Deep Learning Course - Final Project

Nitsan BenHanoch (208585927), Nir Koren (316443902), Tsvi Tabach (311292304)

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

1 Introduction

Deep-Learning in healthcare is a hot topic. As deep neural networks become increasingly involved in patient care, these networks must meet several requirements. Such requirements include high precision rate, transparency in their decision-making, and performing effectively even when valuable data is missing.

1.1 The Dataset

Kaggle's chest-xray-pneumonia dataset consists of 5,856 lungs X-Ray images of men and children. Each image is labeled either "healthy", "bacterial-infected" or "viral-infected". The data is split between train/set sets. One important issue to note is the classes are not balanced; each set contains a different ratio of images from the different classes.

1.2 Our Tasks

In this project, we demonstrate several different deep-learning techniques we learned during the course.

Task 1.a. Binary Classification: we train a CNN on healthy/sick labeled x-ray images of human lungs;
Task 1.b. Multiclass classification: we further improve the CNN to distinguish between bacterial/viral infection;
Task 2. Embedding Space: we apply KNN and t-SNE on embedding vectors we get from the aforementioned networks;
Task 3. Anomaly Detection: having trained on healthy lungs alone, we use auto-encoders to tell if lungs are sick;
Task 4. Explainability: we decipher the model's decision making, e.g. what's important to each layer and neuron.

1.3 Related Works

Previous research, such as CheXNet, study by Rajpurkar et al. (2017), has shown the advantage of using CNN in medical diagnostics. The model proposed in the paper used CNN to output probability of pneumonia with a heat-map localizing the pneumonia areas from chest X-ray images.

2 Task 1.a: Binary Classification

2.1 Solution

2.1.1 The Task

In this task, we treat both bacterial and viral images as *sick*, and train a model to classify *healthy*/*sick* lungs x-ray.

2.1.2 General approach

We use several convolution layers, followed by a few dense layers. It is the field's classic and go-to way to classify images. Furthermore, the limited dataset size (approximately 5K images) posses a serious restriction on the complexity of the network, as overly powerful architectures risk significant over-fitting.

2.1.3 Design

Our network's architecture mimics the great: we took inspiration from the architecture of the VGG networks family. We tailored their layer sizes to our needs after thorough experimentation, maximizing accuracy, while battling over-fitting. The final network structure is portrayed in Figure 1.

Platform: we built the model as a Keras Sequential on a kaggle notebook. It ran for ∼30 minutes using a P100 GPU.
Loss: we used binary-crossentropy, as it's classic in the context of binary classification, and it worked for us.
Optimizer: we used Adam with initial learning rate of 1e-4, but also installed a callback to further lower it on plateau.

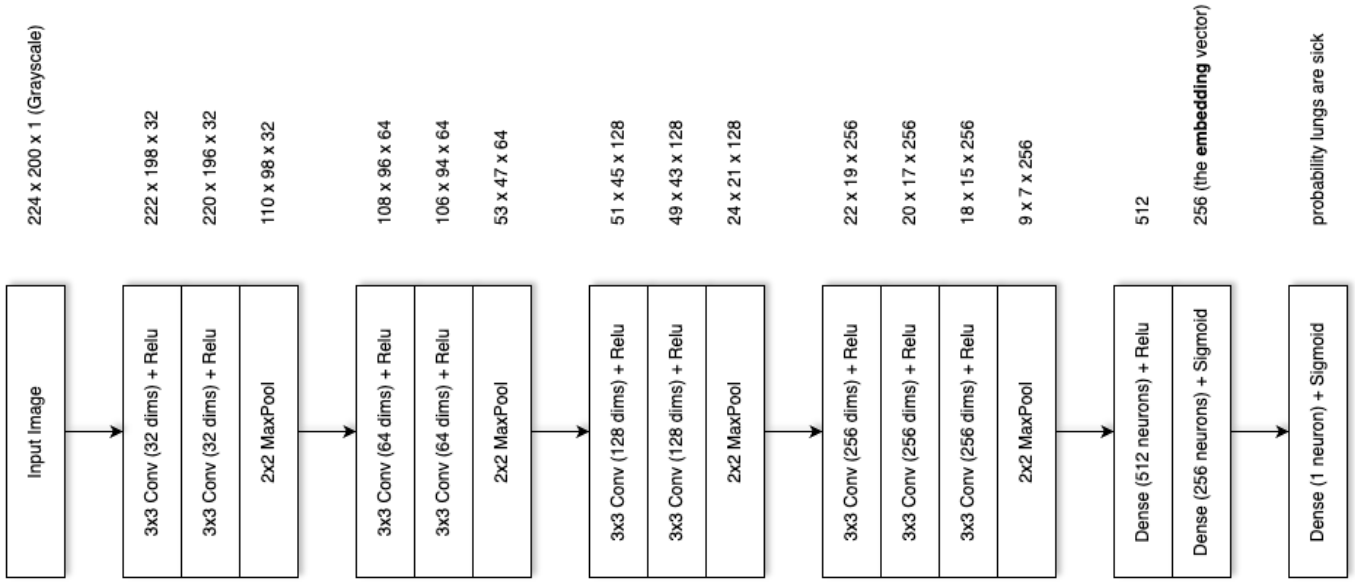


Figure 1: Binary Classification Model

The Splits: The train and the 16-images given val were concatenated and re-split into 80% train and 20% validation.

Limited dataset size: as said, we only had a few thousand images to train on. To combat it, we used many augmentation methods to compensate for the lack of data. We also limited the number of epochs.

Class imbalance: the sets had different distributions of labels. We removed excess data from the train/val sets until each class appeared 50%; we found that to be more effective than assigning weights. The test-set, having a 62.5% class imbalance, was left as is.

2.2 Experimental results

Provide information about your experimental settings. What alternatives did you measure? Make sure this part is clear to understand, provide as much details as possible. Provide results with tables and figures.

2.3 Discussion

Provide some final words and summarize what you have found from running the experiments you described above. Provide some high level insights.

Note - your project will be evaluated for aspects, including the technique you selected, the rational of the experiments you decided to run, the insights you learned from this process and more. Remember, for the purpose of this course, the process that you demonstrate is very important.

2.4 Code

Please provide a link to your colab notebook.

3 Task 1.2: Multiclass Classification

3.1 Solution

3.1.1 General approach

For the multiclass class classification task, We used the ResNet34 architecture with 3 sized classification head. Some alternatives such as Vision Transformer and Dense Net were tested, but the ResNet34 provided the most accuracy on the test set.

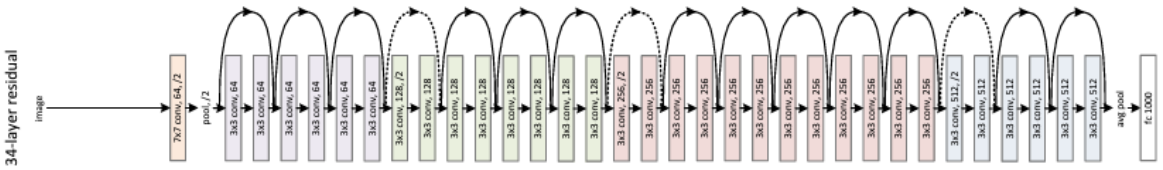


Figure 2: ResNet34 architecture from the original paper

3.1.2 Design

The ResNet34 architecture we used was taken from the original paper Deep Residual Learning for Image Recognition by Kaiming He. The architecture includes 34 convolution layers and residual connections between each block. The model is divided into two main components: ResNet34 backbone and 3-sized classification head. A single modification was made to the original ResNet34 architecture: the activation functions were switched from ReLU to GELU to enhance performance.

Platform: We built the model as a Pytorch class on a kaggle notebook. It ran for 20 epochs and took ∼9 minutes using a P100 GPU.

Loss: We used cross-entropy loss, a go-to loss function for classification tasks.

Optimizer: We used AdamW with an initial learning = 1e-3, weight decay = 0.005

Scheduler: We used a Warmup Cosine Schedule with 5 warmup steps

The Splits: For a more representative validation set we moved 400 samples from the training dataset to the validation dataset. At last, we had 4816 training samples, 416 validation samples, and 624 test samples.

3.2 Experimental results

The ResNet34 model underwent 20 epochs of training, showcasing a gradual improvement in both training and validation accuracies. Initial epochs demonstrated a significant loss decrease and accuracy improvement, stabilizing as the epochs progressed. The final recorded training accuracy was 81.89

The test performance was independently evaluated using the model weights that achieved the best validation accuracy. This assessment resulted in a test accuracy of 83.28

3.2.1 Comparison to Alternatives

Several alternative architectures were evaluated, including Vision Transformer, DenseNet, and fine-tuning of pretrained models. Ultimately, the non-pretrained ResNet34 model was selected as it surpassed all other options in performance. Our primary hypothesis for its superior performance is that the residual connections within the ResNet34 help mitigate the vanishing/exploding gradient problem, thus enabling the construction of a deeper network that excels in classification tasks. Additionally, the pretrained models were initially trained on datasets significantly different from ours, such as ImageNet. This discrepancy likely contributed to the poorer performance observed when fine-tuning these models compared to using a freshly initialized network.

3.2.2 Visualizations

Training and validation loss and accuracy over epochs were plotted, with the final test loss and accuracy 3. Additionally, a confusion matrix on the test set was plotted to visually assess the relationships and misclassifications between classes.

True/Predicted	Normal	Bacteria	Virus
Normal	189	17	28
Bacteria	9	231	2
Virus	14	33	101

3.3 Discussion

Our exploration validated the non-pretrained ResNet34 as the optimal choice for multiclass classification on the chest-xray-pneumonia dataset. Moreover, selecting the right optimizer and learning rate decay schedule is crucial for optimal performance. Further research into hyperparameter optimization could significantly enhance results.

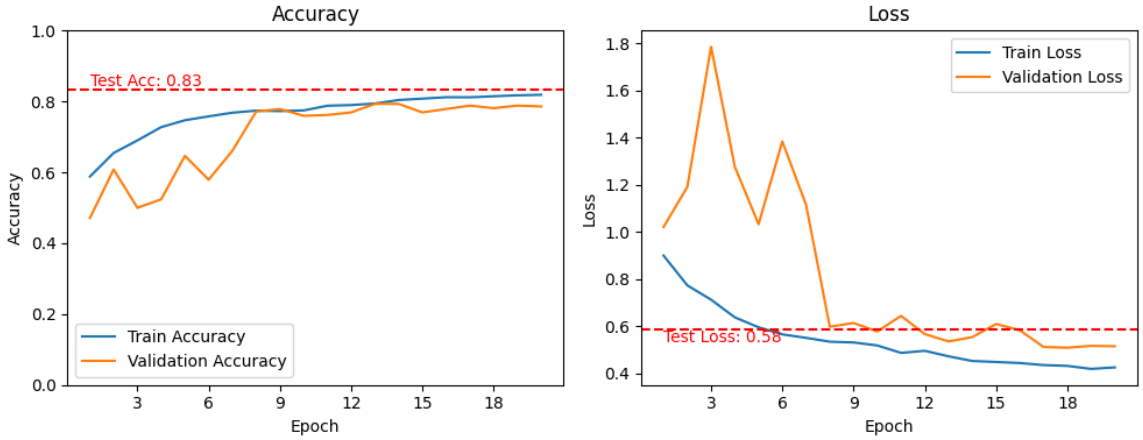


Figure 3: ResNet34 training trends

3.4 Code

Testing Notebook Training Notebook

4 Task 2: Model Embeddings and Visualization

4.1 Task 1.b

4.1.1 Embedding and KNN classification

In Task 1.b, we employed a ResNet34 model for multiclass classification purposes. This architecture generates embeddings as 1000-dimensional vectors, which are subsequently inputted into a final linear layer with dimensions 1000x3 for classification. Leveraging these embeddings, we trained a K-Nearest Neighbors (KNN) classifier using the same training dataset as in Task 1.b. We configured the KNN model to utilize 7 neighbors, which achieved an accuracy of 70.03% on the test set. Additionally, a confusion matrix on the test set was plotted to visually assess the relationships and misclassifications between classes using KNN classifier.

True/Predicted	Normal	Bacteria	Virus
Normal	97	55	82
Bacteria	3	226	13
Virus	1	33	114

4.1.2 Classes Visualization

To visualize how the trained network perceives the classes within the dataset, we extracted the embeddings produced by the ResNet34 backbone. These embeddings were then visualized using the t-SNE technique, facilitated by the t-SNE library, to represent the high-dimensional vector space in a two-dimensional plot. As illustrated in Figure 4, the plot reveals a distinct separation between the Normal and non-Normal classes. Additionally, there is a moderate but discernible linear separation between the Virus and Bacteria classes.

4.2 Code

Please provide a link to your colab notebook.

5 Task 3: Anomaly Detection

5.1 Solution

5.1.1 General approach

Describe your preferred approach to solve the problem. what alternatives you plan to try and why.

Figure 4: T-sne visualization of the classes

5.1.2 Design

Provide some general information about your code, platform, how long it took you to train it, technical challenges you had, Loss functions, Optimizers, Architecture, and more.

5.2 Experimental results

Provide information about your experimental settings. What alternatives did you measure? Make sure this part is clear to understand, provide as much details as possible. Provide results with tables and figures.

5.3 Discussion

Provide some final words and summarize what you have found from running the experiments you described above. Provide some high level insights.

Note - your project will be evaluated for aspects, including the technique you selected, the rational of the experiments you decided to run, the insights you learned from this process and more. Remember, for the purpose of this course, the process that you demonstrate is very important.

5.4 Code

Please provide a link to your colab notebook.

6 Task 4: Explainability in Deep Learning

6.1 Solution

6.1.1 General approach

For the explainability of the network in Assignment 1.a, we employed three main approaches: visualizing the receptive fields that correspond to the maximum activations of various filters, generating heatmaps through the occlusion of different classes (Normal and Pneumonia), and synthesizing inputs that trigger high activations in specific layer neurons.

6.1.2 Receptive Fields Corresponding to Maximum Activations

In this approach, we selected one of the later convolutional layers (conv2d) of the model. For each of the five filters in this layer, we processed ten images from the dataset. During each forward pass, we identified the point of maximum activation for each filter (a single value from the filter's output) and calculated the corresponding receptive field. We then visualized these receptive fields to gain insights into what the network focuses on during the forward pass. See Figure 6a for the plots. We can conclude from the plots that every filter is "looking" for some feature of the image. For example, filter 1 reacts the most for the left lung, while filter 2 reacts the most to the spine.

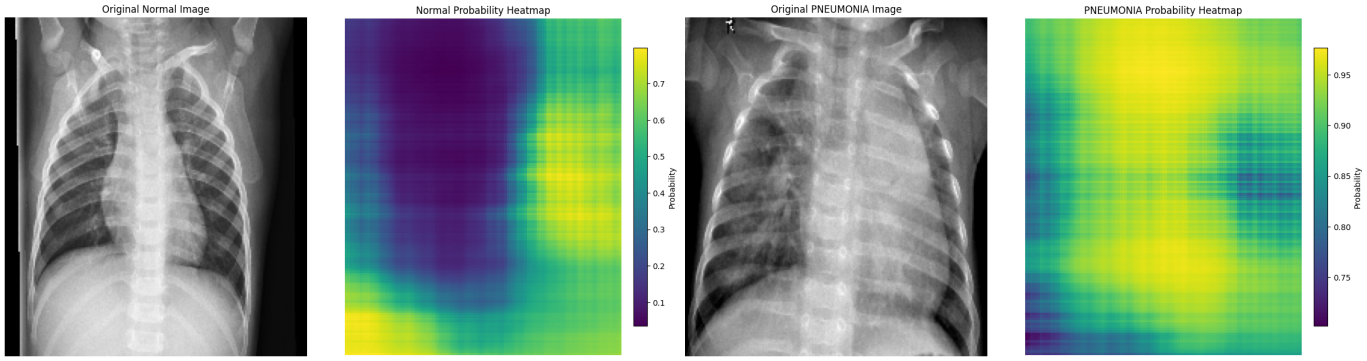
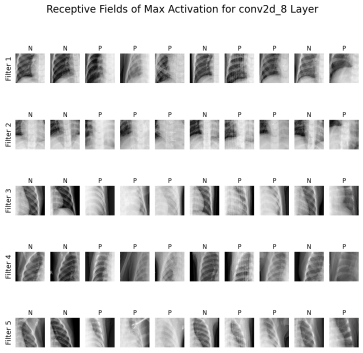
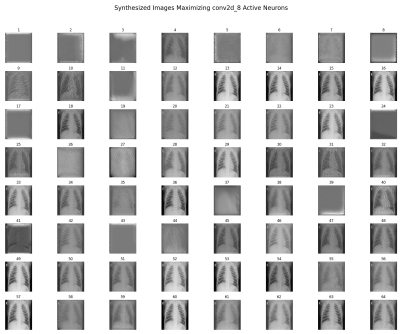


Figure 5: Heatmaps by Occlusion



(a) Receptive fields of max activation. (N=Normal, P=Pneumonia)



(b) Synthesizing inputs that cause high activation

Figure 6

6.1.3 Heatmaps by Occlusion

This method provides insights into the network's decision-making process. We selected two images, one labeled "Normal" and the other "Pneumonia," and applied a masking technique where different areas of the images were obscured with a zero mask of size 90 pixels. This process generated 14,985 masked variations for each image. Each masked image was then processed through the network, and the predicted probabilities were recorded, creating a probability map of size 111x216 for each image. We adjusted the Normal image's probability map by converting the values to 1-value, aligning it with the Pneumonia map, where values closer to 0 indicate critical features (poor model accuracy) and values closer to 1 indicate non-critical features (high model accuracy). As depicted in Figure 5, the analysis reveals that the network focuses on a large portion of the lungs when classifying a Normal image and concentrates more on the central parts of the lungs for a Pneumonia image, despite high overall precision indicating that the model can utilize multiple areas effectively.

6.1.4 Synthesizing Inputs

This approach involves selecting a single image from the dataset and a specific layer within the network. We initiated 64 distinct "training" sessions, using the chosen image as the starting input with its pixels set as trainable parameters. For each session, a different filter from the selected layer was targeted, and gradient ascent was employed to maximize the filter's activation in response to the input. This method helps identify the visual patterns and features that each filter within the layer is most responsive to. As shown in Figure 6b. Each session consisted of 1000 epochs, with a notably high learning rate of 10,000.0. Although this rate may seem unusually large, it was necessary to effect significant modifications to the input image.

6.2 Code

Please provide a link to your colab notebook.

Good luck!!