Deep Learning - Final Course Project

Daniel Meir Karl (ID. 318324563), Eli Sinai (ID. 325564243)

Submitted as final project report for the DL course, BIU, 2024

1 Introduction

During the past semester, we learned about Deep Neural Networks, focusing particularly on their application in Computer Vision tasks using CNNs. This project aims to address a significant medical challenge: detecting pneumonia using chest X-ray images.

The dataset provided contains 5,863 X-ray images categorized into two main groups: Healthy and Pneumonia (further classified into bacterial and viral pneumonia, discernible from the image names).

The initial phase of the project involves creating two Deep Neural Networks (DNNs):

1. Classification of healthy/sick.
2. Classification of healthy/bacterial pneumonia/viral pneumonia.

In the second phase of the project, we aim to illustrate the process of classifying a new image by utilizing the embedding vector generated from the classification network along with KNN. We will also visualize the distinct classes using t-SNE.

In the third part of the project, we will employ anomaly detection methods, with access to only "healthy" images, our objective is to detect and identify "sick" images within the dataset.

In the final phase, we'll implement and analyze one of the explainability techniques acquired during the course for the model trained in part 1.

1.1 Related Works

We spoke with Dr. Adam Chen, a pediatric specialist from Schneider Hospital. He emphasized that to accurately diagnose pneumonia, several key factors are essential:

1. Patient's medical history.
2. Medication history.
3. Duration of illness, as prolonged illness can lead to increased heart volume.

4. Physical examination, including assessing the patient's position - whether lying on their back or stomach.

In addition, he noted that most of the time, evidence of pneumonia is usually found at the bottom of the image. So, we chose to crop the images s.t. our models will focus on the important ares of the chest.

2 Solution

2.1 General approach

First Part In the initial phase, we faced a challenge due to the limited size of our dataset. To tackle this issue, we implemented data augmentation techniques, which involve creating new training samples through transformations like rotation, scaling, and flipping. This strategy aimed to increase the dataset's diversity, thus aiding our model in generalizing better to new data.

Recognizing the significance of the central portion of the image where the most relevant information is often located, we consulted with expert, as mentioned above, and subsequently incorporated central cropping techniques into our approach. This adjustment allowed our models to focus exclusively on the important aspects of the image.

Data augmentation played a crucial role in preventing overfitting by introducing variability and discouraging the model from memorizing specific details of the training samples.

On our first try, we tried a very simple architecture of CNN: Three convolutional layers with 64, 32, and 64 filters, respectively, each followed by ReLU activation and max pooling Batch normalization and dropout layers for regularization. Two fully connected layers with 1024 and 256 units, respectively, with ReLU activation and dropout. A final dense layer with sigmoid activation for binary classification. Compiled using Adam optimizer, binary cross-entropy loss, and accuracy metric. The accuracy on the test set was disappointingly low (around 5 percent), likely due to either the small number of parameters or insufficient data, despite employing data augmentation. Therefore, we employed a technique we learned—transfer learning—utilizing pre-trained weights from another task to address our problem.

On our second attempt, we employed the *InceptionV3* model for transfer learning, utilizing its weights and depth for the convolutional part of our network. We adopted a binary crossentropy loss function to address the vanishing gradients issue, and incorporated Dropout layers between each fully-connected layer to mitigate overfitting. The output layer consists of a single neuron with sigmoid activation, ensuring outputs fall within the [0,1] range, interpretable as probabilities. For optimization, we opted for the Adam optimizer with a fixed learning rate of 0.001. The accuracy has been enhanced, yet it still falls short of what's needed for a proficient classifier.

On our last attempt, we employed the *VGG19* model for transfer learning, utilizing **fine-tuning**. This process involved taking the base model of

VGG19, incorporating its pre-trained weights, and augmenting it with additional dense layers along with dropout regularization. Subsequently, we trained all the model's weights, starting from the initial weights derived from *VGG19*. For optimization, we utilized Binary Cross Entropy loss and the *RMSprop* optimizer with a learning rate set to 0.0001, employing Exponential Decay. This decay mechanism gradually reduces the learning rate over time as training progresses.

The effectiveness of this approach was evident as it yielded an impressive accuracy of 98 percent on the validation set, and around 85 percent on the test set.

VGG19 is helpful for transfer learning in CNNs because it has been pre-trained on a large dataset (such as *ImageNet*), learning rich hierarchical features that are generally useful across a wide range of visual recognition tasks. These pre-learned features can serve as a strong starting point for our task, enabling faster convergence and often better performance with less labeled data required.

