UCLA
UCLA Electronic Theses and Dissertations

Title
Application of Convolutional Neural Network in Pneumonia Chest Image Classification

Permalink
https://escholarship.org/uc/item/1ff411z0

Author
Park, JungHwan

Publication Date
2022

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

Los Angeles

Application of Convolutional Neural Network in Pneumonia Chest Image Classification

A thesis submitted in partial satisfaction of the

requirements for the degree Master of Applied Statistics

by

JungHwan Park

2022

© Copyright by

JungHwan Park

2022

ABSTRACT OF THE THESIS

Application of Convolutional Neural Network in

Pneumonia Chest Image Classification

by

JungHwan Park

Master of Applied Statistics

University of California, Los Angeles, 2022

Professor Yingnian Wu, Chair

In this application, multiple Convolutional Neural Network models with different complexities are built for finding the most moderate fitting. The bias-variance tradeoff and the ways of tuning the model parameters were introduced to guide the readers who have little knowledge in building neural network models for the image classification problem.

The thesis of JungHwan Park is approved.

Nicolas Christou

Hongquan Xu

Yingnian Wu, Committee Chair

University of California, Los Angeles

2022

List of Figures vii

1 Introduction 1

2 Problem definition 1

3 Methodology 2

3.1 Convolutional Neural Network 2

3.1.1 Convolutional Layer 2

3.1.2 Pooling layer 3

3.1.3 Fully-connected (FC) layer 4

3.2 Depth of CNN 4

3.3 Binary cross entropy 5

3.4 Bias-variance tradeoff 6

3.5 Hyperparameter 7

3.5.1 Number of epochs 8

3.5.2 Learning rate 8

4 Data pre-processing 8

4.1 Data structure 9

4.2 Data augmentation 10

4.3 Normalization 12

5 Models **12**

5.1 Schema A 13

5.1.1 Simple model A 14

5.1.2 Medium model A 15

5.1.3 Medium-complex model A 16

5.1.4 Complex model A 17

5.2 Schema B 18

5.2.1 Simple model B 19

5.2.2 Medium model B 20

5.2.3 Medium-complex model B 21

5.2.4 Complex model B 22

6 Analysis of the result **23**

6.1 Schema A analysis 23

6.1.1 Simple model A analysis 24

6.1.2 Medium model A analysis 25

6.1.3 Medium-complex model A analysis 27

6.1.4 Complex model A analysis 28

6.1.5 Further explanations for schema A analysis 29

6.2 Schema B analysis 30

6.2.1 Simple model B analysis 30

6.2.2 Medium model B analysis 31

6.2.3 Medium-complex model B analysis 33

6.2.4 Complex model B analysis 34

6.2.5 Further explanations for schema B analysis 35

7 Conclusion **36**

8 Python code **37**

9 References **38**

List of Figures

1 Shift of filter across the image 3

2 Revolution of Depth, ImageNet Classification top-5 error (%) 4

3 Training and test errors caused by bias-variance tradeoff 7

4 Pneumonia (left) and normal (right) images from the dataset 9

5 Imbalanced data 10

6 Augmented sample images 11

7 VGG-16 model architecture 13

8 Train and test accuracy graph (left), and train and test loss graph (right) 24

9 Train and test accuracy graph (left), and train and test loss graph (right) 26

10 Train and test accuracy graph (left), and train and test loss graph (right) 27

11 Train and test accuracy graph (left), and train and test loss graph (right) 28

12 Train and test loss for simple to complex model A 30

13 Train and test accuracy graph (left), and train and test loss graph (right) 31

14 Train and test accuracy graph (left), and train and test loss graph (right) 32

15 Train and test accuracy graph (left), and train and test loss graph (right) 33

16 Train and test accuracy graph (left), and train and test loss graph (right) 34

17 Train and test loss for simple to complex model B 35

1 Introduction

Throughout the history, image classification has been developed in many ways as there has been a huge improvement on technology. Specifically, the healthcare industry has been using the radiographical images to diagnose diseases. For example, pneumonia has been diagnosed by digital chest x-ray radiography imaging, and the way it was detected has been changing from the last few decades. The conventional way might be a human diagnosis on the chest film and the patient symptoms, manually implemented one at a time. Nonetheless, it will take a substantial cost and effort, which might not be the best way for the patients in that the early diagnosis of pneumonia is critical to secure the survival rates. In this thesis, one of the well-known computer vision techniques, Convolutional Neural Network, will be employed to deal with the problem.

2 Problem definition

In this application, our goal is to classify the digital chest x-ray radiography images into normal and pneumonia. This is a common binary classification problem, and when it comes to the image data, the machine learning model we can employ is Convolutional Neural Network as it is the most well-known image classification algorithm. For the novice in Convolutional Neural Network, it can be extremely challenging to learn how to build and fit the model to the certain image data. Amazingly, intuition is critical when we conduct model building and fitting and it often comes from the experience in experimentation. The implementation of experimental design varying some hyperparameters might be a descent approach to test our intuition. It is generally known that too simple models can cause underfitting problems, while too complex models can cause overfitting problems. Therefore, it is always helpful to conduct experimentation starting from the very simple model fitting and increase the model

complexity to examine the results to achieve the moderate model for the data. This thesis will be a complete guide for the beginners in Convolutional Neural Network to learn how to experimentally decide the optimal model for the image data.

3 Methodology

In this part, the primary approaches used to drive the results will be introduced below. The methodologies which will be actually employed for the analysis are investigated and the background knowledge for those methodologies are also presented. The programming language I employed throughout the project is Python, and keras is the primary application programming interface used for data preprocessing, model fitting, and visualization.

3.1 Convolutional Neural Network

Convolutional Neural Network [1] is a type of artificial neural network that uses convolution as a part of the network architecture and learns features over a wide range of scales. Along with the fact that it is inspired by the animal visual cortex, it learns neurons with weights and biases during the training process, and it consistently updates them. Convolutional Neural Network is composed of three different layers, which are convolutional layer, pooling layer, and fully-connected layer.

3.1.1 Convolutional layer

The first layer in the network is called convolutional layer [2]. It is where the filter, as known as kernel, moves to the right with a certain stride value until it finishes its calculation for the whole width. It repeats the same process after it moves down to the left until the complete image is passed through. This

filter functions as a weight for the model and it is essentially a 2 dimensional array which features the image. Consequently, the image is converted into numerical values, and the convolutional layer allows the network to capture the suited patterns and interpretations out of the image. In this layer, there is another activation function which maps the negative values to zero in order to preserve positive values, and we will utilize Rectified linear unit (ReLU) activation.

