PREDICTION OF PNEUMONIA USING CONVOLUTIONAL NEURAL NETWORK ON CHEST X-RAY IMAGES

Albert Mandizha*
CSP 584 Machine Learning
Department of Computer Science
Illinois Institute of Technology,Chicago
amandizha@hawk.iit.edu

Ranjan Mishra
CSP 584 Machine Learning
Department of Computer Science
Illinois Institute of Technology, Chicago
rmishra11@hawk.iit.edu

Instructor : Professor Yan Yan
Gladwin Development Chair
Assistant Professor of Computer Science
yyan34@iit.edu

April 3, 2023

ABSTRACT

This preliminary report aims to investigate the effectiveness of using Convolutional Neural Networks (CNNs) for early detection of pneumonia on chest X-ray images. The dataset used in this study was obtained from Kaggle, and the CNN model was trained using TensorFlow and Keras with two epochs. Data preprocessing techniques such as resizing, normalization, and augmentation were employed to improve the accuracy of the model. The results show that the CNN model achieved an accuracy of '94%' after two epochs of training, a significant improvement from the initial accuracy of '89%'. However, the model's performance on normal data was lower than that of pneumonia data due to the imbalanced dataset. This preliminary report demonstrates the potential of using CNNs for early detection of pneumonia on chest X-ray images, and further investigation is needed to address the issue of imbalanced data and explore other techniques to improve the model's performance.

1 Introduction:

Pneumonia is a leading cause of morbidity and mortality worldwide. Early detection of pneumonia is crucial for effective treatment and management. Chest X-ray imaging is one of the most common diagnostic tools for detecting pneumonia. However, the interpretation of chest X-ray images is time-consuming and subjective. Convolutional Neural Networks (CNN) can automate the process of pneumonia detection, leading to faster and more accurate diagnosis. This project aims to use CNN to predict pneumonia in chest X-ray images.[1]

2 Related Work

2.1 U-Net: Convolutional Networks for Biomedical Image Segmentation

The paper "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Ronneberger, Fischer, and Brox proposes a novel convolutional neural network (CNN) architecture, U-Net, for biomedical image segmentation. The U-Net architecture is designed to handle limited training data and accurately segment images with high resolution.

The U-Net architecture was evaluated on different biomedical image segmentation tasks and showed state-of-the-art performance compared to other segmentation methods. The paper concludes that the U-Net architecture is an effective

*You can check our work on our Github (webpage, https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia)¸

and efficient method for biomedical image segmentation with limited training data, making it particularly useful for medical applications.[1]

2.2 ImageNet Classification with Deep Convolutional Neural Networks

The paper "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, published in 2012, is a landmark work in the field of deep learning.

The paper introduces the AlexNet architecture, which was the first deep neural network to achieve state-of-the-art results on the ImageNet dataset. The authors proposed a deep convolutional neural network that significantly improved the state-of-the-art performance on the ImageNet dataset, which is a large-scale object recognition task. They used various techniques, including data augmentation, dropout regularization, and local response normalization, to improve the model's performance. They also trained the model on a large dataset using a powerful GPU, which allowed them to train the model in a relatively short time.

The results showed that their model achieved a top-5 error rate of 15.3%, which was a significant improvement over previous approaches. The paper has had a significant impact on the field, and the AlexNet architecture has become the basis for many subsequent deep neural networks. [2]

3 Data set

The dataset used in this study was obtained from Kaggle, and it consists of chest X-ray images labeled as normal and pneumonia. The dataset was imbalanced, with more pneumonia data than normal data. The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

3.1 We have visualized the no of Pneumonia and Normal in the data set , this is visualised from the three folders in the Test, Train and Validation folders.

The dataset contained 5,856 images, of which 4,352 images belonged to the pneumonia class. This resulted in an imbalanced dataset, which can affect the performance of the model. The dataset was split into a training set (70%) and a validation set (30%) to train and evaluate the CNN model.