We proceeded to construct a multi class classifier of healthy, bacterial pneumonia, and viral pneumonia cases. Given that our task involves distinguishing between three classes, namely healthy, bacterial, and viral, we employed the *softmax* activation function for the last layer of the network.

Since our dataset was not initially organized into three classes, we partitioned it ourselves based on the names of the images using a custom script. Following this data preprocessing step, we adopted a similar approach to our previous binary classification task, with some adjustments.

We utilized the *RESNET101* architecture for feature extraction and fine-tuning to address our multi-class classification task. We tried using fine-tuning, but we found out that **not** using fine-tuning has better performance. Additionally, we made adjustments to the learning rate, setting it to 0.001 and incorporating Exponential Decay. This allows the learning rate to gradually decrease over time as training progresses, aiding in effective model convergence and performance optimization. The accuracy was not perfect, but we achieved over 80 percent accuracy on the test set.

Second Part In this section, we demonstrate the process of classifying a new image by leveraging the embedding vector generated from the classification network along with KNN (K-Nearest Neighbors). Additionally, we visualize the distinct classes using t-SNE (t-Distributed Stochastic Neighbor Embedding).

We obtained the embedding vectors from both the binary and multi-class models by removing the last layer from each model. Subsequently, we utilized the KNN implementation from scikit-learn to construct a new model using the embedding vectors generated from the training set. When a new image is inserted, we calculate its corresponding embedding vector and insert it into the KNN model, setting $K = 7$ for the binary classifier and $K = 5$ to the multiclass classifier. The model outputs the class based on the majority of the k nearest neighbors.

In addition, we used T-SNE to demonstrate the scattering of the data. In

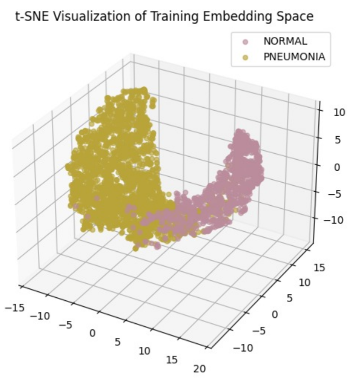

Figure 1: T-SNE of the binary classifier

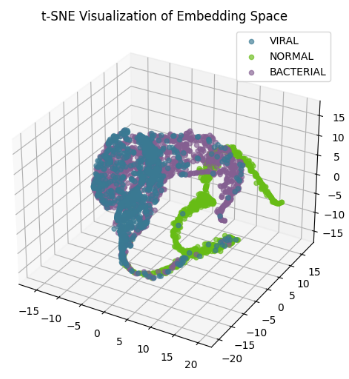

Figure 2: T-SNE of the multi-class classifier

simpler terms, t-SNE tries to find a mapping from the high-dimensional space to the lower-dimensional space where similar data points in the original space are still close to each other in the lower-dimensional space, while dissimilar points are far apart. This allows for the visualization of clusters and patterns in the data that might not be immediately apparent in the high-dimensional space.

Overall, t-SNE helps to visualize the structure and relationships within the embedding vectors, making it easier to understand and interpret the underlying data.

We can observe from the first figure that there is a distinct separation between the embedding vectors of the healthy images and the sick images. In the second figure, we can observe a clear separation between the embedding vectors of the healthy images and the sick images . However, there is less distinction between the bacterial and viral images.

This helps us comprehend the dispersion of the images and gain a better understanding of how the model classifies them.

Third Part On this part, we were provided with only healthy images and tasked with constructing an anomaly detector capable of identifying "sick" images, i.e., anomalies. To accomplish this, we employed an encoder-decoder architecture. The encoder component accepts an image as input and transforms it into a latent space representation, which is significantly smaller in size compared to the input image. Subsequently, the decoder component takes this representation and endeavors to reconstruct the original image, aiming to produce an output image as similar as possible to the input image.

The encoder starts with an input shape of image with original size and uses a series of Conv2D layers with ReLU activation to extract features from the input image. Each Conv2D layer reduces the spatial dimensions while increasing the depth (number of filters). MaxPooling2D layers further downsample the feature maps, capturing the most salient features.

The decoder part begins with Conv2D layers that increase the spatial dimensions through upsampling. Each Conv2D layer refines the features learned by the encoder. Finally, the model uses a Conv2D layer with 3 filters (representing RGB channels) to reconstruct the output image.