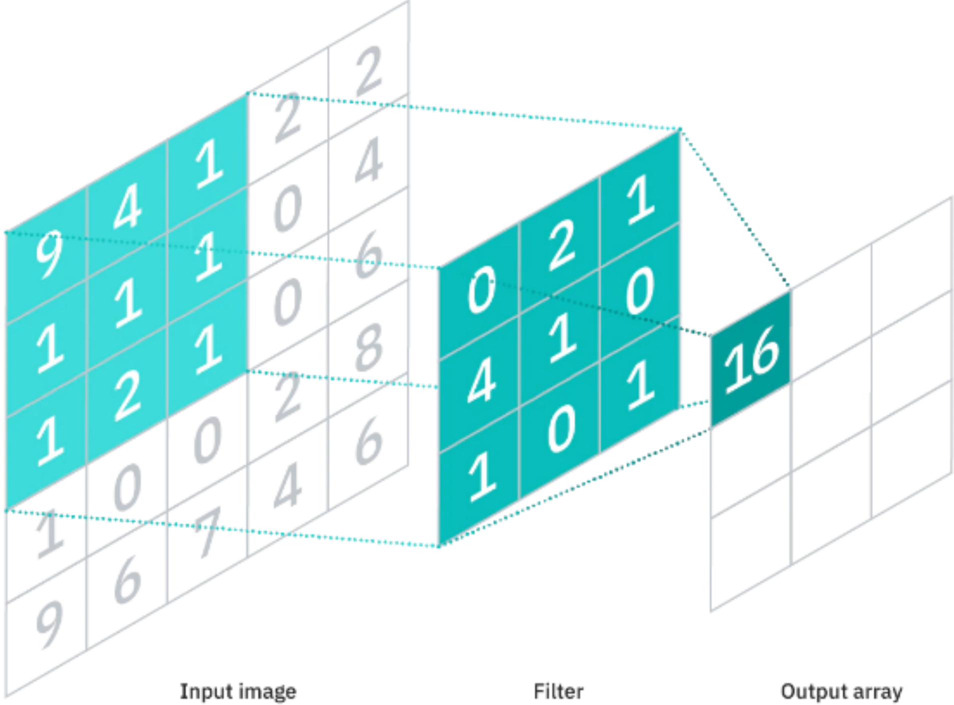

Figure 1: Shift of filter across the image

Source: https://www.ibm.com/cloud/learn/convolutional-neural-networks

3.1.2 Pooling layer

Pooling layer simplifies the output by conducting downsampling, where it reduces the number of parameters that is required for the model to learn. The filter is also employed in the pooling layer as it slides across the whole input applying an aggregation function to the values within the receptive field. There are two types of pooling which are max pooling and average pooling, where the former gets the maximum value while the latter chooses the average value for the output.

3.1.3 Fully-connected (FC) layer

Fully-connected (FC) layer is where the classification is being held, and is the most dense and complex layer in the network. Each node in the output layer is connected to a node in the previous layer, and that is the reason for its name to be called a fully-connected layer. This layer classifies inputs using a softmax function where the input to the fully-connected layer is the output from the last pooling or convolutional layer after flattened. Flatten is equivalent to the vectorization of the matrix or array in calculation.

3.2 Depth of CNN

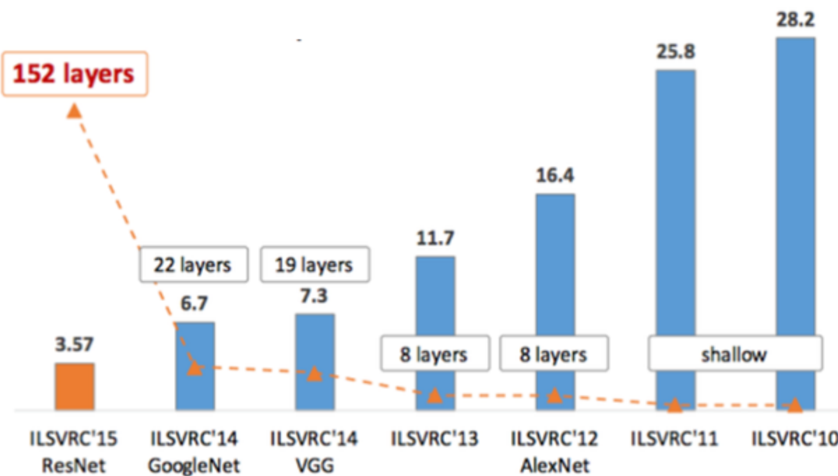

Figure 2: Revolution of Depth, ImageNet Classification top-5 error (%)
Source: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

There are many Convolutional Neural Network architectures which have appeared in the past decade. They were presented to people in a competition like the ImageNet challenge [3], and the winners of each year are the ones many people are now familiar with and like to use for the pre-trained model. One of them is AlexNet, which is the winner in the ImageNet 2012, and it is the early model of the history of Convolutional Neural Network where it is composed of only 8 convolutional layers. VGG-16 performed top-5 error rate in the ImageNet 2015 with 19 layers, while GoogleNet with 22 layers is the winner of the ImageNet 2014. Lastly, the winner in the ImageNet 2015 is ResNet, and it succeeded in

training extremely deep neural networks with 152 layers. It is noticeable from the figure that the bigger number of layers contribute to better results with lower error rates in history. At this point, however, we must not misunderstand that having more layers with their higher complexity necessarily makes the best model. In addition, the number of layers is not the only factor which makes the model complex or simple, but there are some other factors which affect the complexity of the model such as the number of channels or filters, and parameters like learning rate, the number of epochs, etc. Being that said, complete understanding of the depth of layers on top of other hyperparameters and ways of tuning those are critical in image classification experiment.

3.3 Binary cross entropy

Handling the error of the prediction is critical for the optimization algorithm. The key for managing the prediction error is to choose the right loss function. As our model predicts the normal and pneumonia cases encoded into 0 and 1, the binary cross entropy [4] is selected for our loss function. Mathematically, binary cross entropy is calculated by the following formula:

$$Log\ loss\ = \frac{1}{N}\sum_{i=1}^{N} -(y_i\ log(p_i) + (1-y_i)log(1-p_i))$$

$$= -\frac{1}{N}\sum^{N}\sum^{M} y_{ij}\ log(p_{ij})$$

Here, N is the number of rows, which is the length or our test set, and M is the number of classes. It is the negative average of the log of the corrected probabilities, where it penalizes the probabilities based on the difference between the actual and expected values.

In our problem, supposing that the prediction for pneumonia on an image is 0.999, the following is the cost with just one picture if our prediction is wrong and the actual image was normal:

$$-2\ log(1-0.999)\ =\ 13.816$$

While the binary cross entropy cost is calculated by dividing this number by the number of test images, which is 624. The calculation will be:

$$-2\ \frac{log(1-0.999)}{624} = 0.022$$

This value explains how volatile the binary cross entropy can be if the dataset is hard to predict. For the further analysis, it might be crucial to consider the base rate logistic cost. The naïve prediction probability is calculated by dividing the normal cases by the total number of train dataset. In other words, if we need to make predictions on whether it is normal, we argue that the probability that any given image which is randomly chosen is simply the proportion of the normal data. In our train dataset, there are 3875 pneumonia cases while there are only 1341 normal cases, so the calculated naïve base rate would be 0.25709. In this case, the logistic cost will be 1.14, calculated by:

$$-2\big(0.25709 * log(0.25709) + (1-0.25709) * log(1-0.25709)\big) = 1.14$$

If the proportion of normal is much different than the proportion of pneumonia, then the base rate for the logistic cost might come down. Therefore, we can compare this base cost to our later analysis. In other words, even if we see the volatility on the plot of the train logistic loss, if that is far less than the base rate cost, it is not necessarily a bad result.

3.4 Bias-variance tradeoff

As mentioned previously, the major interest is to experiment with different complexity of the model from tuning the number of parameters or hyperparameters. In general, the more the number of parameters that model carries, the more complex it becomes. As the model gets complex, train accuracy increases and train loss decreases. On the other hand, the test loss decreases before it reaches the optimal model complexity and increases again along with the change in model

flexibility. That optimal point is where the bias-variance tradeoff [5] happens. The low variance and high bias with low complexity of model gradually switch to high variance and low bias as the model complexity gets higher. The mathematical formula for the expected prediction error contains both the variance and the squared bias, and our goal is to find the model with its optimal variance and the squared bias.