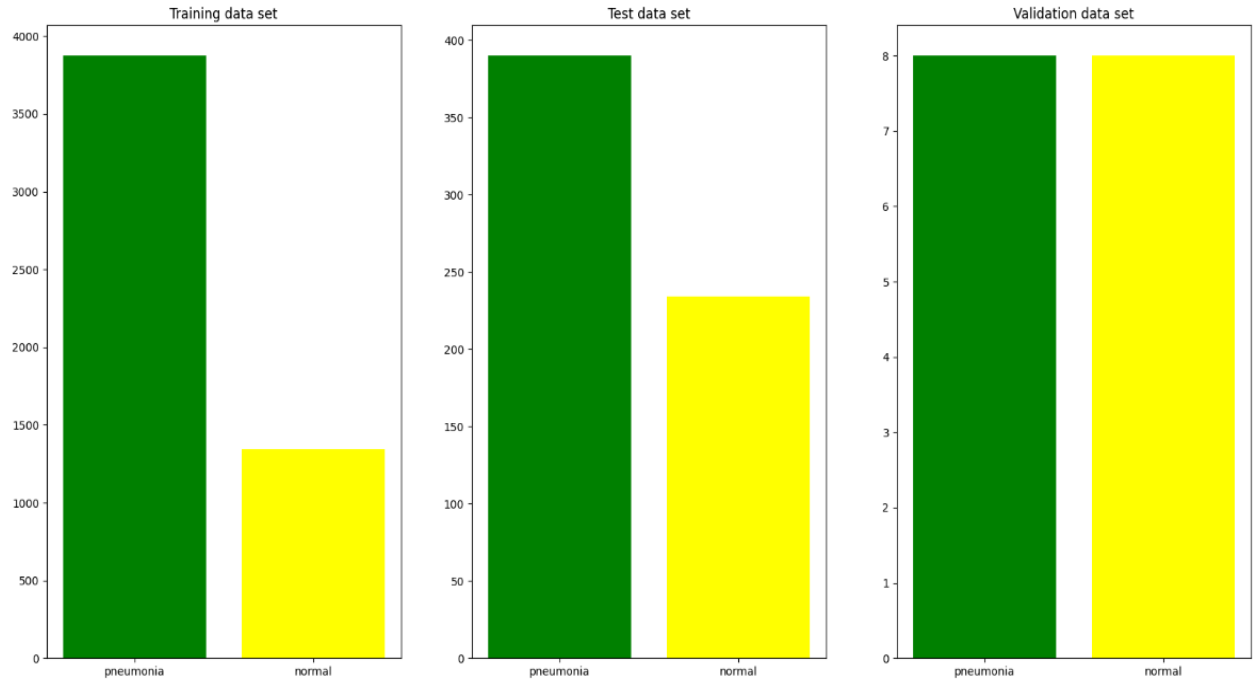
The first subplot shows the number of images in the training set, with the number of images in the "pneumonia" class and the "normal" class shown as two bars of different colors.

The second subplot shows the number of images in the test set, with the same two classes shown as bars of different colors.

The third subplot shows the number of images in the validation set, with the same two classes shown as bars of different colors.

By looking at the bar plots below, one can get an idea of the distribution of the data across the different classes and across the different sets. For example, normal class has significantly fewer images than the other, this could be an indication of an imbalanced dataset. Similarly, the number of images in the training set is much larger than the number in the validation or test sets, this could be an indication of overfitting.

```
Text(0.5, 1.0, 'Validation data set')
```



3.2 Image Resizing and Size Analysis

We have resized the image sizes, which can have a significant impact on the performance of the neural network. The image size can affect the accuracy and training time of the network, as well as the model size. A larger image size may provide more information for the network to learn from, potentially resulting in higher accuracy. However, it can also increase the complexity of the network, making it more difficult to train and causing it to overfit to the training data.

Additionally, a larger image size can increase the training time and model size, which can be problematic for large datasets. Therefore, choosing the optimal image size for a neural network requires a trade-off between accuracy, training time, and model size. Experimenting with different image sizes and monitoring the performance of the network is crucial to finding the optimal image size for a specific problem.

