted into separate models (model_1, model_2, model_3). An ensemble model is constructed by averaging the outputs of the individual models using the Average layer. The ensemble model is compiled and trained on the training data, similar to the individual models. Callbacks for early stopping and model checkpoint are provided during training.

Finally, the actions of saving of the best model, loading and predicting the test images and generating the .csv file are the same as the previous case studies.

3. COMPARATIVE EXPERIMENTS AND RESULTS

The custom-built 15-layer Convolutional Neural Network demonstrated a high level of stability during the training process. No signs of overfitting or underfitting were observed in the training and validation curves, which suggest that the model was able to generalize well from the training data to unseen data. The custom-built 15-layer CNN yielded an evaluation score of 83.39%.

On the other hand, the pre-trained VGG19 model showcased slightly superior performance with an evaluation score of 83.561%. It's an excellent achievement considering that the VGG19 model was originally trained on a different task (ImageNet classification), and then fine-tuned on our specific task. Stability in this context means that the performance of the model did not vary significantly showcasing only minor changes in the training data or during different phases of training. The robustness of VGG19 can be attributed to its architecture and the pre-training process it underwent. While VGG19 is less complex than our custom model in terms of the number of layers, it's still quite deep compared to many other models. This depth is what allows VGG19 to learn complex features and achieve high performance. However, its relatively less complexity compared to our custom model, and its pre-training, likely contributed to its stability, as it may be less sensitive to noise in the data and less prone to overfitting.

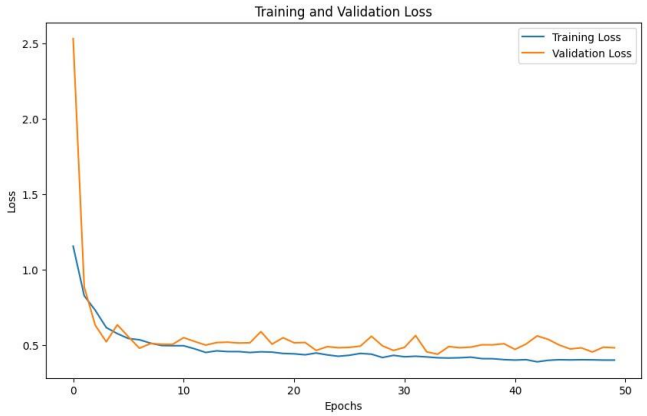

Figure 9 Training and validation loss of VGG19 model

Figure 10 Training and validation accuracy of VGG19 model

The ensemble model was built by combining the custom-built 15-layer Convolutional Neural Network (CNN) and VGG19 and Xception pre-trained models. Contrary to our expectations, the ensemble model did not outperform the individual models. Its performance was marginally worse than that of the VGG19 model and on par with the custom CNN. This is not an uncommon outcome, and could be due to various factors. One possible explanation is that the predictions of the individual models were highly correlated, meaning they made similar errors, so the ensemble model wasn't able to benefit from the diversity of the models' predictions.

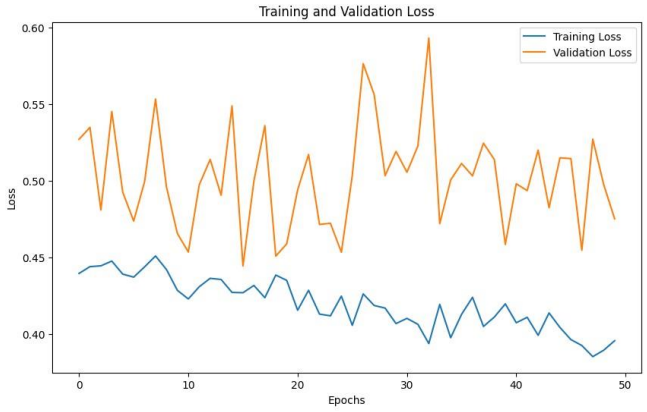

Figure 11 Training and validation loss of ensemble model

Figure 12 Training and validation accuracy of ensemble model

4. CONCLUSIONS

After thorough experimentation, interesting and quite insightful results have been found regarding the performance of different model architectures. It was observed that the VGG19 model, one of the pre-trained models, had the highest performance accuracy. Its stability during training and validation phases was notable, making it a solid choice for the stucied image classification task. Its performance accuracy was slightly higher to that of the custom-built 15-layer Convolutional Neural Network (CNN) model.

Despite the high individual performances of these models, the ensemble method, which aimed to combine the predictive power of both the custom-built and pre-trained models, did not yield superior results as expected. This outcome could be due to a variety of factors including the potential for model correlation, the complexity of the ensemble approach and the nature of our specific dataset.

In conclusion, while ensemble methods often provide a strategy to improve performance by leveraging the strengths of multiple models, it is not always the case. In this specific

scenario, VGG19 showed exceptional performance and stability, making it a highly suitable choice for the task. As always, the choice of model and strategy highly depends on the specific requirements and constraints of the problem at hand.

5. REFERENCES

[1] Torres A, Serra-Batlles J, Ferrer A, Jiménez P, Celis R, Cobo E, Rodriguez-Roisin R. Severe community-acquired pneumonia. Epidemiology and prognostic factors. Am Rev Respir Dis. 1991;144(2):312–8. doi: 10.1164/ajrccm/144.2.312. [PubMed] [CrossRef] [Google Scholar]