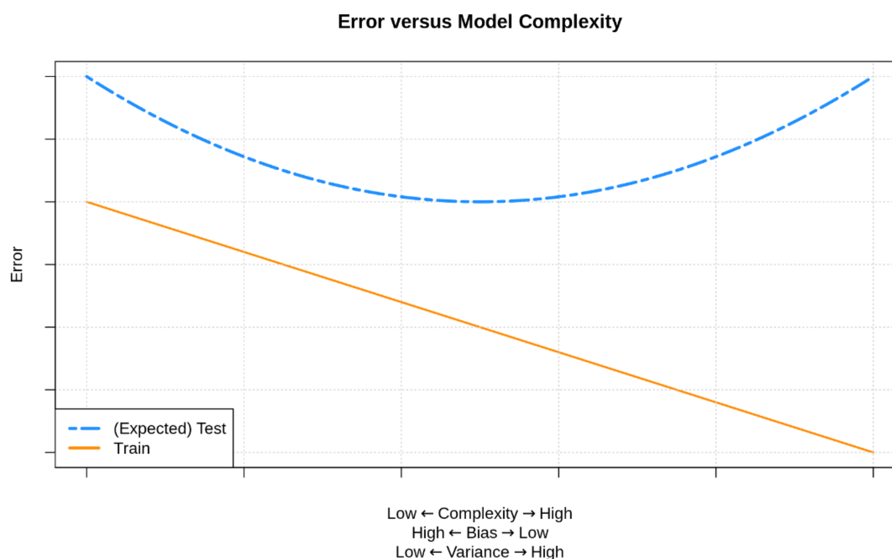

Figure 3: Training and test errors caused by bias-variance tradeoff

Source: https://daviddalpiaz.github.io/r4sl/biasvariance-tradeoff.html

3.5 Hyperparameter

In machine learning, controlling hyperparameter [6] is the key for achieving the highest performance of the model. Other than simply controlling the number of parameters in the model, tuning and adjusting hyperparameter makes significant changes in the model complexity and the optimization. Investigation for the optimal hyperparameters or the experiment by trying out some choices is critical for the beginners to attain the strong model. In this section, only two hyperparameters out of many others are introduced as they are tuned for the schema in this experiment. Other hyperparameters such as batch size and activation functions are consistent

throughout the experiment. The batch size is fixed for 32 and ReLU activation is employed as it is easy to train the model and it often achieves decent performance.

3.5.1 Number of epochs

By setting the number of epochs, we can manage the number of iterations for training or test sets to fit the neural network model. Like other parameters, the number of epochs must be determined depending on the data. The epoch numbers I employed are 100 and 200 for two different schemas, and it was determined from the data and the computational cost considering GPU performance of my personal device.

3.5.2 Learning rate

A neural network utilizes the optimization algorithm called stochastic gradient descent algorithm, which updates the weights from backpropagation. The learning rate is the step size for updating these weights used for that optimization search process. The employment of the learning rate for the model is determined considering the choice of optimizer.

In this application, we are using a fixed learning rate for Adam and a decaying learning rate for RMSprop optimizer for each schema. In general, the moderate learning rate makes the loss gradually decrease in U shape so it converges to 0 while the low learning rate can cause the linear decrease in loss and the high learning rate might cause it to get stuck at a certain loss.

4 Data pre-processing

Among the list of image classification algorithms, Convolutional Neural Networks generally require less pre-processing compared to the others, because it automatically learns to optimize

kernels. Nevertheless, exploring and understanding the data must be preceded by modeling and analysis. For the image classification problem, data is a collection of image files, not a relational database like a table. Therefore, generally it is uncommon to find cases dealing with missing values, but rather what is usual is to conduct a different type of pre-processing. In this section, we will explore the data structure and how they were pre-processed before the modeling.

4.1 Data structure

The original chest X-ray images are initially screened data retrieved from the website, Mendeley Data, and there are 5,863 image files [7] in two categories, pneumonia and normal. On the pneumonia chest X-ray image [8], we can see the opacity as seen in white. However, it can be wrongly interpreted from the other issues like lung scarring or the congestive heart failure, as well as the fact that pneumonia is not always seen on x-rays which might cause false negative cases.

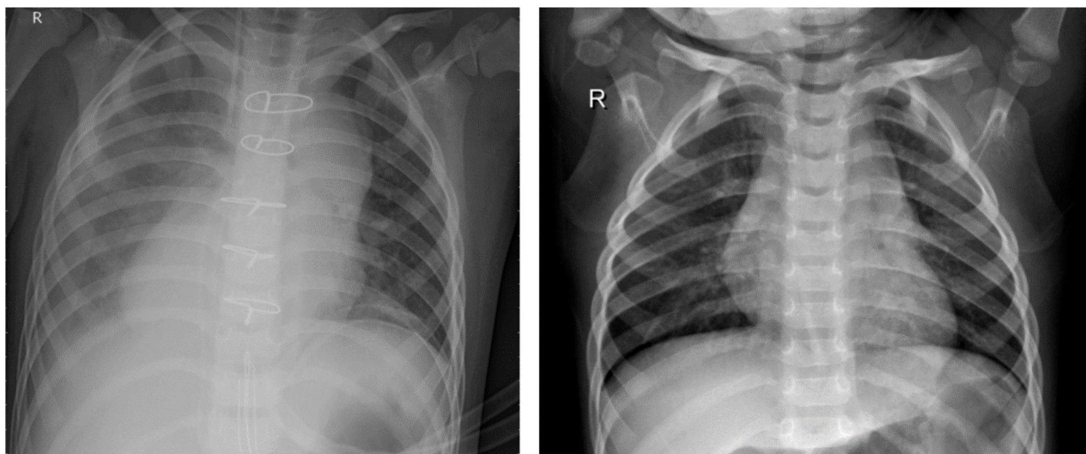

Figure 4: Pneumonia (left) and normal (right) images from the dataset

After the data exploration, obviously the data is imbalanced as there are much more pneumonia chest images than the normal. To tackle this problem, more data must be substituted for the normal

chest x-ray images, but collecting data is always hard or expensive. Insufficient data may cause overfitting when we fit the model, and the outcome of overfitting can be invalid or can even mislead the conclusion.

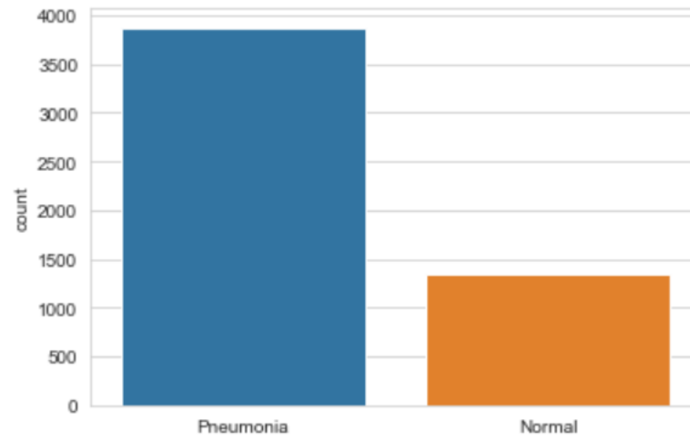

Figure 5: Imbalanced data

4.2 Data augmentation

To handle the previous problem, the technique called data augmentation was conducted to solve the issue of "lack of density." Data augmentation [9] is a technique used to enlarge the amount of data by creating randomly transformed copies of the existing data. There are numerous types of variations to augment the data such as horizontal flipping, vertical flipping, random crops, rotation, color jitters, grey scales, etc. Conducting data augmentation can be easily done by using the 'ImageDataGenerator' class in keras.

There is no fixed transformation which must be applied to certain images because it is merely for oversampling.

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range = 30,
        zoom_range = 0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip = True,
        vertical_flip=False)
```

The arguments in 'ImageDataGenerator' I picked for the augmentation are a horizontal flipping, random shift of vertically by 10 percent of the image height, horizontally by 10 percent of the image width, image zoom by 20 percent of the images, and the 30 degrees rotation of the images. Some samples of the augmented images are displayed on the figure.

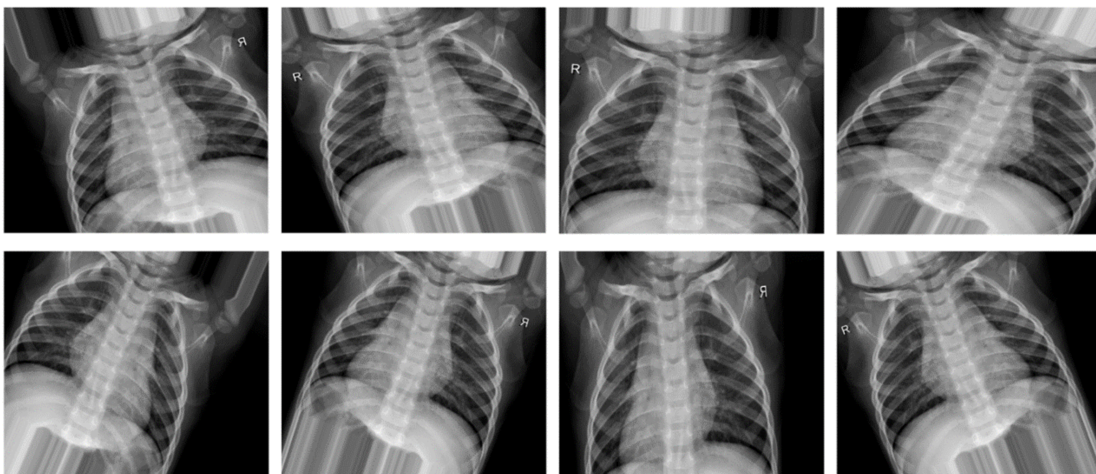

Figure 6: Augmented sample images

4.3 Normalization

Normalization [10] is the most critical data pre-processing technique, which is used to standardize data for image classification tasks. Skipping the normalization step can affect the learning speed and the issue of propagating gradients. As it makes different sources of data inside the range from 0 to 1, it brings down the number of training epochs which is required to train the model. The image data for computation are the pixel values between 0 and 256, and hence the values are divided by 255 for the normalization.