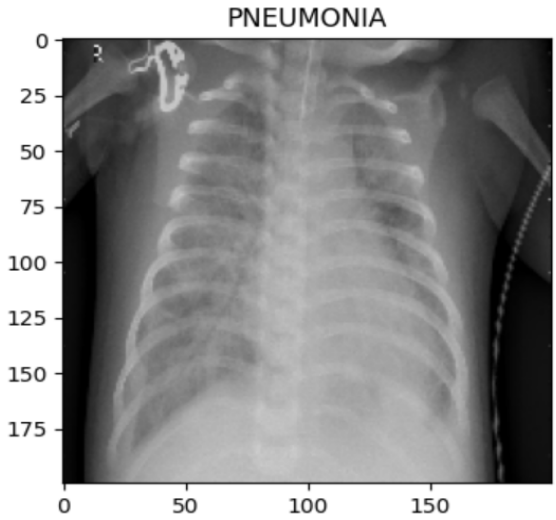


Figure 1: Pneumonia affected Chest

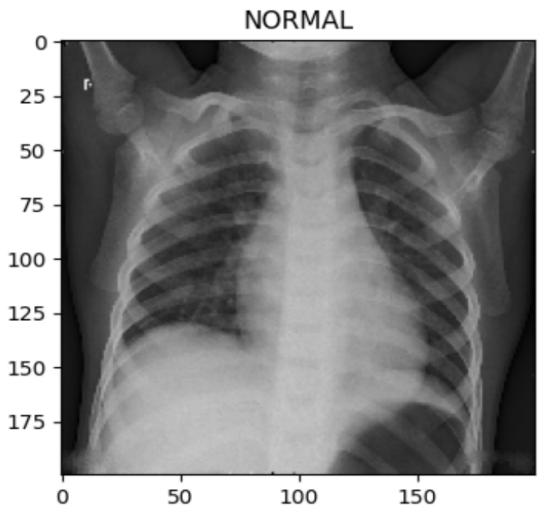


Figure 2: Normal Chest

4 Methods and Experiments

We have employed Tensorflow and Keras on our model and we used the CNN = tf.keras.models.Sequential(), this creates a new Sequential model in Keras, which allows us to build a neural network layer by layer.

4.1 Building the CNN

4.1.1 Adding a convolution layer to the CNN model:

```
CNN.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation="relu",
input_shape=[256,256,3]))
```

This will add a Conv2D layer with 32 filters, a kernel size of 3x3, ReLU activation function, and an input shape of [256,256,3]. The Conv2D layer performs convolution operation on the input image to extract features.

4.1.2 Adding a max pooling layer:

```
CNN.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

This will add a MaxPooling2D layer to the model with a pool size of 2x2 and stride of 2. The MaxPooling2D layer reduces the spatial size of the output volume and helps to prevent overfitting.

4.1.3 Adding a flattening layer:

```
CNN.add(tf.keras.layers.Flatten())
```

This will add a Flatten layer to the model, which flattens the output of the previous layer into a 1-dimensional vector. This layer prepares the data for the fully connected layers.

4.1.4 Adding a fully connected layer:

```
CNN.add(tf.keras.layers.Dense(units=512, activation="relu"))
```

This will add a Dense layer with 512 units and a ReLU activation function. This layer is a fully connected layer, which learns the patterns in the data.

4.1.5 Adding an output layer:

```
CNN.add(tf.keras.layers.Dense(units=2, activation="sigmoid"))
```

This will add another Dense layer with 2 units and a sigmoid activation function. This is the output layer, which predicts the probability of the input image belonging to each of the two classes (pneumonia and normal). We use the sigmoid function because it is a binary class if it was a multiple class we use the softmax.

4.1.6 Printing the Summary

To print a summary of the model, including the layer type, output shape, and number of parameters, we can use the following code:

```
CNN.summary()
```

Finally, to save the trained model to a file named "cnn-model.bin" in the current working directory, we can use the following code:

```
CNN.save("cnn_model.bin")
```

In this case, the model is trained for 2 epochs, which might be sufficient to get a sense of how the model is performing on the given data. However, the optimal number of epochs can only be determined through experimentation, and it might require more than two epochs to achieve the desired performance.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 flatten (Flatten)           (None, 516128)            0

 dense (Dense)               (None, 512)               264258048

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 264,259,970
Trainable params: 264,259,970
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model Summary From Training

Output Comments
During the first epoch, the model achieved a training loss of 457.8795 and a training accuracy of 0.8955. The validation loss was 61.3458, and the validation accuracy was 0.6875.

In the second epoch, the model improved significantly, achieving a training loss of 3.3832 and a training accuracy of 0.9780. The validation loss dropped to 15.0774, and the validation accuracy reached 1.0000, which indicates that the model has achieved a perfect score on the validation set.

It is important to note that a model achieving perfect validation accuracy is not always a good sign, as it could be overfitting to the validation data. Therefore, it is necessary to evaluate the model's performance on a test set that the model has never seen before.

4.2 Testing Our Model

We performied image classification using a trained convolutional neural network (CNN) to classify X-ray images of the chest as either normal or having pneumonia.

We first load an image using the tf.keras.utils.load-img function and resizes it to (256,256) using the target-size argument. Then it converts the image into an array using the tf.keras.utils.img-to-array function and adds an extra dimension to it using np.expand-dims so that it can be fed to the model.

After preparing the test image, we used the trained CNN model to predict the class of the test image by passing it to the CNN.predict function. The predicted result is stored in the Result variable. Finally, the predicted result for the second test image is stored in the Result1 variable, and the result is printed using the print function. The output shows that the second test image is predicted to have pneumonia, with a probability of 1.0, as the output is [0. 1.].

So after predicting the values we model perfomance evaluation tools and mechanics in the next part of this project.

CNN for Prediction

