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.

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. We applied many forms of augmentation (e.g. shear, shift, rotation, brightness, zoom) to compensate for the lack of data, making room for a deeper network while avoiding over-fitting. We also limited the number of epochs; we found that lowering the learning-rate too much, to squeeze more "improving" epochs, actually hurt the the network's accuracy measured on the test set.

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.

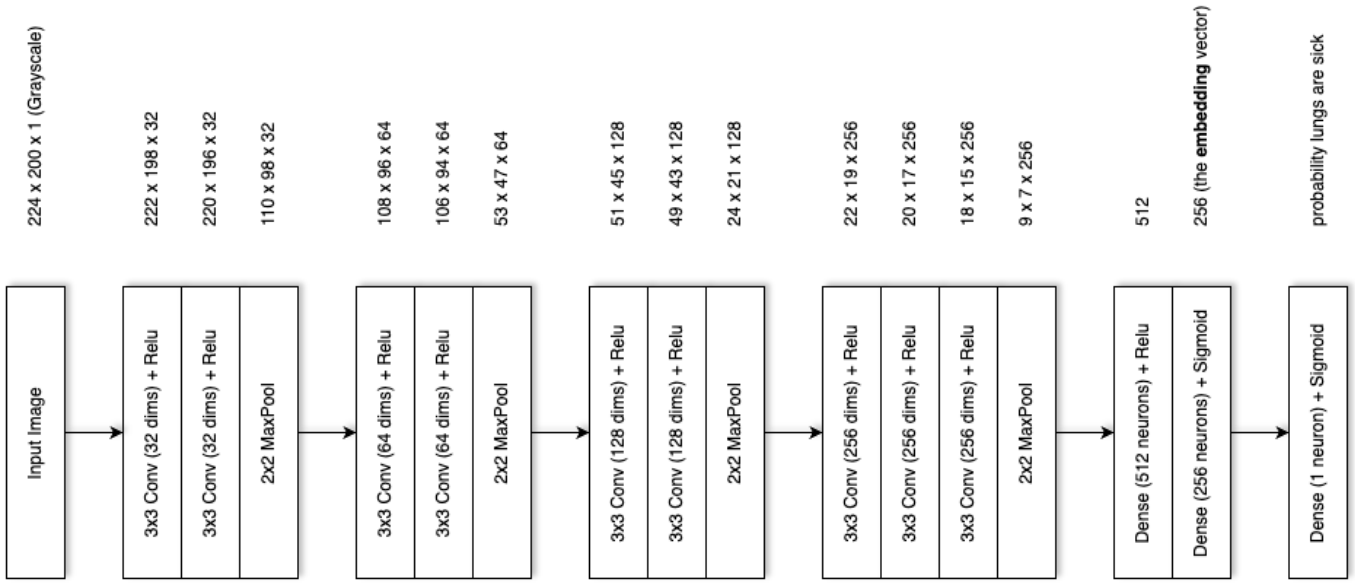


Figure 1: Binary Classification Model

2.2 Experimental results

We tweaked many aspects of our code during its journey to its final best form.

Class imbalance: the most challenging factor turned to be the class imbalance in the data. When we matched the *distribution of labels* in the train/val sets to 62.5% (which is the distribution of labels in the test-set), the network achieved a final accuracy score of ∼96% (measured on the test-set). Not altering the *distribution of labels* in the train/val was (as expected) the worst - it limited the final accuracy to about ∼80%. We decided (after e-mailing the course's TA) that "peeking" at the distribution of classes in the test-set was cheating, so we had two options: removing excess train/val data until the distribution was 50%, or using different weights for each class in the loss function. We tried both things, and achieved similar results, but the trimming of data into 50% representation has yielded results a tad superior, giving us a final accuracy score of ∼92% over the test-set.

Optimizer: we shortly plugged in other optimizer algorithms, and found they brought no improvement over Adam, or (like in the case of Adamax), resulted in inferior final test accuracy.

Learning rate: we tried other values other than 1e-4, and were surprised to find out they hurt the final network's test-accuracy to a significant degree. Even starting with 1e-4, and then lowering it on plateau to 1e-6, has led to immense over-fitting, causing the final accuracy score to drop to around 80%.

Weight transferring: we tried copying the actual weights from the early layers of VGG16 (whose structure we adopted and tweaked), but it provided no help to the model final accuracy. We suspect it might happened due to our needs being specific to intricate blobs in the lungs, while VGG is more general form, and was designed to classify more general shapes; hence transferring its early weights would probably help detecting that the image *is* lungs, but not further. Also, VGG was trained on RGB images, while our x-ray images are grayscale; using VGG might just lead to the model seeking information in the wrong places.

Input image dimensions: VGG uses 224x224 images. We measured the aspect ratio of our images to be roughly 1.124, which is better approximated in 224x200 images. However, after experimentation, it turned out that both resolution bring the network to the same final performance.

Combating over-fitting: over-fitting was a great source of tension in developing the model. On one side, a bigger network is capable of more accurate results. On the other sides, given the limited size of the data (after balancing the classes, the train-set contained only 2,158 images to work with), a network too powerful would only memorize our data, achieving

poor score on the test. We applied many forms of train augmentations (shear, shift, rotation, brightness, zoom) and it did help. We also stretched the size of the network (being its depth, the number of channels in each convolution layer, the number of neurons in the dense layers) to the absolute limit defined by the final test accuracy.

"Hyper-Convergence": it was difficult to draw rigorous scientific conclusions about the optimal network structure, as every experiment took half an our to run, and random luck carried an important $\pm 5\%$ role in determining the network's final accuracy. Of course, repeating each experiment many times in parallel would allow us to reduce the variance, but our compute resources were limited.

In Figure **??**, you can see the Loss fluctuating, dipping almost randomly at its lowest point on epoch 43. Indeed, different runs yielded different minimum points (meaning different final test accuracy).

Our goal was maximizing test-accuracy, even at the cost of val-loss fluctuation in training, and even at the cost of having to run the notebook a few times and pick the best performing version. Overall, this way we were able to reach the highest test-accuracy of $\sim$92% (again, this is without matching the labels distribution to the test. allowing it would make our model's score peak at $\sim$96% test-accuracy).

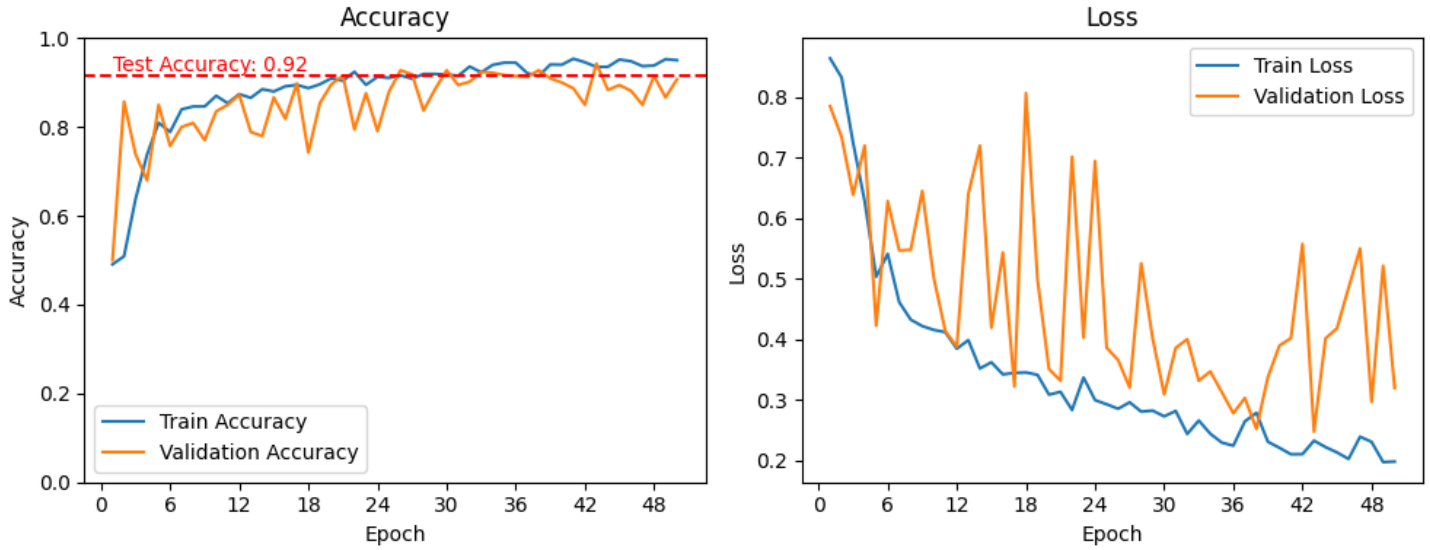


Figure 2: Training the Binary Classification Model. Best Epoch is 43.

2.3 Discussion

To summarize the above points, we found that data is indeed key; a limited number of train examples posses an immense over-fitting barrier on the model size, which in turn sets a ceiling to the complexity of the patterns the model can learn. Matching the distribution of labels with the target test distribution also matters a lot; moving the final test accuracy between 80% and 96%. We found that exact image dimensions do not matter; we suspected that 224 could halve better (in max-pooling), and that 200 would better compliment the original aspect ratio; but in practice, every number in that range yields similar results. Predictability is hard; our models achieved different scores when we repeated the experiment on Kaggle. Please refer to the above sections for more thoughts, as this section cannot contain everything mentioned above.

2.4 Code

View the notebook online. There we're creating the binary classification model, training it, and assessing its performance. You'll also see answers to later questions on the same model, such as KNN in embedding-space, t-SNE, explainability, etc.

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.

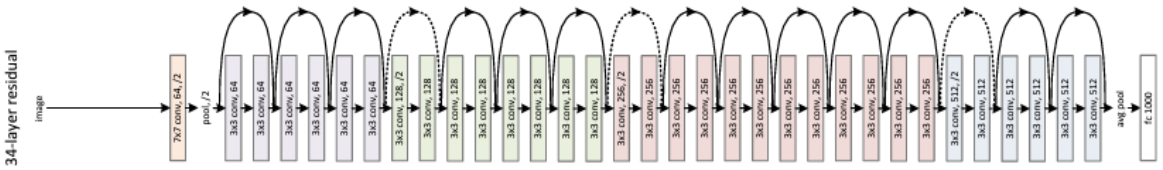


Figure 3: ResNet34 architecture from the original paper

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%, with a loss of 0.4253, whereas the validation accuracy peaked at approximately 79.33% with a similar loss range. The model weights were saved every time the validation accuracy peaked.

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% and a loss of 0.5847, demonstrating robust generalization.

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.

We implemented data augmentation techniques as a strategy to prevent overfitting. However, our experimental results did not show an improvement in test accuracy from these augmentations.

Furthermore, we experimented with using the Stochastic Gradient Descent (SGD), and regular Adam optimizers, but this approach resulted in reduced accuracies compared to our baseline AdamW optimizer.

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

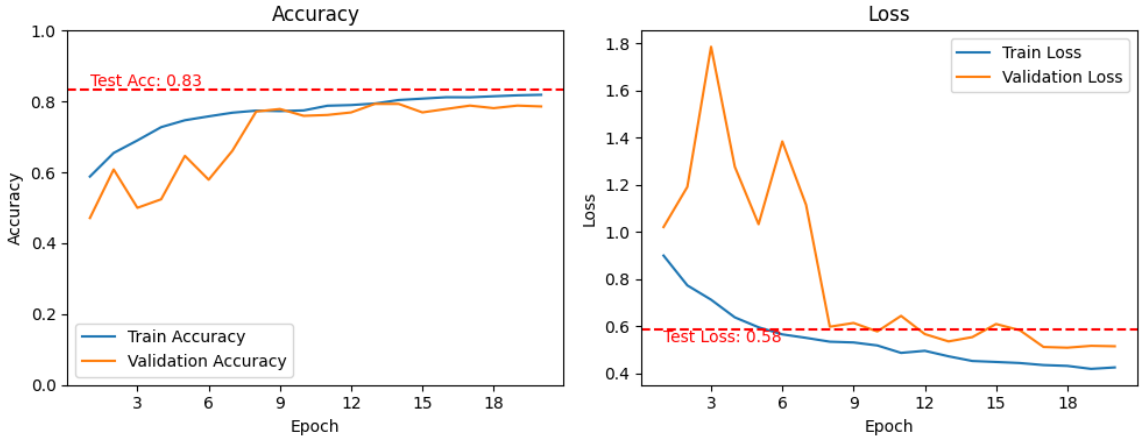


Figure 4: ResNet34 training trends

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.

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 **??**, 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.

Figure 5: T-sne visualization of the classes

5 Task 3: Anomaly Detection

5.1 Solution

5.1.1 The Problem

In this task, you only have data for "healthy" images. Try to identify "sick" individuals using Anomaly Detection methods.

This task is the most challenging. For once, the area the lungs occupy is only ∼20% of each image's pixels, yet in it lies most (if not all) of the relevant data; The need to direct to model to focus on some moving ∼20% area (the lungs) is clear. In addition, the size of the train-set this time was even more limited than in previous tasks, being only 1,079 "normal" images (after 20% cut for validation).

5.1.2 General approach

We use an existing lungs-segmentation model to **concatenate a lungs-only image** next to each original image, and feed the resulting *224 x 448* image as an input to our anomaly detection model. This helps our model focus on the lungs, while still seeing the "complete picture", which is more regular and predictable than just a blob of lungs floating in an all-black sea.

Our anomaly detection model is an **auto-encoder with a wide "bottle-neck"**. The "encoder" expends the dimensionality of the image, in an attempt to capture more patterns in it, rather than losing them in compression.

The "encoder" (which more appropriately could be named "feature-expender") mostly consists of the early layers of VGG16 (with frozen weights); we only append one trainable convolution layer.

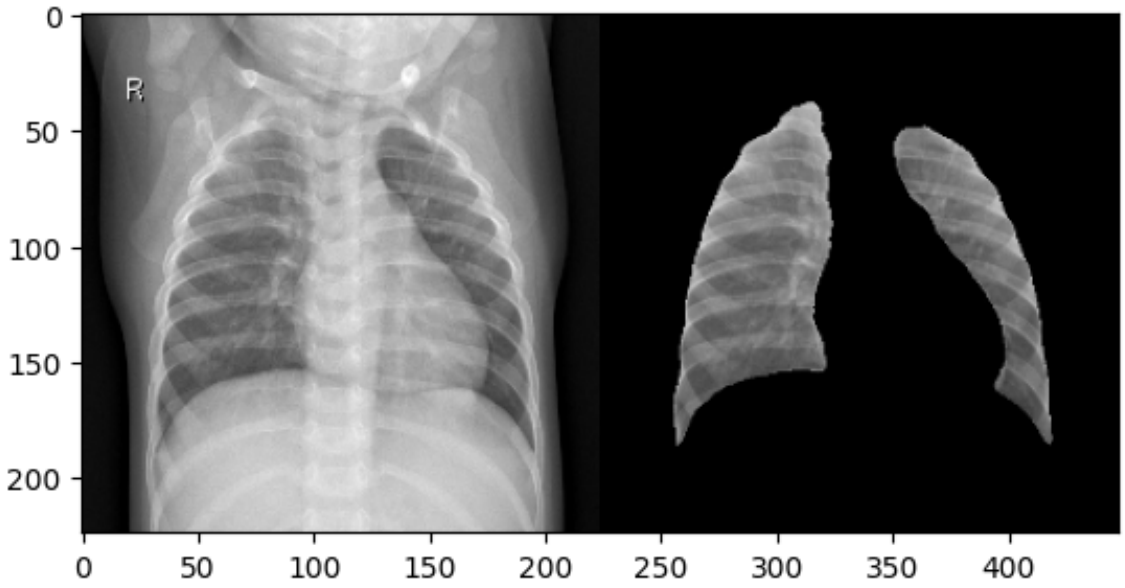


Figure 6: Pre-processed Image for Anomaly Detection Model

5.1.3 Design

Concatenating each Image alongside Lungs-Only Image: after asking the TA for permission, we used a pre-trained lungs-segmentation U-Net, to create a lungs-only version of each image in the dataset; we then concatenated the lungs-only version to the right of each original image.

Wide Bottle-Neck "Auto-Encoder": our "encoder" outputs tensors of shape *112 x 224 x 32*; It's 8 times more dimensions than the input image, a mere *224 x 448 x 1*. This makes our model "describe" an image with more details, rather than trying to compress it like a traditional auto-encoder.

Custom Loss: we compute SSIM on each half individually, then sum the results with weights. The reconstruction of lungs half is multiplied by *0.75*, and the reconstruction of the original half is multiplied by *0.25*.

Augmentation: apart from the lungs-segment concatenation, our pre-processing is minimal. It consists of slight variation in brightness and contrast, which we only apply to the original half.

Anomaly Definition: we compute the *mean* (μ) and *standard deviation* (σ) of the validation loss. A test image is considered an anomaly if its reconstruction loss is NOT within $\mu \pm \sigma$.

5.2 Experimental results

Lungs-Segment Concatenation:

- When we trained the model on the original images alone, it treated the entire x-ray (including the chin, armpits, spine, heart, and so on) the same; it resulted in negligible loss difference between test-normal and test-sick.
- When we trained the model on the lungs-segment images alone, the pattern was too irregular; the model would always converge to the local minimum of constructing almost completely-black images, no matter the loss we used (MSE, SSIM, cross-entropy, dice-loss).
- Our solution was to concatenate each original image alongside its lungs-segmented image, creating a dataset of 2:1 rectangular images. Our custom loss function makes the model focus on the lungs, while still using the "complete picture", which is easier to reproduce.

Loss Function:

Using the same loss strategy (a weighted sum of the loss of each half), we tried replacing SSIM with MSE. We were surprised to find it completely broke the model: instead of the regular result, where the standard deviation of the loss is an order of magnitude less than its mean, we got the opposite: a huge standard deviation, ten times the mean. This nullified the possibility to distinguish between "normal" and "sick" based on reconstruction loss.

"Bottle-Neck" Shape:

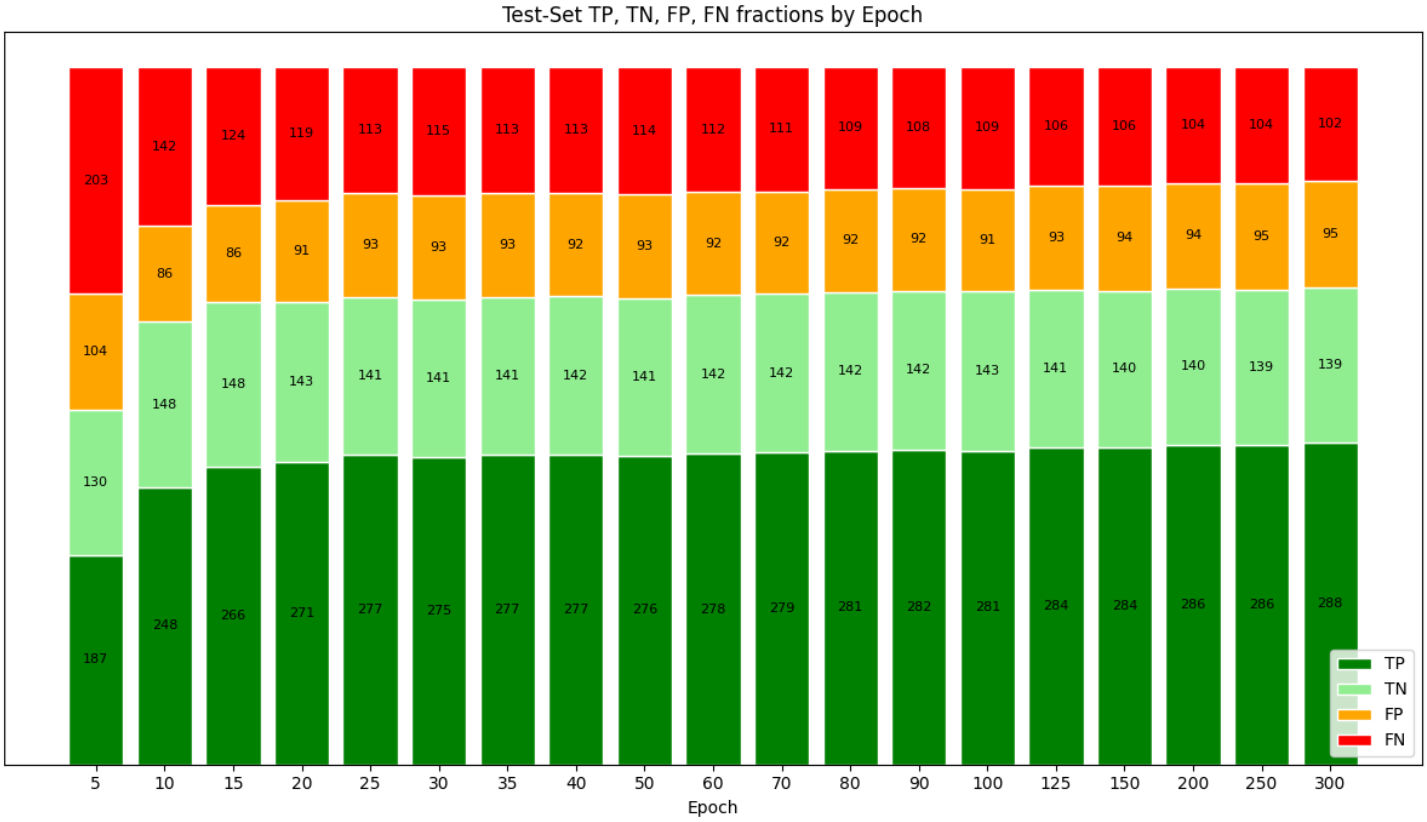
In its final version, our model looks like a diamond, rather than a traditional auto-encoder's bow-tie. When we tried using an actual "bottle-neck" - being a vector of size 256 to 2048, the model's reconstructed images looked very blurred. Even in the best blurred version, the lungs were a monochromatic area; the model captured their contours, but missed 100% of the details inside them.

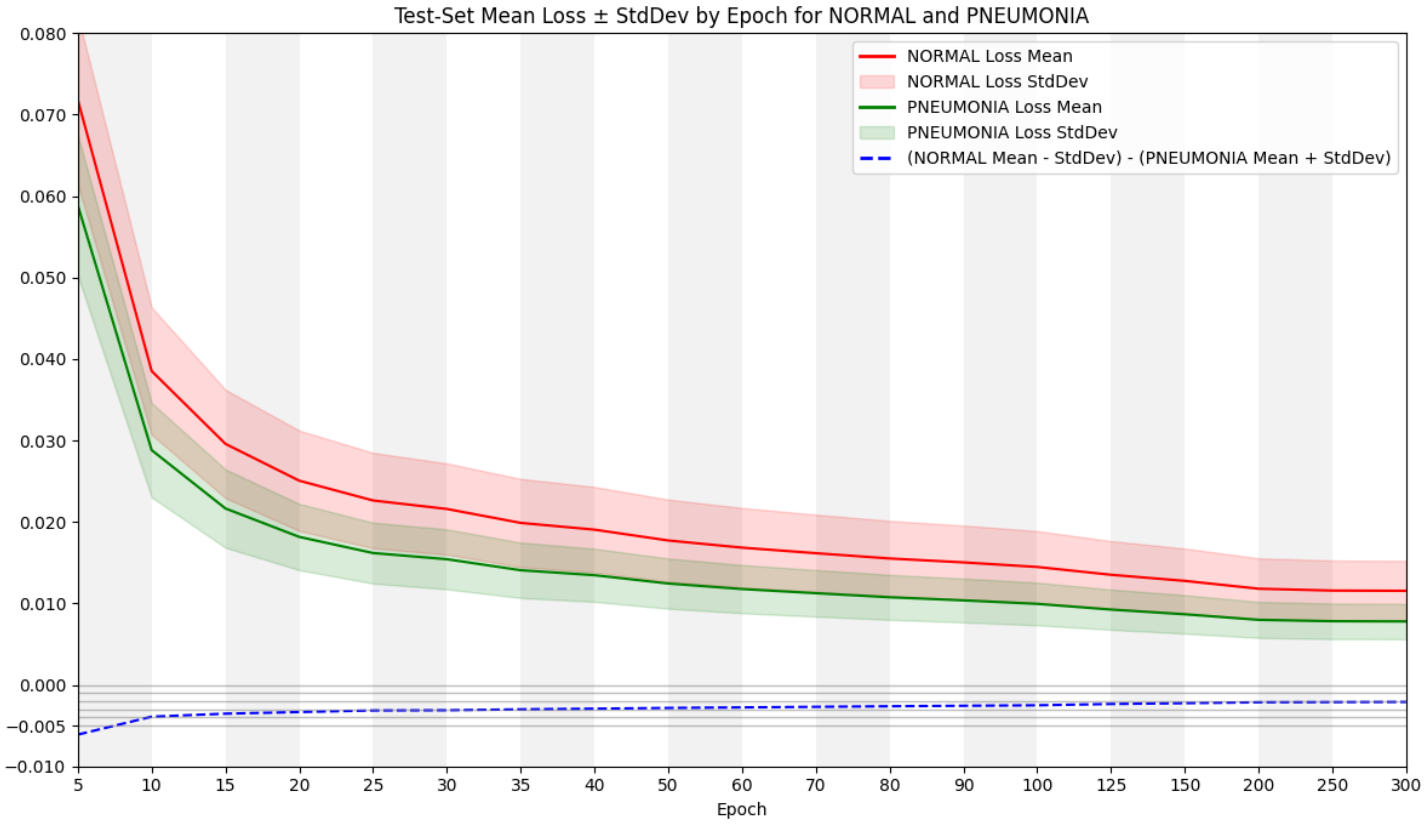
Training Duration and Fine Tuning:

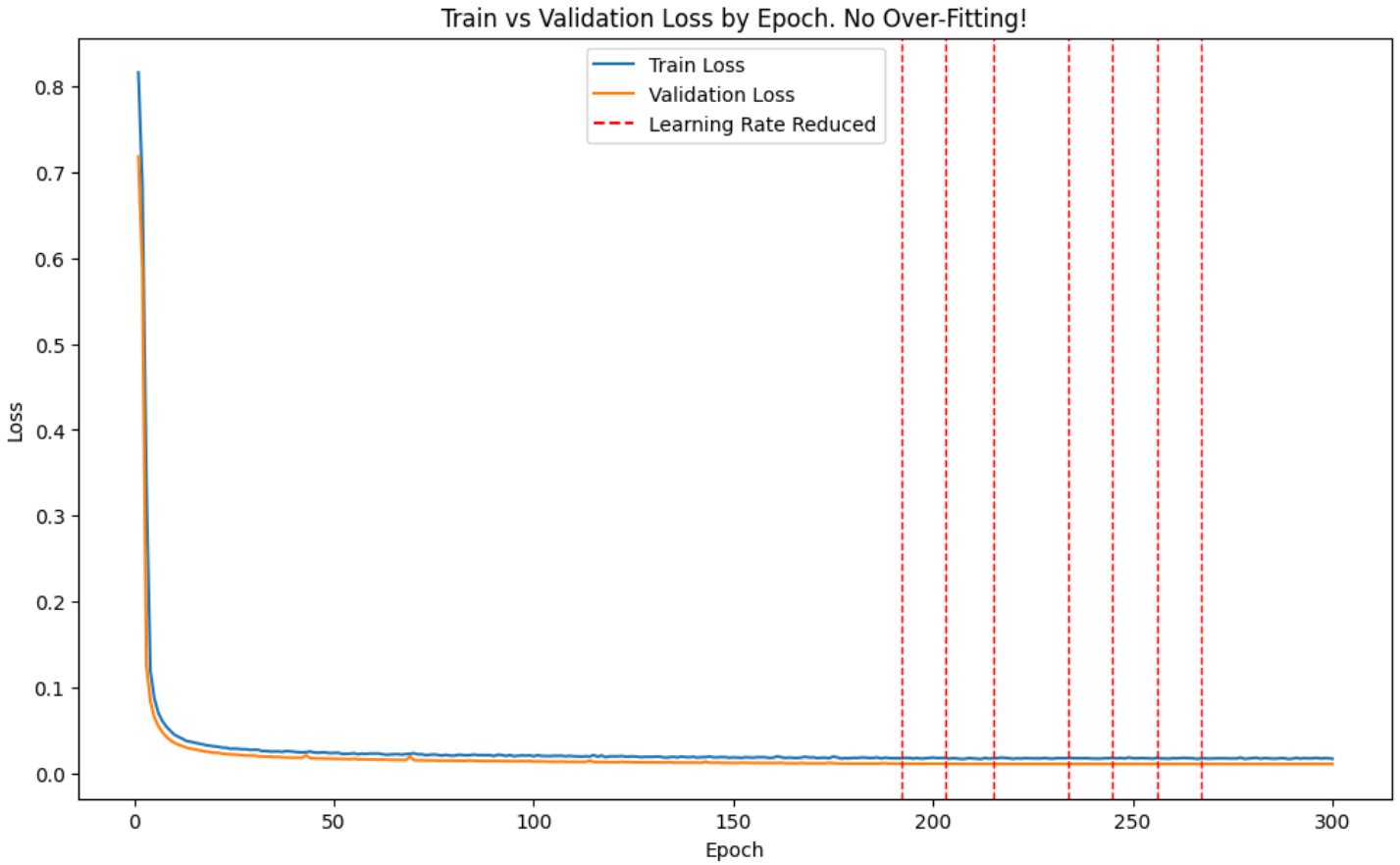
When we transferred the first layers of VGG16, we froze its weights. We then let the model [TODO]

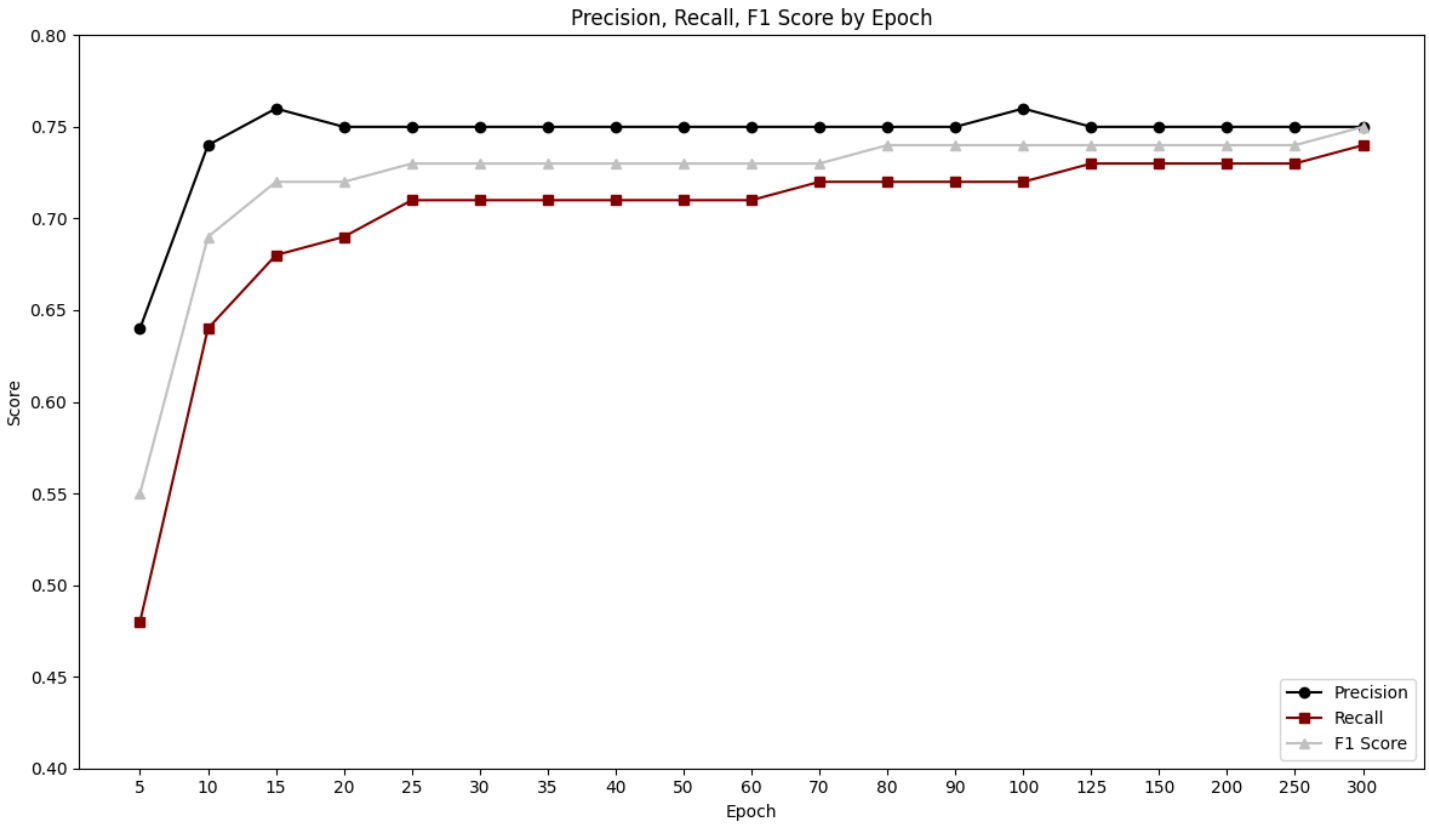
Measuring Improvement During Training:

We relied on validation [TODO]




Test-Set Mean Loss ± StdDev by Epoch for NORMAL and PNEUMONIA
NORMAL Loss Mean
NORMAL Loss StdDev
PNEUMONIA Loss Mean
PNEUMONIA Loss StdDev
(NORMAL Mean - StdDev) - (PNEUMONIA Mean + StdDev)
0.080
0.070
0.060
0.050
0.040
0.030
0.020
0.010
0.000
−0.005
−0.010
5
10
15
20
25
30
35
40
50
60
70
80
90
100
125
150
200
250
300
Epoch


Train vs Validation Loss by Epoch. No Over-Fitting!
Train Loss
Validation Loss
Learning Rate Reduced
Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
50
100
150
200
250
300
Epoch



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 5a 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.