```
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255
x_test = np.array(x_test) / 255
```

5 Models

There are many popular models for image classification such as AlexNet, VGG16, VGG19, GoogleNet, Inception v2, Inception v3, ResNet, etc. Also, there are quite a few APIs offering pre-trained versions of those models trained on the images from the ImageNet database. In our pneumonia chest x-ray classification experiment, I will utilize a simply modified version of VGG-16 [11] for a very complex model and reference it to build other simpler models. VGG-16 is a very deep Convolutional Neural Network which consists of 16 layers. As shown in the figure, the architecture of VGG-16 are structured with several blocks where each block contains two or three convolutional layers with a following pooling layer with corresponding filter which increase its size exponentially from 64 to 512. For the classification, VGG-16 contains three fully-connected layers, each containing 4096, 4096, and 2 units. As there is no solid solution for the depth of the network we must employ for the certain data, there is few guidelines that explains the application of the architecture for the network in accordance with the input data.

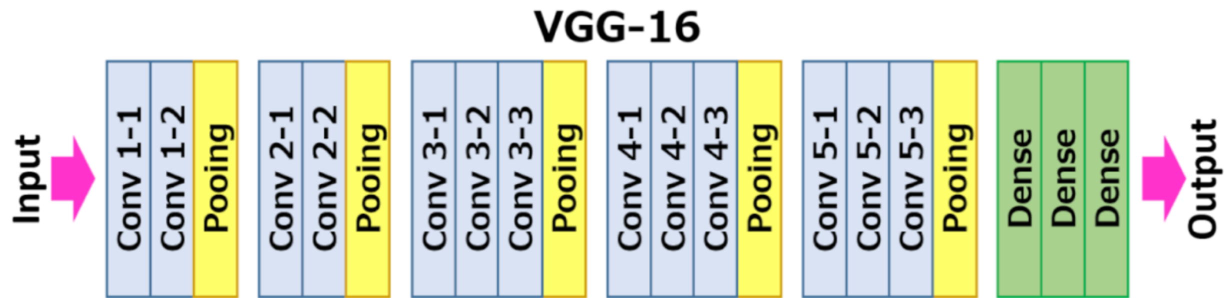

Figure 7: VGG-16 model architecture

Source: https://imgur.com/gallery/AGYuLL1

Therefore, the most reasonable bet for the beginners is to configure the network structure by experimentation. In this section, I designed two schemas with the different hyperparameters and corresponding four different models named as simple, medium, medium-complex, and complex. Again, the complex model I designed was referenced from VGG-16 architecture, but in a way more simplified version. The way the four models were tuned was varying the number of parameters. The parameters in Convolutional Neural Network act as training weights, and specifically weight matrices which helps boost the power of the model prediction. Therefore, the number of parameters serves as a key control factor for the model complexity.

5.1 Schema A

In this section, the first schema for the experiment with the models having certain hyper parameters will be explained, and we will call it schema A. There will be one more schema in the later chapter. For schema A, input shape for the image was shaped as (224, 224, 3). It is very general to use 224 for the height and width for the image processing although it does not cause dramatic change for the results if it is not too small or not too big. The depth element of the input shape is generally set to 3 for the RGB image which is for the natural images, or 1 for the gray scale images. This schema, the 3 will be used as our data is composed of a mixture of RGB and gray scale images. We will

use 32 for the batch size and 200 epochs for the fair model fitting. However, 200 epochs are not necessarily big considering there is some 1000 epochs performed to fit the data regarding the essence of the data. In addition, the Adam optimizer was employed with the default learning rate.

5.1.1 Simple model A

As can be observed on the code snippet, the 3 stacks of 2D convolution layers (Conv layers) with 4 to 8 filters and ReLU activation followed by max-pooling layers was built along the code. Those blocks are followed by 2 blocks of some fully-connected layers which have their own activation functions.

```
model = Sequential()
model.add(Conv2D(4, (3,3), strides = 3, padding = 'same', activation =
'relu', input_shape = (224, 224, 3)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(8, (3,3), strides = 3, padding = 'same', activation =
'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 4, padding = 'same'))
model.add(Conv2D(8, (3,3), strides = 3, padding = 'same', activation =
'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 4, padding = 'same'))

model.add(Flatten())
model.add(Dense(units = 32, activation = 'relu'))
model.add(Dropout(0))
model.add(Dense(units = 2, activation = 'sigmoid'))
```

The number of parameters [12] in each convolutional layer is calculated by the formula:

$$(shape\ of\ width\ of\ the\ filter \times shape\ of\ height\ of\ the\ filter * number\ of\ filters\ in\ the\ previous\ layer + 1) \times number\ of\ filters$$

For example, the first Conv layer in our simple model, the number of parameters is calculated by $(3 \times 3 + 1) \times 1$, which is 40. The number of the dense layer is calculated by the formula:

$$(current\ layer\ neurons\ c \times previous\ layer\ neurons\ p) + 1 \times c$$

Therefore, the number of parameters in the first dense layer is 32$*$8 + 1$*$32, which is 288, given that the previous flatten layer has 8 nodes and c value is 32. Nevertheless, keras, one of the Python application programming interfaces, supports the model summary function to show the total trainable number of parameters and the parameters for each layer. Our simple model contains 1,386 total trainable parameters according to the summary output.

5.1.2 Medium model A

The medium model is composed of three stacked blocks of 2D convolution layers with the same ReLU activation and 16 to 32 filters, followed by pooling layers. Those blocks are also followed by the fully connected layers. Our medium model starts the layer with 16 kernels to 32 for the high complexity, where it sticks with the max-pooling size of (2,2). Other than the increased number of kernels and the number of convolution layers, the added number of fully-connected layers adds significant number of parameters due to the essence of its mathematical formula.

Regarding the model summary output, our medium model contains 314,258 total trainable parameters, where it enlarges the complexity by having 227 times the number of parameters of the simple model.

```
model = Sequential()

model.add(Conv2D(16, (3,3), strides = 1 , padding = 'same' , activation =
'relu' , input_shape = (224,224,3)))

model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))

model.add(Conv2D(32, (3,3), strides = 1, padding = 'same' , activation =
'relu'))

model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))

model.add(Conv2D(32, (3,3), strides = 1, padding = 'same' , activation =
'relu'))

model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))

model.add(Conv2D(16, (3,3), strides = 1, padding = 'same' , activation =
'relu'))

model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))

model.add(Flatten())

model.add(Dense(units = 128, activation = 'relu'))

model.add(Dropout(0))

model.add(Dense(units = 2, activation = 'sigmoid'))
```

5.1.3 Medium-complex model A

The structure of our medium-complex model again starts with 4 stacked blocks of 2D convolutional layers with 16 to 128 kernels for each layer and followed fully-connected layers. Every convolutional layer also come with the ReLU activation layers and the max-pooling layers. There are four fully-connected layers which have 128, 256, 25, and 32 neurons each which will end up adding tremendous number of parameters to the model.

The model summary output for medium-complex model demonstrates that the total number of trainable parameters is 2,563,970, which is more than eight times that of the first medium model.

```
model.add(Conv2D(16, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(MaxPool2D((2,2), padding = 'same'))
model.add(Conv2D(32, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(MaxPool2D((2,2), padding = 'same'))
model.add(Conv2D(64, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(MaxPool2D((2,2), padding = 'same'))
model.add(Conv2D(128, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(MaxPool2D((2,2), padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 256, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 256, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 2, activation = 'sigmoid'))
```

5.1.4 Complex model A

Sticking with having four stacked blocks of 2D convolutional layers with the same ReLU activation layers and 16 to 128 filters with the max-pooling, our complex model increases its complexity by adding more fully-connected layers up to six, which adds another huge number of parameters. According to the model summary output, our complex model carries the total of 8,900,610 trainable parameters, which does more than triple the number of total parameters of our medium-complex model.

```
model = Sequential()
model.add(Conv2D(16, (3,3), strides = 1, padding = 'same' , activation = 'relu', input_shape = (224,224,3)))
model.add(MaxPool2D((2,2), strides = 1 , padding = 'same'))
model.add(Conv2D(32, (3,3), strides = 1, padding = 'same', activation = 'relu', input_shape = (224,224,3)))
model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))
model.add(Conv2D(64, (3,3), strides = 1, padding = 'same', activation = 'relu', input_shape = (224,224,3)))
model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))
model.add(Conv2D(128, (3,3), strides = 1, padding = 'same', activation = 'relu', input_shape = (224,224,3)))
model.add(MaxPool2D((2,2), strides = 1, padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 256, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 256, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 32, activation = 'relu'))
model.add(Dropout(0.0))
model.add(Dense(units = 2, activation = 'sigmoid'))
```

5.2 Schema B

In this section, another experiment, called schema B, will be introduced. With the same structure of the data and, a few hyper parameters such as filter number, epoch number, learning rate, and the optimizer has been changed from the ones in schema A. In this schema, the epoch number employed will be 100, which is half of what we used for schema A, and the learning rate will be

coded to decay over the epochs until it reaches its minimum value of 0.000001. The optimizer is changed from Adam to RMSprop, and the input shape was set different that we now use (150, 150, 1). The overall structure of the model structures will be the same, whereas some dropouts will be added.

5.2.1 Simple model B

Our simple model for the second schema consists of 2 stacked blocks of 2D convolution layers with ReLU activation and 4 to 8 filters with the following pooling layers. Two fully-connected layers are added after the convolutional layers and a small dropout value of twenty percent of neurons (0.2) is added to prevent overfitting.

The summary output for our simple model indicates that it contains 92,793 total trainable parameters, which is a lot larger than that of the simple model in the previous schema.

```
model = Sequential()
model.add(Conv2D(4, (3,3), strides = 1, padding = 'same', activation =
'relu', input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(8, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 8, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1, activation = 'sigmoid'))
```

5.2.2 Medium model B

Our medium model is composed of three blocks of 2D convolutional layers with ReLU activation and 16 to 32 filters followed by pooling layers with two fully connected layers.

The dropout values of 0.1 and 0.2 are used to prevent overfitting and the only 1 neuron is connected for the last fully-connected layer in this schema.

The model summary output illustrates that the total number of trainable parameters for our medium model is 383,937, which is more than four times the number of parameters as in the very simple model.

```
model = Sequential()
model.add(Conv2D(16, (3,3), strides = 1, padding = 'same', activation =
'relu', input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(32, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(32, (3,3), strides = 1, padding = 'same', activation =
'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 32, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1, activation = 'sigmoid'))
```

5.2.3 Medium-complex model B

The network configuration of our medium-complex model contains three 2D convolutional layers with 64 to 256 filters and denser fully connected layers. The model summary output demonstrates that the total number of trainable parameters is 1,245,313, which is more than three times that of the medium model.

```
model.add(Conv2D(32, (3,3), strides = 1, padding = 'same', activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(64, (3,3), strides = 1, padding = 'same', activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(64, (3,3), strides = 1, padding = 'same', activation= 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(128, (3,3), strides = 1, padding = 'same', activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(256, (3,3), strides = 1, padding = 'same', activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1, activation = 'sigmoid'))
```

5.2.4 Complex model B

Our last model is the most complex model which consists of four stacked blocks of 2D convolution layers with each ReLU activation layer and 64 to 512 filters with the pooling and fully connected layers. In this schema, the complexity of the model has been controlled by adding exponentially increasing number of filters in 2D convolutional layers, while the density of fully-connected layers was controlled for the models in schema A. Our complex model contains 14,659,457 parameters, which is about twelve times as many as our second medium model, and about 158 times as many as our simple model.

```
model.add(Conv2D(64 , (3,3) , strides = 1, padding = 'same',
activation = 'relu', input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(128, (3,3), strides = 1, padding = 'same', activation
= 'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Conv2D(256, (3,3), strides = 1, padding = 'same' ,
activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2 , padding = 'same'))
model.add(Conv2D(512, (3,3) , strides = 1, padding = 'same',
activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2), strides = 2, padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 256, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1, activation = 'sigmoid'))
```

6 Analysis of the result

After building four different models for each schema, I will use the same method to compile them for further analysis. Compiling needs to be performed after the model building and before the model fitting.

```
model.compile(optimizer = "Adam", loss = 'binary_crossentropy', metrics = ['accuracy'])
```

The 'compile' method was employed by specifying the arguments such as optimizer, loss, and metrics.

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2,
                          verbose=1,factor=0.3, min_lr=0.000001)

model.fit(datagen.flow(x_train,y_train, batch_size = 32) ,epochs = 40 , validation_data =
                          datagen.flow(x_test, y_test) ,callbacks = [learning_rate_reduction])
```

For schema B, the learning rate is not fixed at the default value but it will be controlled to prevent wild fluctuation in the testing loss. Using the class called 'ReduceLROnPlateu', we will reduce learning rate by a factor of 0.3 once there is no improvement in validation accuracy for 2 epochs, and we will set the learning rate minimum as 0.000001. Then, we will assign it to the callback argument in our model fit function as an action we want to perform in a specific instance of training.

It was not possible to train over too many epochs since the train set was large and my GPU memory was not enough, and hence I used 100 and 200 epochs for each schema. The batch size will be 32 so that the number of steps per epoch is not too big. Hence, it will be 163 as it is calculated by dividing batch size from the size of the train set.

6.1 Schema A analysis

In this section, the analysis for schema A will be illustrated with the history output for the model fitting over 200 epochs with the default learning rate with Adam optimizer. The analysis will be based on the plots and confusion matrices.

6.1.1 Simple model A analysis

As our validation data is very small as it contains only 16 images, I used test data, which is 624 images in length, for the plotting. The displayed plot of train and test accuracy demonstrates that the train accuracy drastically goes up after a few epochs and it stays at about 0.95, while the test accuracy fluctuates in the range between 0.4 to 0.8. The average train accuracy of the 190th to 200th epochs is about 0.95.

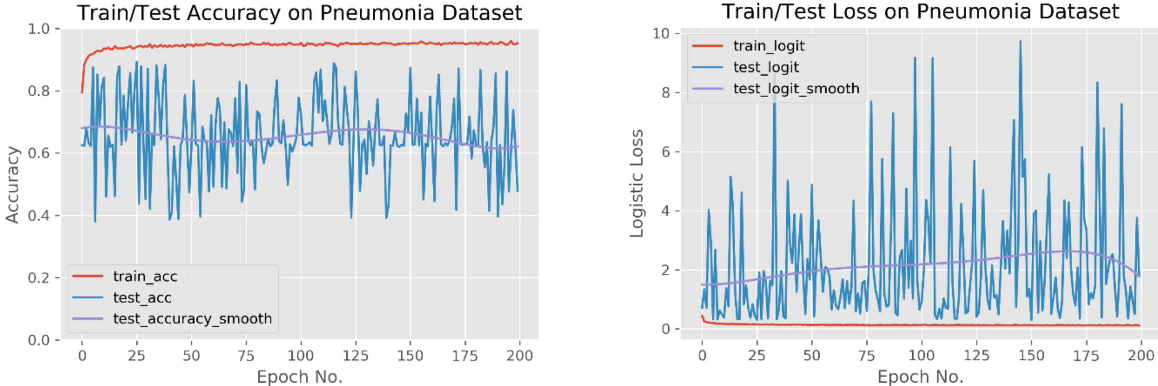

Figure 8: Train and test accuracy graph (left), and train and test loss graph (right)

As seen on the figure of train and test loss on pneumonia dataset, our train loss, after a couple of epochs, stays at about 0.1, but the test loss fluctuates a lot in a range between 0.2 to 9.8, showing the extreme spikes across the whole epochs.

Actual \Predicted	Pneumonia	Normal
Pneumonia	211	23
Normal	346	44

This means that our simple model does not really predict well for the test data. Also, as mentioned in the methodology section, we are using binary cross entropy for our loss function and its mathematical formula can explain the volatility if the prediction is wrong with the high confidence. The confusion matrix was created to show the classification result of the test set. The true positive, false negative, false positive, and true negative values are 211, 23, 346, and 44. The true negative value is low as our simple model might not be capable of classifying the actual normal images as normal, which makes our accuracy as low as 0.41. Moreover, the false positive value is high as well as the low true negative value, and hence the specificity is as low as 0.11. On the other hand, the sensitivity of our simple model is comparably high, which is 0.9, because the true positive and false negative values are fair. The test results of good sensitivity but poor specificity means that our simple model identifies the images with pneumonia, but it also classifies normal as pneumonia. This result is likely to be explained by the imbalance of the data, where the number of pneumonia images are about triple the number of normal images. The precision, calculated as true positive over true positive added by the false positive, is about 0.38. The low precision indicates that our simple model is about 38 percent correct out of the total predicted cases for pneumonia. F-1 Score, a harmonic mean of precision and sensitivity, for our simple model is 0.53.

6.1.2 Medium model A analysis

Our medium model with 314,258 trainable parameters results in better test accuracy as shown in the figure. It fluctuates, but far less than what it was shown for our simple model, within the interval of 0.8 and 0.85. The average test accuracy for the last 10 epochs was around 0.89. Meanwhile, the train accuracy, compared to that of our simple model, has been improved a little as it stabilized its value around 0.97. Our medium model also showed an improvement in the test loss in that the volatility has been decreased while it still fluctuates between 0.22 to 0.55. The train

loss dropped quickly in the early epoch and slowly decreased until it reached around 0.09 at the late epochs. As can be seen from the confusion matrix, the true positive, false negative, false positive, and true negative values for our medium model are 67, 167, 123, and 267. The accuracy has been increased from 0.41 to 0.53 as the sum of true negative and true positive values increased.

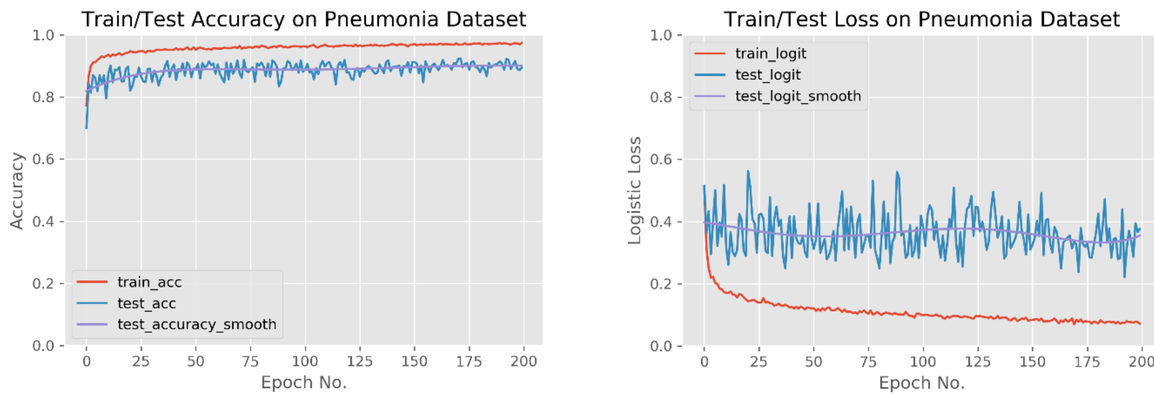

Figure 9: Train and test accuracy (left), and train and test loss graph (right)

Unlike our simple model, our medium model gives higher true negative value and lower true positive value. Hence, compared to the simple model, our medium model showed a better specificity and a worse sensitivity, which are 0.68 and 0.29 each. That means, the medium model is better in ruling out the normal images, but not the pneumonia. The precision decreased to 0.35, but not dramatically, because it was 0.38 for the last model. This number demonstrates that our medium model is about 35 percent correct out of the total predicted cases for pneumonia. F-1 Score, a harmonic mean of precision and sensitivity, for our simple model is 0.32.

Actual \Predicted	Pneumonia	Normal
Pneumonia	67	167
Normal	123	267

6.1.3 Medium-complex model A analysis

The train and test accuracy plot of our medium-complex model looks essentially similar to that of our medium model. The test accuracy, showing a little volatility over the complete epochs, was about 0.9 in average over the last 10 epochs, while the train accuracy almost stayed at the value around 0.98. As can be observed in the graph, the test loss for our medium-complex model is showing the less volatility compared to the previous models, although there is a spike at the 173rd epoch.

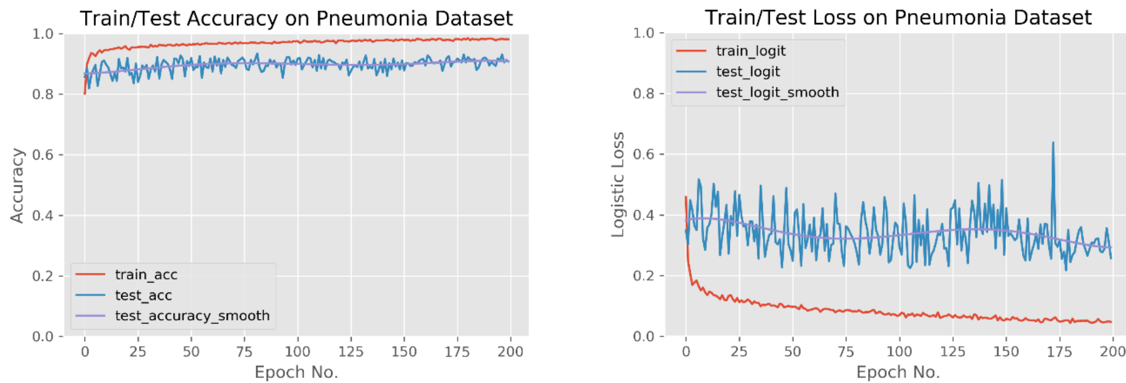

Figure 10: Train and test accuracy (left), and train and test loss graph (right)

The overall loss values have been decreased as the smoothed spline shows that the values are fluctuating around 0.32. The train loss showed a consistent overall decrease until it reaches the value around 0.06 at the late epochs.

From the confusion matrix, it is demonstrated that true positive, false negative, false positive, and true negative values for our medium-complex model are 72, 162, 114, and 276. Those values obviously show that our medium-complex model has the best classification power. The accuracy of the medium-complex model is 0.56, which is the highest among the four models. Moreover, the specificity of this model is 0.71, which is also the highest among the four models, while the sensitivity is 0.31 which is higher than that of our medium model. The precision for our medium-complex model is 0.39, where it is the highest among the four models, and F-1 score is 0.34, which

is higher than that of our medium model.

Actual \Predicted	Pneumonia	Normal
Pneumonia	72	162
Normal	114	276

6.1.4 Complex model A analysis

As can be seen on the graph, train and test accuracy for our complex model look just like the one we could observe for our medium-complex model. The train accuracy stayed at the value around 0.98, while the test accuracy was about 0.9 in average over the last 10 epochs, fluctuating a little over the whole epochs.

While the train loss stayed at the value around 0.07 for the last 10 epochs, the test loss of our complex model was fluctuating around 0.4 which is higher than the test loss value for our medium-complex model. Its volatility is a little worse than that of our medium-complex model as well.

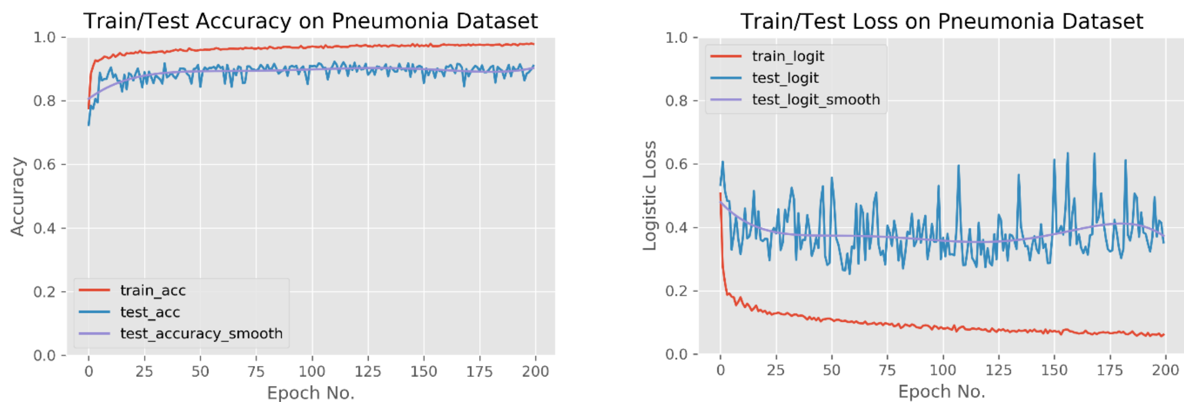

Figure 11: Train and test accuracy (left), and train and test loss graph (right)

According to the confusion matrix, true positive, false negative, false positive, and true negative values for our medium model are 76, 158, 126, and 264. The accuracy of our complex model is 0.54, while sensitivity and specificity are 0.32 and 0.68. The precision and F-1 score are 0.38 and

0.35.

Actual \Predicted	Pneumonia	Normal
Pneumonia	76	158
Normal	126	264

6.1.5 Further explanations for schema A analysis

Throughout the whole processes of model fitting, we recognized that our medium-complex model was showing the best results among the four models. The test loss and the train loss on the figure were evaluated after the model fitting by a simple python API, 'model.evaluate' method, which generates single evaluated value for each input.

As can be seen in the figure, as the complexity of the four models increases, the test loss decreases but it goes back up again for the fourth model. It is because there was bias-variance tradeoff happened at a certain point of model complexity somewhere between our medium-complex model and complex model. That means, we cannot argue that our medium-complex model is the exact point where the bias-variance tradeoff has occurred. The train loss generally tends to decrease consistently as the model complexity increases, but what is interesting was that the train loss of our complex model was

slightly higher than that of medium-complex model. This does not mean that the whole justification for the bias-variance tradeoff is invalid, but sometimes the subtle randomness in outputs present in machine learning.

The earlier explanation for the binary cross entropy as our loss function, it explains the volatility in the majority of test loss values over the epochs. It also explains the wrong prediction presented as results on the confusion matrices. However, the fact that the average range of fluctuation of our test loss over the epochs are merely 0.2 can be interpreted that the model is functioning a lot better than the naïve rate,

which was introduced in the earlier methodology section with the base rate calculation.

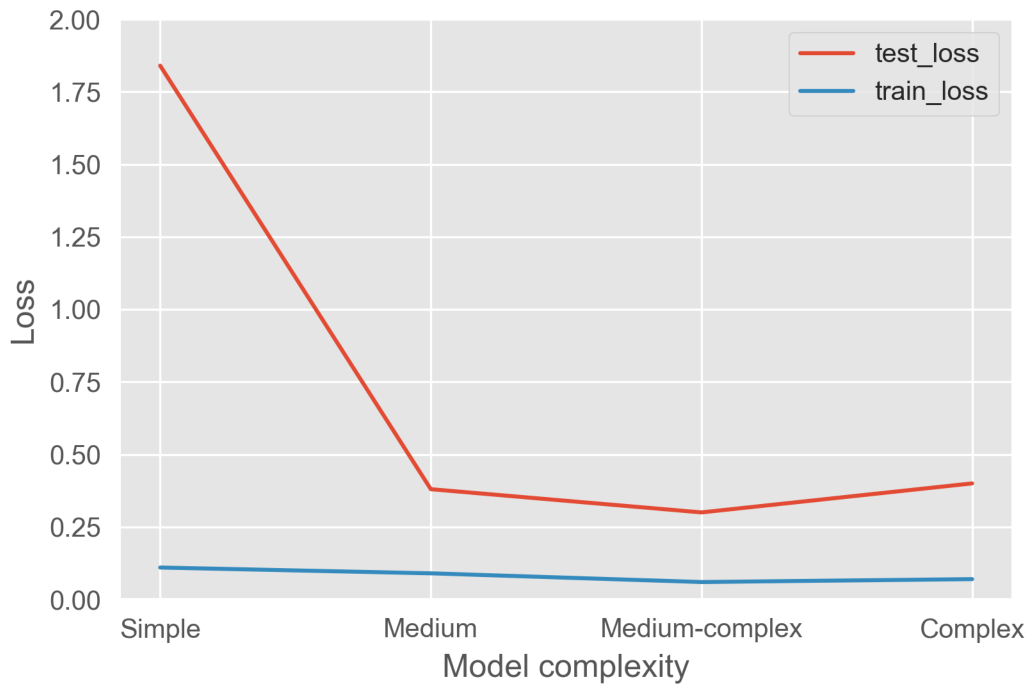

Figure 12: Train and test loss for simple to complex model A

6.2 Schema B analysis

In this section, the analysis for schema B will be described with the history output for the model fitting over 100 epochs with the decaying learning rate with RMSprop optimizer [13]. The analysis will be based on the plots and confusion matrices as well.

6.2.1 Simple model B analysis

As can be seen on the figure, even it was plotted for the smaller number of epochs, the test accuracy and loss show more stable compared to the ones for our schema A, and hence it was unnecessary to plot the smooth lines for the test accuracy and loss. The train accuracy for the simple model starts from 0.8 and stays near 0.92, while the test loss also stays near 0.81 without extreme

volatility. Meanwhile, test loss starts from the fairly low number as 0.75 and fluctuates a lot at the early stages of epochs. It stabilized around 15th to 20th epoch at the value around 0.53. The train loss stayed at its value around 0.21 at the later epochs.

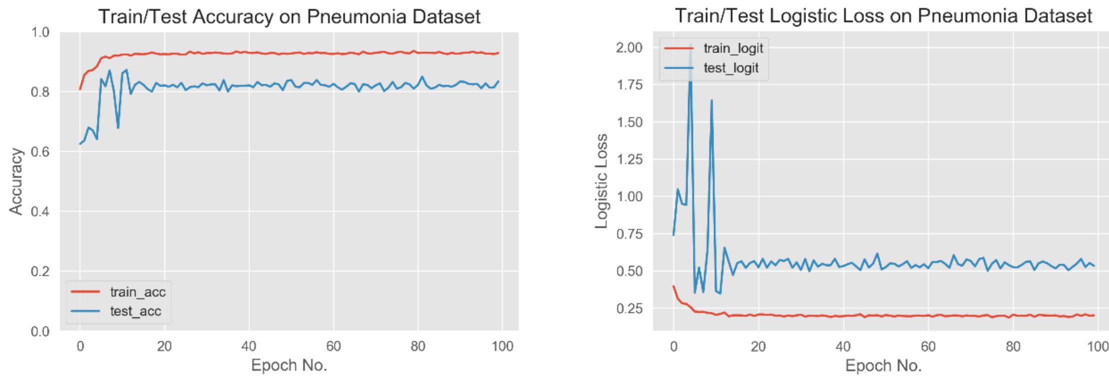

Figure 13: Train and test accuracy graph (left), and train and test loss graph (right)

The confusion matrix of our simple model illustrates that the results are fair enough for the classification, compared to the ones in schema A. The true positive, false negative, false positive, and true negative values are 321, 69, 23, and 211. The accuracy of the model is 0.85, while the sensitivity and specificity of the model are 0.82 and 0.90. A little higher sensitivity than specificity implies that this model is less powerful in identifying pneumonia than normal. The precision of the model is 0.93, while the F-1 score is 0.87.

