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.
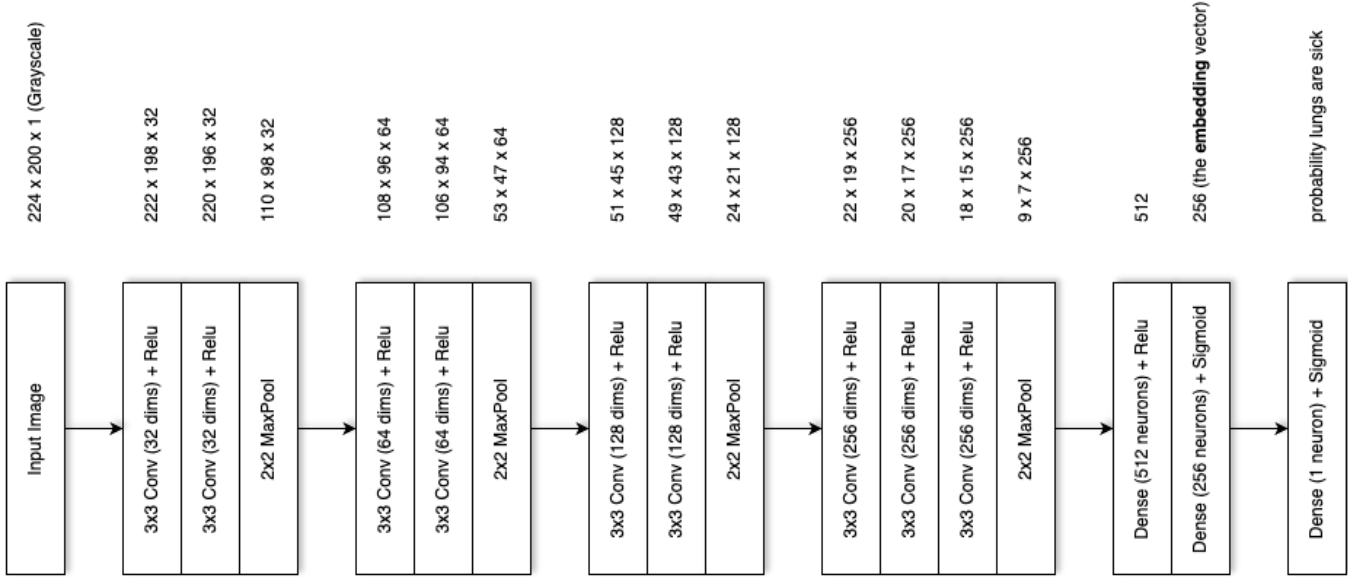
Figure labels (top, left to right):
224 x 200 x 1 (Grayscale) · 222 x 198 x 32 · 220 x 196 x 32 · 110 x 98 x 32 · 108 x 96 x 64 · 106 x 94 x 64 · 53 x 47 x 64 · 51 x 45 x 128 · 49 x 43 x 128 · 24 x 21 x 128 · 22 x 19 x 256 · 20 x 17 x 256 · 18 x 15 x 256 · 9 x 7 x 256 · 512 · 256 (the **embedding** vector) · probability lungs are sick

Figure blocks (left to right):
Input Image · 3x3 Conv (32 dims) + Relu · 3x3 Conv (32 dims) + Relu · 2x2 MaxPool · 3x3 Conv (64 dims) + Relu · 3x3 Conv (64 dims) + Relu · 2x2 MaxPool · 3x3 Conv (128 dims) + Relu · 3x3 Conv (128 dims) + Relu · 2x2 MaxPool · 3x3 Conv (256 dims) + Relu · 3x3 Conv (256 dims) + Relu · 3x3 Conv (256 dims) + Relu · 2x2 MaxPool · Dense (512 neurons) + Relu · Dense (256 neurons) + Sigmoid · Dense (1 neuron) + Sigmoid

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 $\sim$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 $\sim$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 $\sim$92% over the test-set.

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 below 85%.

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 ±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 2, you can see the Loss fluctuating, dipping almost randomly at its lowest point at 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 ∼92% (again, this is without matching the labels distribution to the test. allowing it would make our model's score peak at ∼96% test-accuracy).
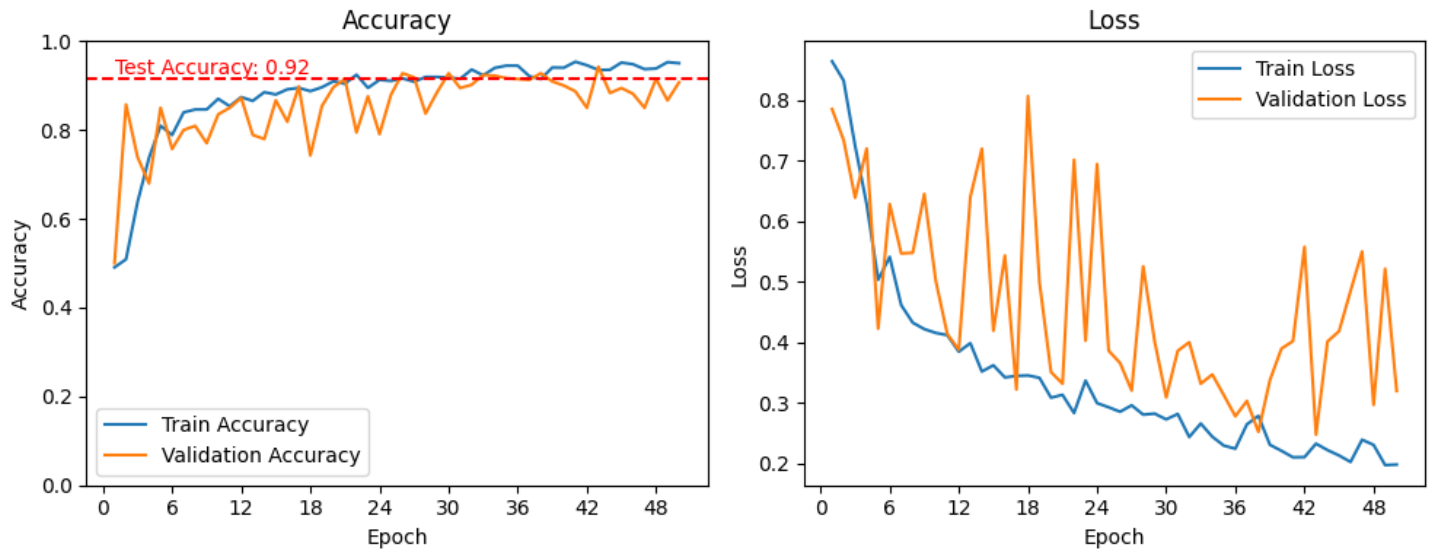

Figure 2: Training the Binary Classification Model. Best Epoch is 43.

2.3 Discussion

To summarizing 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

Please provide a link to your colab notebook.

3 Task 1.2: Multiclass Classification

3.1 Solution

3.1.1 General approach

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

3.1.2 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.

3.2 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.

3.3 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.

3.4 Code

Please provide a link to your colab notebook.

4 Task 2: Model Embeddings and Visualization

4.1 Solution

4.1.1 General approach

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

4.1.2 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.

4.2 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.

4.3 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.

4.4 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.

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

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

6.1.2 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.

6.2 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.

6.3 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.

6.4 Code

Please provide a link to your colab notebook.

Good luck!!