The model is compiled using the Adam optimizer and mean squared error loss, commonly used for image reconstruction tasks.

The encoder-decoder model is trained exclusively with healthy images. Consequently, we expect that when a "sick" image is fed into the encoder-decoder, its reconstruction will not be as accurate as that of a healthy image. To detect anomalies, we calculate the mean difference between healthy images and their corresponding reconstructed images. When a new image is introduced to the model, if the loss between the input and output exceeds the threshold (mean difference between healthy images and their corresponding reconstruction), we raise an alarm indicating that the inserted image is an anomaly (i.e., "sick"). Figure 3 depicts the reconstruction the model generates for healthy images, while figure 4 illustrates the reconstruction for sick images.

Unfortunately, all our tries to find a good threshold failed. we used histograms to find the threshold but we got that there is no good one. surprisingly, the model reconstructed sick images better than health images. Our assumption that it is caused by the "cloudiness" of the image, and although the model didn't train on the sick images, he managed to reconstruct due to similarities between the normal and sick images. the cloudiness is likely to show after reconstructing an health chest, so when the input image is cloudy, the model can reconstruct it better.

Fourth Part To accomplish this task, we need to implement and analyze one of the explainability techniques we learned during the course for the model trained in part 1. Explainability of a model provides insight into why a model chooses to classify an image into a particular class.

We have opted to utilize the *Visualizing patches that maximizes activation* technique, also called *Heatmap* which allows us to infer the pixels that were most influential in making the prediction. In Convolutional Neural Networks

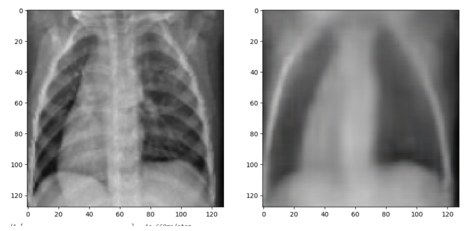
Figure 3: Original health chest image (left) and his Reconstruction (right)

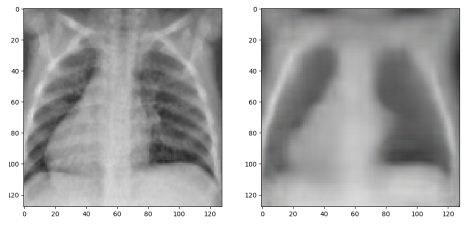
Figure 4: Original sick chest image (left) and his Reconstruction (right)

(CNNs), *Heatmap* techniques for explainability involve generating a heatmap overlay onto an input image, highlighting the regions that the model focuses on for making predictions. Heatmaps facilitate interpretability by identifying the specific features or patterns that contribute most significantly to the model's predictions. By utilizing *Global Average Pooling* in our binary model, the CNN effectively summarizes the spatial information across feature maps, allowing for the generation of heatmaps that highlight the most important regions for classification. This approach provides insight into the model's decision-making process and aids in understanding which image regions contribute most significantly to the predicted class. Let's analyze two images along with their corresponding heatmaps:

In Figure 5, depicting a healthy chest, the model focuses on the sides rather than the presence of "white clouds", which typically indicate a diseased chest.

Conversely, in Figure 6, the model concentrates on the white clouds within the chest region. This indicates that the model searches for these clouds, and their presence influences the prediction. If no clouds are detected, the model is more likely to predict a healthy chest.

2.2 Design

We utilized both *Keras* and *TensorFlow* for our task. Initially, Keras was employed to load the dataset, conduct preprocessing, apply data augmentation, and centrally crop the images. We utilized keras.applications to leverage transfer learning with *VGG19* and *RESNET101*.

To visualize the dataset, results, and the heat map generated for model

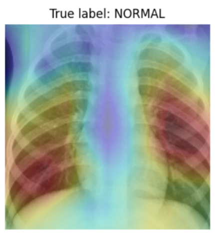

Figure 5: Normal chest heatmap

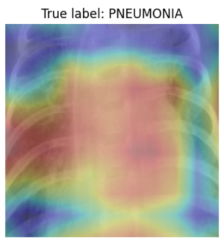

Figure 6: Sick chest heatmap

explainability, we used Matplotlib. Additionally, T-SNE was employed for visualizing the distinct classes.