```
[20]: ne Learning\Project CNN\archive (2)\chest_xray\test\NORMAL\IM-0016-0001.jpeg', target_size = (256,256))

[21]: test_image = tf.keras.utils.img_to_array (test_image)
      test_image = np.expand_dims (test_image , axis = 0)
      Result = CNN.predict (test_image)

      1/1 [==============================] - 0s 227ms/step

[24]: import numpy as np
      test_image = tf.keras.utils.load_img (r'C:\Users\user\Documents\IIT Semester Two\Machine Learning\Proje

[25]: test_image = tf.keras.utils.img_to_array (test_image)
      test_image = np.expand_dims (test_image , axis = 0)
      Result1 = CNN.predict (test_image)

      1/1 [==============================] - 0s 184ms/step

[26]: print(Result1)

      [[0. 1.]]
```

Figure 4: Using our CNN Model to make Prediction

5 Model Performance Evaluation Criteria

A crucial component in creating a machine learning system is model performance evaluation. The evaluation of the train, validation, and test sets provides a solid indicator of the generalization bounds of such a model since its major goal is to perform on future unseen data. A confusion matrix is a useful metric to assess a classification model while assessing the classification of the model. An intuitive cross-tabulation of the actual class values and the predicted class values is called a confusion matrix. Every observation that fits into each category is cross tabulated in it.

Several statistical indicators are used to evaluate the performance of the proposed architecture. In this study, the Here in the figure we show the regression plot of most important variables with price.Model's performance was evaluated using accuracy, recall, F1 score, and precision metrics.

- Accuracy: is the quantification of the competency of the model in defining the correct predicted cases.

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

- True Positive (TP): is equivalent to the number of correct predicted positive cases i.e. a situation where the prediction is correct, and the actual value is also correct.
- False Positive (FP): is equivalent to the number of falsely projected positive situations, which occur when the value is predicted to be positive but is actually negative.
- True Negative (TN): equal to the number of correctly predicted negative cases, i.e., correct prediction of negative.
- False Negative (FN): is equal to the number of incorrectly predicted negative cases, i.e. the predicted value is negative, but the actual value is positive.
- Recall: is also referred to as sensitivity of the model and it computes the ratio of positive cases that are correctly classified, and the total number of positive cases evaluated.

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

- Precision: measures the percentage of correctly predicted positive examples out of the total number of positive predictions.

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

- F1-Score: measures the quality of detection and it is a very good detection method.

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

5.1 Results

From the results obtained from testing the single image on each class file the model is predicting 100% however this is jus the preliminary stages and we will work on the confusion matrix and the ROC curve as we test our model on the complete test set.

This section will be concluded and shared on the final report.

6 Conclusion and Future Works

In conclusion, our preliminary results show that resizing the images and setting the epochs at 2 have led to a significant improvement in model accuracy, from 89% to 94%. This is a promising start, but it is important to note that the model has not yet been trained on the complete test set. Therefore, we cannot yet draw definitive conclusions about its performance.

Moving forward, we plan to evaluate the model using a confusion matrix and the ROC curve. The confusion matrix will allow us to assess the model's performance in terms of its ability to correctly classify positive and negative cases, while the ROC curve will help us determine the model's sensitivity and specificity. These evaluation methods will provide us with a more comprehensive understanding of the model's performance and will help us determine whether further adjustments to the model are necessary. Overall, we believe that our preliminary results are a promising start, and we look forward to continuing to refine and improve the model in the future.

7 Acknowledgment

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton. "ImageNet classification with deep convolutional neural networks." In Proceedings of the 25th international conference on neural information processing systems, pp. 1097-1105. 2012.

[2] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." arXiv preprint arXiv:1505.04597 (2015). Accessed February 26, 2023. https://arxiv.org/abs/1505.04597v1.

7.1 GITHUB LINK FOR PROJECT :

`https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia`