Introduction to Deep Learning
ResNet-D for medical image classification

Nguyen Quang Huy - 22BI13195
Nguyen Tuan Khai - 22BI13202
Nguyen Hai Dang - 22BI13073
Le Linh Long - 22BI13262
Pham Thai Son - 22BI13397

University of Science and Technology of Hanoi

Abstract

In this report, we present the research result on the application of ResNet-D architecture for the task of medical image classification, a crucial aspect in healthcare for early disease detection. The classification of medical images using machine learning/deep learning techniques has been an ongoing area of research for several decades, tracing as far back as the late 20th century. Over the years many new and novel implementations have been discovered to improve the overall accuracy and robustness of the model, some of which achieved results comparable to the diagnosis of board certified physicians [1]. For our work here today, we will also be applying one of these state-of-the-art models to classify lung diseases based on the X-ray scans of patients, and evaluate the results through metrics such as Accuracy, Precision, Recall, and F1 score.

1 Introduction

ResNet (Residual Network), first introduced in 2015 by K.He et al. [7], has quickly become popular due to its state-of-the-art performance: first place in ILSVRC 2015 classification, detection, localization, first place in COCO 2015 segmentation and detection.

ResNet introduced the concept of residual learning, i.e., instead of approximating an unknown function H($\mathbf{x}$) like other machine learning/deep learning models, ResNet learns the residual function $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$. This reformulation overcomes the vanishing, exploding gradient problem and allows training very deep neural network, which used to be prevented because of the degradation problem.

To implement residual learning, ResNet's authors apply shortcut connection using identity mapping, which is often inside a residual block (Figure 1, left image). It consists of two convolution layers with Batch Normalization and ReLU activation function. The output of these layers is then added to the input, which bypasses these layers by a skip connection (or could be understood as going through an identity function). In case the input and output do not have the same dimension, the input will go through a 1x1 convolution layer before the summation. A modification of the basic residual block is bottleneck block (Figure 1, right image). It includes three convolution layers with kernel sizes of 1x1, 3x3, and 1x1. The main idea is that the 1x1 layer decreases the number of channels in the feature map, allowing efficient computation in the 3x3 layer, then restores it back to its original shape.

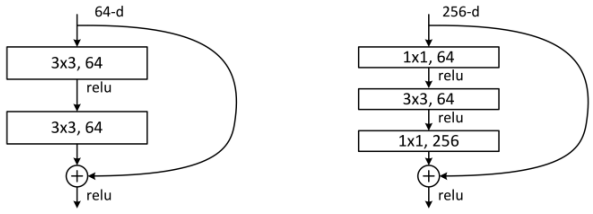

Figure 1: [7] **Left**: Basic residual block. **Right**: Bottleneck residual block

Figure 2 shows the architecture of ResNet-50. The input image first goes through the input stem, which consists of a 7x7 convolution layer and a 3x3 max pooling layer. Both use stride = 2. The output will then go through 4 stages, each has multiple residual blocks. The first block in each stage is called the downsampling block, due to its first convolution layer using stride = 2. At the end of the network, we have an adaptive average pooling, followed by fully connected layers. The operation is the same as normal average pooling, except that we specify the output shape and the stride, kernel size will be automatically chosen.

1. Convolutional Layers:
   - Initial layer with a 7x7 convolution, stride 2, and padding 3; followed by a 3×3 max pooling layer also with a stride of 2 to reduce spatial dimension.
2. Residual Blocks:
   - Basic Block: Consists of two 3x3 convolutional layers with Batch Normalization and ReLU activation. A skip connection bypassses these layers to add the input directly into the output.

- Bottleneck Block: In deeper versions (ResNet-50, 101, 152), it includes three layers: 1x1, 3x3, and 1x1 convolutions, allowing for efficient computation.

3. Global Pooling Layer:

   - Adaptive Average Pooling: This layer reduces each input feature map. It adapts to the output size.
   - Flatten: Converts the feature map into a 1D vector, starting from the first dimension to the last, thus preparing the data for the fully connected layer.

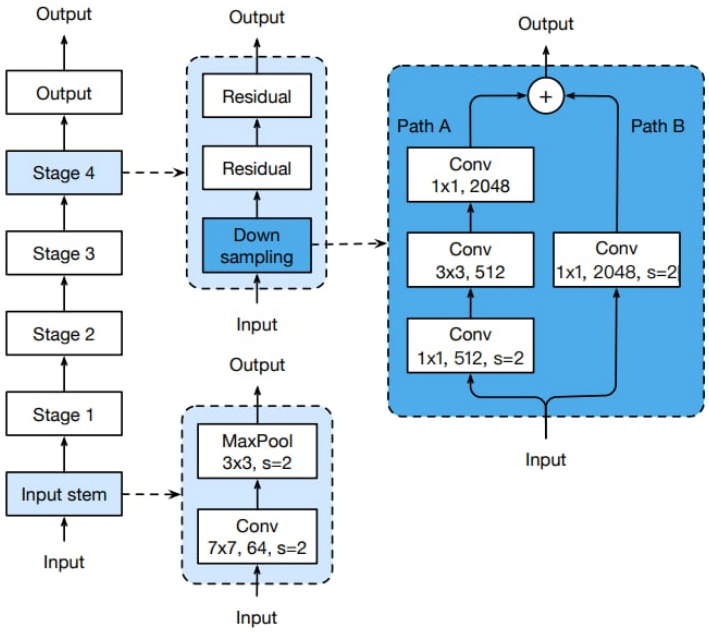

Figure 2: The architecture of ResNet-50. The convolution kernel size, output channel size and stride size (default is 1) are illustrated, similar for pooling layers. [2]

2 ResNet-D Model Architecture in Practice

The ResNet-D model is a variant of the popular ResNet (Residual Network) architecture, introduced in the 2015 paper "Bag of Tricks for Image Classification with Convolutional Neural Networks" by He et al. [2]. Prior to the advent of the ResNet family of models, the baseline structure encountered numerous problems such as information loss during convolution. Several modifications have been introduced to the architecture since, of which the key innovation in ResNet-D is the use of an average pooling layer for downsampling rather than the 1x1 convolution layer used in the original ResNet model. This change was prompted by the observation that the use of 1x1 convolution in the downsampling block meant that the standard ResNet model ignores 3/4 of the input feature maps, leading to a significant loss of information. The ResNet-D architecture addresses this issue by inserting a 2x2 average pooling layer with stride of 1 before the convolution, limiting the information discarded in the downsampling process with little impact to the computational cost [3]. Empirical results have also shown that when combined with various training refinement techniques, it can outperform the original ResNet-50 model by up to 1% in top-1 accuracy on the ImageNet dataset. It has also demonstrated improved performance in other computer vision tasks, such as object detection and semantic segmentation, making it a valuable addition to the ResNet family of models.

ResNet-D introduces several modifications to improve the baseline architecture, specifically to the downsampling block, with an emphasis on reducing information loss and improving feature extraction. First, the initial 7x7 convolution is replaced with three consecutive 3x3 convolutions (introduced in ResNet-C [9]). Researchers also found that adding a 2x2 average pooling layer with stride of 1 before

the convolution serves to prevent information loss, without significant impact to computational cost [8].

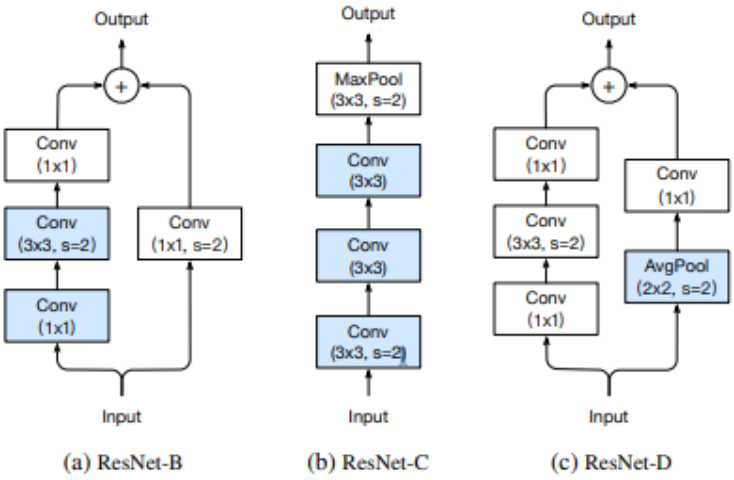

Figure 3: Changes through each model iteration in the ResNet family. Blue highlights the changed/introduced elements.

In our experiment, we will be testing the performance of various ResNet-D model structures (18, 34, 50, 101, 152 and 200 layers) and observing their respective results. For each of these model we replace the last fully connected layer with a new linear layer to adapt to the new classification task with 4 classes. This layer is followed by a softmax activation function [10] applied for each output vectors in order to convert the raw logits into class probabilities.

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

where σ_i = predicted probability of class i^{th}, x = input vector, $x_i = i^{th}$ element in input vector, N = num classes, $x_j = j^{th}$ element for output vector.

Throughout the training process, we monitor the training loss through the use of the Cross Entropy Loss function [10], which compares the model's predicted class probabilities with the ground truth.

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik})$$

where θ = weights, N = num samples in one batch, K = num classes, y_{ik} = ground truth of class k sample i, $x_i = i^{th}$ input vector, $\hat{y}_{ik}$ = predicted probablity of class k sample i.

The model uses an Adam Optimizer with learning rate = 0.001 to update the weights during training, which in turn uses the gradients of the loss obtained during back propagation. Alongside the loss function we also use another checkpoint system to preserve the best model through each training session. Our training set is subdivided into 2 parts, one for training and one for validation. Through each epoch we observe the model accuracy on the validation set, and the model state with the highest validation accuracy is stored in a temp variable. By the end of the training process with each model we store the parameters in a .pth file using *torch.save()* , effectively freezing the model for future use.

3 Training process

3.1 Image Processing + Data Analysis

The figure 4 shows the sample images from the dataset.

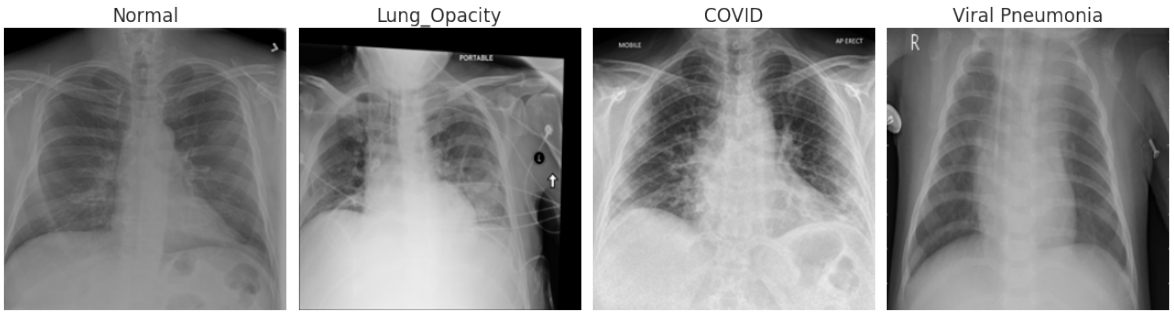

Figure 4: Sample images from the dataset

To give more context surrounding the dataset chosen, the COVID-19 Radiography Dataset [4] was created by a team of researchers from Qatar University, University of Dhaka, Bangladesh along with their collaborators from Pakistan and Malaysia in a collective endeavour with a number of medical doctors. It contains chest X-ray images for COVID-19 positive cases along with Normal and Viral Pneumonia images. The Lung Opacity images in particular do not refer to not a specific condition, but a broad spectrum of symptoms that can all cause an area in the lungs to appear denser or whiter on a chest X-ray or CT scan. This dataset contains a total of 21173 X-ray images, which is unevenly distributed among these 4 classes as show in figure 5.

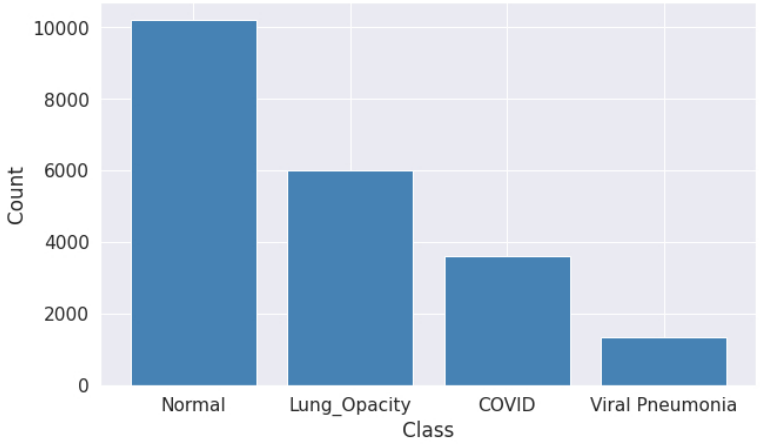

Figure 5: Label Distribution in COVID-19 Radiography Dataset

For classification tasks, the ideal state for the dataset is to have an even or close to even class balance, but when there is a drastic inequity in label distribution, practices such as data augmentation can be judiciously implemented [5]. Some common data augmentation techniques for images include geometric transformations (e.g. rotation, flipping), color space alterations, kernel filters, introducing noise at random, and many more. When some classes are underrepresented in a dataset these techniques can help even out the imbalance with these synthetic samples, thus preventing the model from overfitting on the majority class. To this end we will also be using a number of data augmentation techniques to the training set, and observe the difference in ResNet-D's performance later on. All of the images in the dataset have a dimension of 299x299 with 3 channels, stored in .png format. Since they all have minimal detail, there is little preprocessing needed to prepare them for the model. Our pipeline is as follows:

1. Resize the images to 224x224.
2. Convert the input images to PyTorch tensors, which are multi-dimensional arrays representing the image data.
3. Normalize the image tensor to a predefined mean and standard deviation across each channel.

3.2 Basic model architecture - no optimization methods

Our training procedure involves partitioning the dataset into training and testing sets with varying sizes (90/10, 80/20, 70/30). The training sets are then partition. For each model instance we train for 20 epochs, making sure to preserve the best model state from each session. Our training results on the specific model architectures are as following:

	18 layers	34 layers	50 layers	101 layers	152 layers	200 layers
90/10	97.5909%	**98.2362**%	96.9296%	97.7799%	97.5744%	97.4492%
80/20	**98.1101**%	98.1101	97.2596%	97.9060%	**97.5904**%	97.5667%
70/30	97.8206%	98.0384	**97.4646**%	**97.9788**%	97.5697%	**97.6210**%

Table 1: Model test Accuracy across different test sizes and architectures

When observing the table 1 , we can understand the following:

- Since our input images only has the dimension of 299x299, there is not much information or hidden patterns within the images needing to be extracted. Therefore, as the layer of convolution layers increase within the model architecture, this actually serves to the detriment of the models' performance. This gives us a better understanding of the impact of model complexity on the training process on this particular dataset, and for further experiments we will only be implementing ResNet-18D.
- For ResNet-18D, we observe that overall for the 80/20 split, the test accuracy for the models are consistently better. This ratio ensures the statistical significance in the test results, and at the same time providing sufficient training data for our model to learn from while mitigating overfitting [11].

	precision	recall	f1-score	support
0.0	0.99	1.00	0.99	710
1.0	0.97	0.97	0.97	1230
2.0	0.98	0.98	0.98	2009
3.0	0.97	1.00	0.99	284
accuracy			0.98	4233
macro avg	0.98	0.99	0.98	4233
weighted avg	0.98	0.98	0.98	4233

Figure 6: Classification Report - ResNet-18D on 80/20 dataset

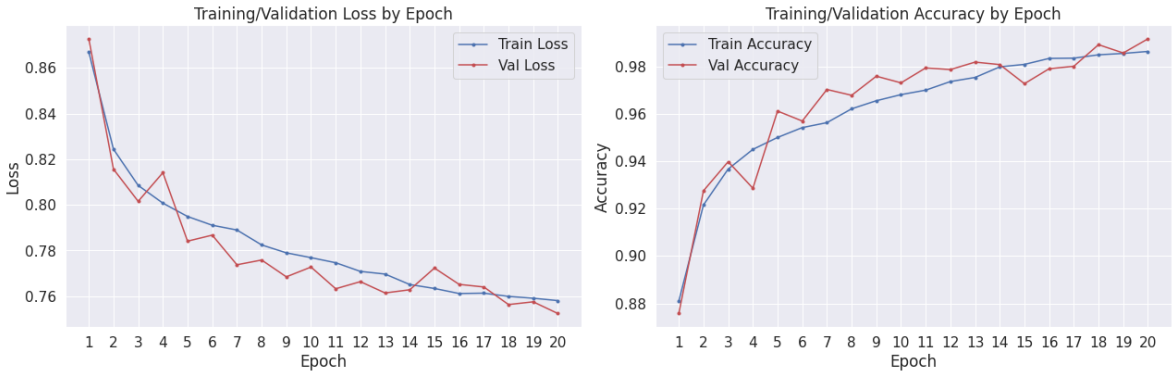

Figure 7: Model Training Process by Epoch

4 Model Improvements

Throughout the process we have been able to highlight several problems with the model and data that can be improved upon, which mainly has to do with the class imbalance and the model's learning method. This section will further delve into the techniques we used to increase the overall performance of the model in sections.

To further elaborate on our methodology, we selected the ResNet-18D model for optimization due to its efficiency and robust performance, and from earlier tests we settled upon using the 80/20 train test split to evaluate our metrics. These optimization methods are implemented independent of one another, meaning we will not be applying 2 to the same model until we have isolated their results and shown that they both are able to contribute to a better model.

4.1 Advanced Learning Rate Optimization

Here, we delve into several ways to improve the model, namely through Learning Rate warm-up and Cosine Learning Rate Decay. Both techniques solve different problems that can arise during training, and what we want to do it see how much of an impact they can create. Learning Rate warm-up and Cosine Learning Rate Decay are both techniques referred to in [2]. In particular, learning rate warm-up is a technique used in training deep learning models where the learning rate starts at a lower value and gradually increases during the initial training or warm-up epochs, as defined by the user. In cases where a large initial learning rate can lead to numerical instability, this helps to accomplish the goal of stabilizing training by allowing the model to initially explore the parameter space liberally before increasing learning rate for faster convergence. In practice, we use a LambdaLR scheduler in PyTorch, where the warm-up epoch number is 10% of the total epochs (following K. He et al. [7]).

Learning rate adjustment is crucial to the training. In a separate training session, after the initial learning rate warmed up, we steadily decreased the learning rate overtime following a cosine curve pattern over a specified number of iterations. The total number of iterations (denoted T_max) can be calculated as such:

$$T\_max = \frac{num\_train\_samples * epochs}{batch\_size}$$

We apply this methodology via the CosineAnnealingLR function, initializing the scheduler to start after the warm up process and update the learning rate iteratively to the last epoch. Implementing these techniques gives us the following:

Warm-up	Cosine	Both
98.5353%	98.4408%	**98.7717%**

Table 2: With LR optimization

From observing these results, we can assess that these changes has made a positive impact on the model performance, and thus are noteworthy additions to the training process when using ResNet-D.

4.2 Data Augmentation

Common practice in data augmentation involves raising the training size to equal samples across each classes. However for our dataset this would be unrealistic, seeing as we would have to create tens of thousands of new images from existing data. So what we will do instead is increase each minority class by X% samples respectively, and observe how our models respond to the change.

Augmentation techniques:

- Random rotation with a partial crop: The selected image is randomly tilted by 6 degrees to the left or right. Since this action introduces some black regions at the corner of the image, we applied center cropping with size of 210x210. This was enough to remove the unwanted regions, as well as preserve the image of the patient's lung.
- Color Inversion: reversing the colors of an image by inverting the brightness of each pixel. This transformation changes white to black, black to white, and reverses shades of grey accordingly.

- Gaussian blur: x,y is the coordinates of kernel and sigma is the standard deviation – influences how significantly the centre pixel's neighbouring pixels affect the computations result. We utilised two separate 3x3 kernel filters in this project, with one having a sigma value ranging from 4 to 5, and the other from 6 to 8 [12].

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Gaussian noise: Noise with a probability density function equal to that of the normal distribution. Appears as random fluctuations in pixel intensity, giving the image a grainy or speckled appearance.

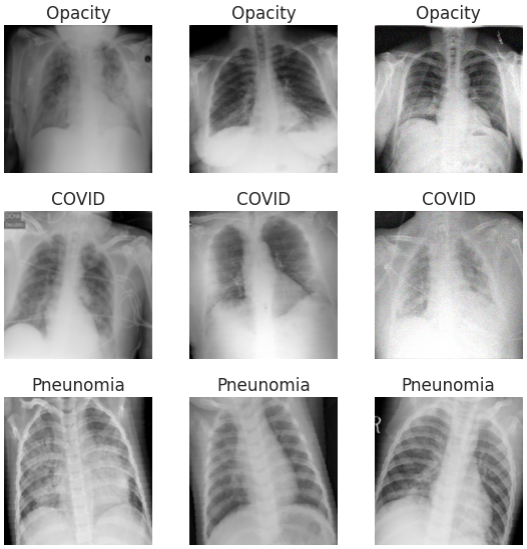

Figure 8: Synthetic Samples from Data Augmentation

% Augmented	10%	20%	30%
Accuracy	97.8975%	**98.3307**%	97.9370%

Table 3: Model Accuracy by % Augmented Samples introduced

The increase in accuracy when creating augmented samples from 10% to 20% suggests an overall improvement in generalization, but by 30% the accuracy decreases. This shows a diminishing return when introducing too many augmented samples in the minority classes, and that there needs to be a balance between quality and quantity. Other tests may help us understand this dynamic further, such as whether the drop in accuracy is statistically significant or if the model's reliability with respect to newer samples has improved, but this method still proved to be successful in addressing the problem of class imbalance.

5 Conclusion

This research has delved into the application of ResNet-D architecture for medical image classification, and we have been able to showcase its efficacy in enhancing feature extraction and improving accuracy. Our results suggested that with proper employment of batch size, architecture, and train/test partitioning, we can reliably improve the performance of the model. Further testing with novel techniques such as Data Augmentation, Learning Rate Warm-up and Cosine Learning Rate Decay also showed a positive impact on the model accuracy and robustness. The results are very promising, and a clear indication of how far the field of computer vision in medical treatment has come.

References

[1] Maryam Naqvi, Syed Qasim Gilani, Tehreem Syed, Oge Marques, and Hee-Cheol Kim (2023). *Skin Cancer Detection Using Deep Learning - A Review.*

[2] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, Mu Li. (2018). *Bag of Tricks for Image Classification with Convolutional Neural Networks*

[3] Sik-Ho Tsang. (2022). *Review - ResNet-D: Bag of Tricks for Image Classification with Convolutional Neural Networks*

[4] Preet Viradiya: *COVID-19 Radiography Dataset.*

[5] Connor Shorten, Taghi M. Khoshgoftaar. (2019) . *A survey on Image Data Augmentation for Deep Learning*

[6] *Relation between Learning Rate and Batch Size*

[7] K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition.* In Proceedings of the IEEE con- ference on computer vision and pattern recognition, pages 770–778, 2016

[8] S. Gross and M. Wilber. *Training and investigating residual nets.* http://torch.ch/blog/2016/02/04/resnets.html.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. *Rethinking the inception architecture for computer vision.* CoRR, abs/1512.00567, 2015

[10] Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York, Springer.*

[11] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva (2018) *Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation*

[12] Aryaman Sharda: *Image Filters: Gaussian Blur*