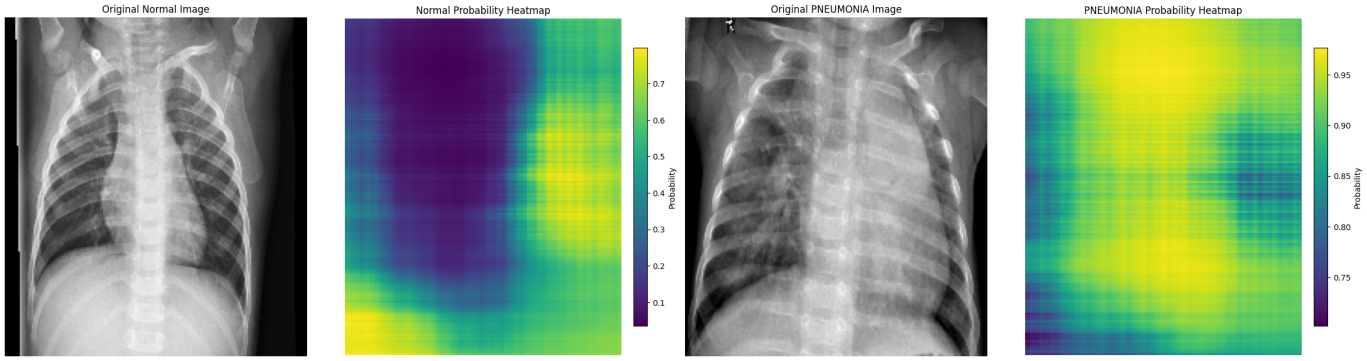
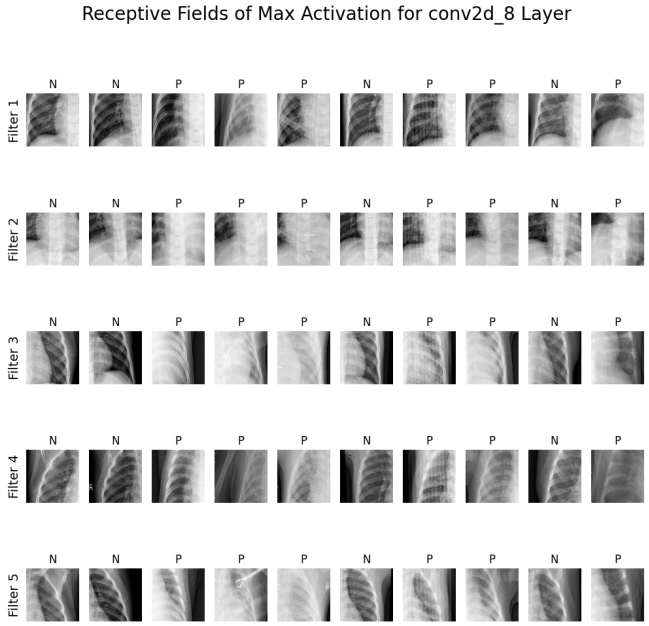
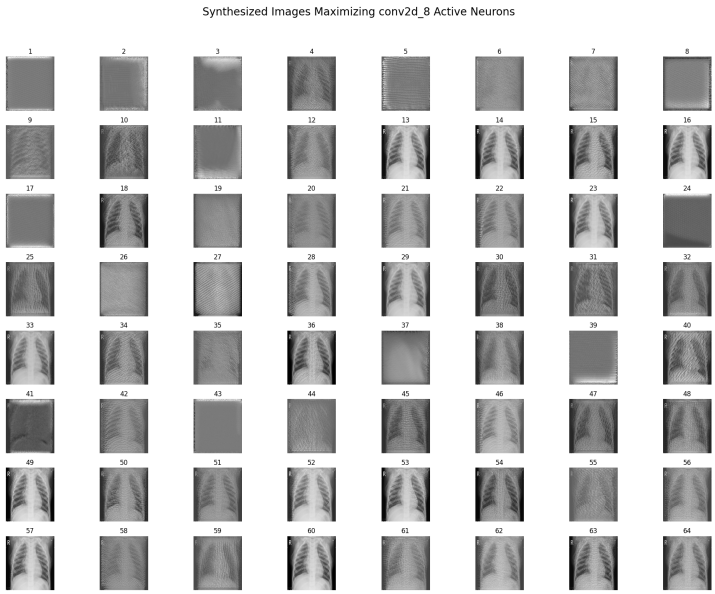


Figure 7: Heatmaps by Occlusion



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



(b) Synthesizing inputs that cause high activation

Figure 8

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 4, 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 5b. 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!!