Thanks to our access to GPU resources, the training time for the initial phase was relatively short, owing to our utilization of Google GPU engines.

For the binary classifier in the first phase, we utilized Binary Cross Entropy loss and the RMSprop optimizer with a learning rate set to 0.0001, incorporating Exponential Decay. RMSprop dynamically adjusts learning rates for each parameter based on the magnitudes of their gradients, facilitating faster convergence and effectively handling varying gradient scales across different dimensions.

For the multi-class classifier, we employed Categorical Cross Entropy loss, RMSprop optimizer, and a learning rate of 0.001.

For the anomaly detector, we used *Adam* optimizer and mean squared error (*MSE*) loss. The train took a long time, maybe because we used the default learning rate of adam which is 0.001. Adam optimizer adapts learning rates for each parameter, blending momentum and RMSProp concepts for efficient convergence in training neural networks.

For the *KNN* classifier, we utilized the sklearn library to split the data into images and their corresponding labels. Then, we constructed a *KNN* classifier from the library, requiring only the training images as input. The data splitting process took a considerable amount of time.

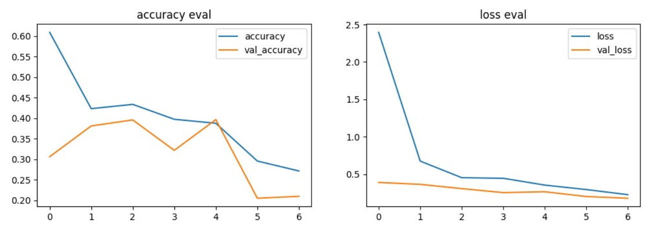

Figure 7: Binary model with transfer learning using *InceptionV3*

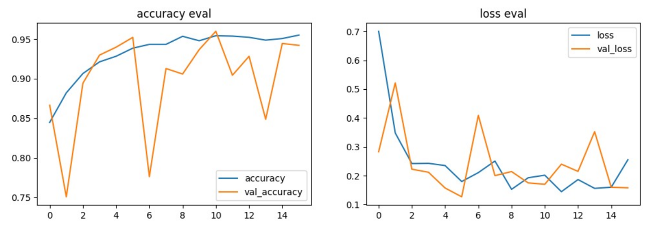

Figure 8: Binary model with transfer learning using *VGG19* with Fine-tuning

3 Experimental results

As we mentioned earlier, we experimented with various models for the binary classifier. Initially, we attempted to construct a CNN from scratch. However, we ultimately found that employing transfer learning was better suited to our task. Afterward, we attempted transfer learning using *InceptionV3* as our base model. However, we only achieved an accuracy of 30 percent, as depicted in Figure 7.

So, we experimented with another base model, *VGG19*. Although we observed some improvement, it wasn't sufficient. Subsequently, we attempted fine-tuning, updating the weights of the base model as well. This approach yielded satisfactory results, as illustrated in Figure 8, achieving 90 percent accuracy on the test set.

We initially experimented with *VGG19* for the multi-class problem, but its accuracy fell short of expectations. Consequently, we opted for another transfer learning approach using *RESNET101* as our base model. Initially, we employed fine-tuning and achieved approximately 70% accuracy on the test set. However, surprisingly, upon removing the fine-tuning, the accuracy soared to 81% on the test set, as illustrated in Figure 9.

For the anomaly detector, we plotted histograms of the mean squared error (MSE) between the original images and their reconstructions for both healthy and sick images. However, we couldn't find a suitable threshold to effectively distinguish between healthy and anomalous chests. Surprisingly, we found that the reconstruction of sick chests was sometimes better than that of healthy chests. Figures 10 and 11 display the histograms of the MSE for the healthy

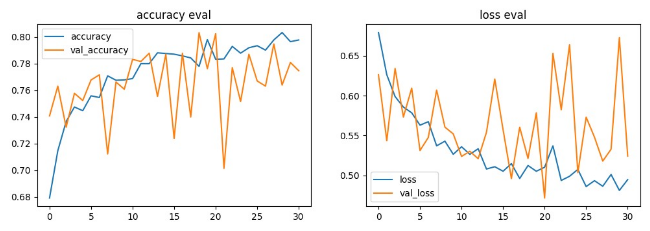

Figure 9: Multiclass model with transfer learning using *RESNET101* without Fine-tuning

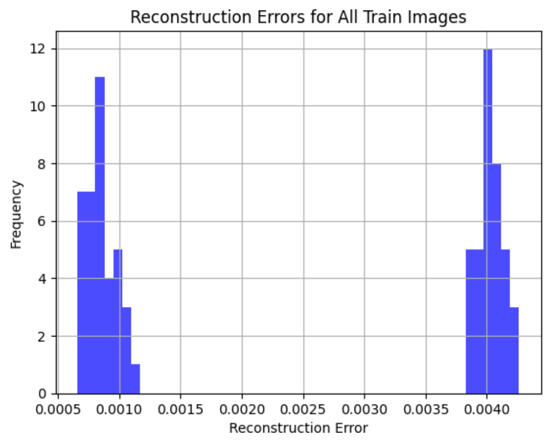

Figure 10: Histogram of the *MSE* between the original healthy (normal) images and their reconstructions

and sick images after they were processed by the encoder-decoder model.

4 Discussion

We found the project both challenging and enjoyable. We encountered several hurdles along the way. Initially, we attempted to build a CNN from scratch, but the results were disappointing. However, leveraging transfer learning proved to be a crucial turning point. It significantly enhanced our model's performance, and further fine-tuning improved results even more.

Moreover, we realized the significance of data augmentation in classification tasks. It not only helped prevent overfitting but also allowed the model to focus on key features within the images.

The t-SNE visualization we employed proved to be extremely beneficial in understanding how the model generates 'signatures' from the images fed into it, and subsequently utilizes these signatures for classification. The scatter plot

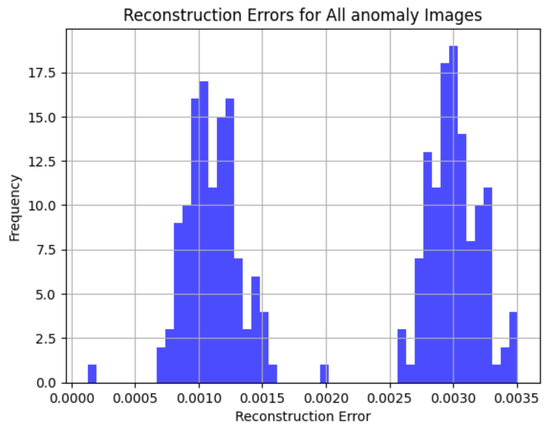

Figure 11: Histogram of the *MSE* between the original anomaly images and their reconstructions

of the t-SNE visualization effectively portrays the separation of different classes in a visually appealing manner within a 3-dimensional space. We found this visualization particularly helpful when assessing the model's ability to accurately distinguish between different classes.

Through this process, we gained valuable insights into the importance of trial and error techniques in this field.

We faced challenges that proved to be difficult to address in practice. For example, while developing an anomaly detector meant to identify abnormalities in diseased chests, we experimented with various architectures of the encoder-decoder. However, we observed an unexpected outcome: the model succeeded in reconstructing images of anomalies more accurately than those of healthy chests. This made it hard to set a good threshold between normal and abnormal images.

Furthermore, when employing explainability methods, we recognized their significance. When a doctor utilizes the model, they can comprehend why the model classifies an image as healthy or not. From this, we can infer that the explainability of the model is not only theoretically valuable but also practically indispensable .

In conclusion, this project presented numerous challenges commonly encountered in practical applications. While some were resolved, others remained unsolvable. Nonetheless, we gained a profound appreciation for the potential of deep learning in benefiting humanity, especially in the medical field. It was particularly striking to witness the efficacy of transfer learning in classification tasks, and the ability of models to learn from data typically understood only by domain experts. This underscores the transformative power of deep learning, even when developers may not fully grasp the intricacies of the tasks they seek to address.

5 Code

- Train notebook: https://bit.ly/KARLSINIDLTRAIN
- Test notebook: https://bit.ly/KARLSINIDLTEST

References

- Transfer learning with fine-tuning - https://www.tensorflow.org/tutorials/images/transfer$_learning$
- Transfer learning of VGG19 - https://koushik1102.medium.com/transfer-learning-with-vgg16-and-vgg19-the-simpler-way-ad4eec1e2997
- Intro to Autoencoders - https://www.tensorflow.org/tutorials/generative/autoencoder
- DigitalSreeni YouTube Channel
- Heatmap - https://tree.rocks/get-heatmap-from-cnn-convolution-neural-network-aka-grad-cam-222e08f57a34
- t-SNE - https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a
- embedding vector for CNN - https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526