Actual \Predicted	Pneumonia	Normal
Pneumonia	321	69
Normal	23	211

6.2.2 Medium model B analysis

According to the plots for the train and test accuracy of the medium model B, the test accuracy stabilizes even better than the previous simple model B with the value around 0.82, while the

train accuracy also stops at the value at around 0.93.

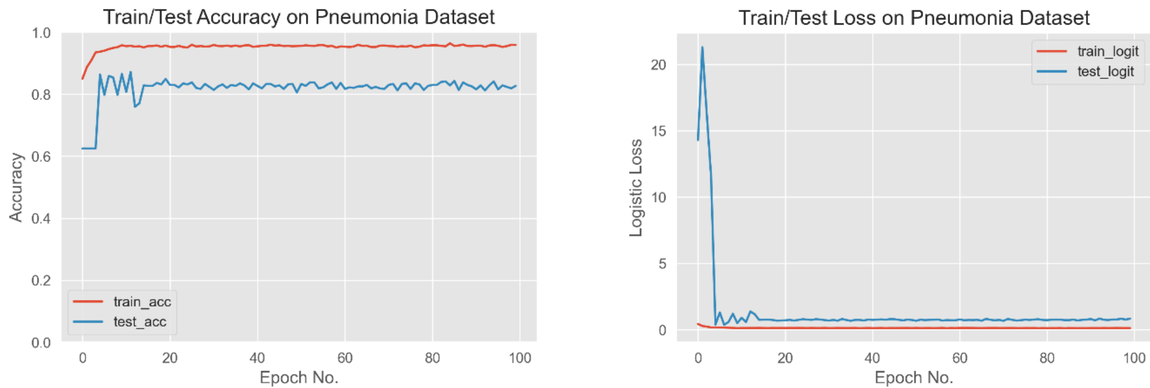

Figure 14: Train and test accuracy graph (left), and train and test loss graph (right)

The test loss surged up to 23 in the early stages of epoch, but soon it drops drastically with some fluctuating period afterwards and stabilizes its value at around 0.75. In the meantime, the train loss nearly stays at 0.14.

The result values in the confusion matrix of the medium model is a little different from the previous one for the simple model, although it is not a significant change. The true positive, false negative, false positive, and true negative values are 374, 16, 40, and 194. The accuracy of the model is 0.91, which is higher than that of the previous model, while the sensitivity and specificity of the model are 0.96 and 0.83. This time, the specificity is higher than the sensitivity meaning that the model is more powerful in predicting pneumonia than normal. Meanwhile, the precision of the model is 0.90, while the F-1 score is 0.93.

Actual \Predicted	Pneumonia	Normal
Pneumonia	374	16
Normal	40	194

6.2.3 Medium-complex model B analysis

Compared to the medium model B, our medium-complex model results in a more stabilized graph for train and test accuracy. The train accuracy quickly stops at the value approximately at 0.97, and the test accuracy shows its values around 0.82. The test loss along the epochs also demonstrates its improved values which fluctuates less than it did for the previous medium model at the lower value around 0.64. The train loss almost sticks to its value around 0.08 throughout the whole epochs. As reported by the confusion matrix, our medium-complex model shows the result values for the true positive, false negative, false positive, and true negative values as 347, 43, 21, and 213. The accuracy of the model is 0.90, which is approximately the same as the previous model, while the sensitivity and specificity of the model are 0.89 and 0.91.

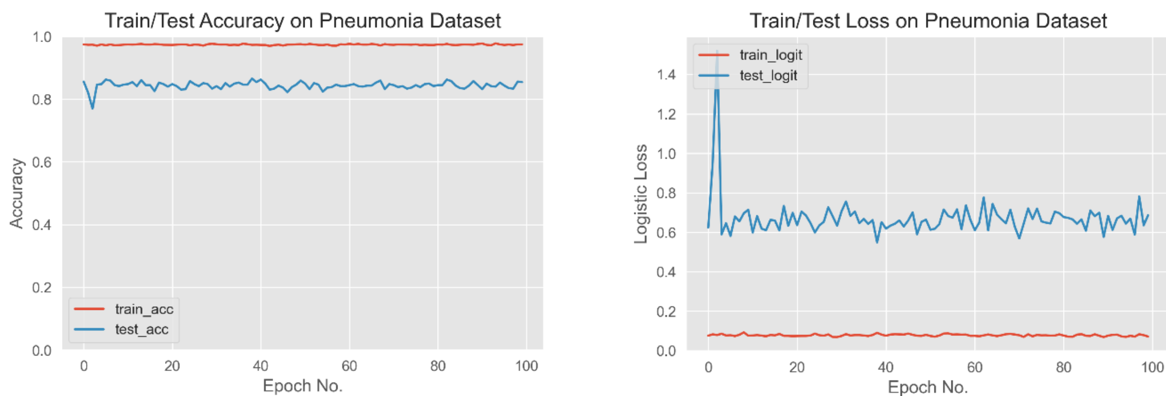

Figure 15: Train and test accuracy graph (left), and train and test loss graph (right)

The sensitivity and specificity of the model are about the same and that suggests that the model correctly classifies the pneumonia and normal images approximately at the same extent. In the meantime, the precision of the model is 0.94, which is higher than the previous results, while the F-1 score is 0.92.

Actual \Predicted	Pneumonia	Normal
Pneumonia	347	43
Normal	21	213

6.2.4 Complex model B analysis

As shown in the figure, the test accuracy shows a little more volatility than it does for the medium-complex model at the early stages of epoch, however it lingers around 0.83 later on. The train accuracy stays at 0.98, in the meantime. The figure on the right shows that the test logistic loss fluctuates extremely in a wider range than that of the medium-complex model at the early epochs and then stabilizes after 15th to 20th epoch around 0.7. As can be observed from the confusion matrix, corresponding values for the true positive, false negative, false positive, and true negative are 354, 36, 26, and 208.

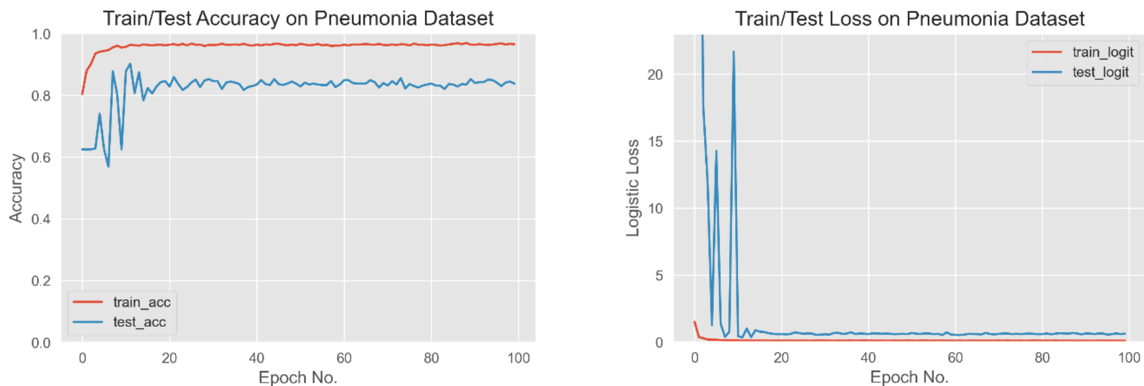

Figure 16: Train and test accuracy graph (left), and train and test loss graph (right)

The accuracy of the model is 0.91, which is slightly higher than the previous model, while the sensitivity and specificity of the model are 0.91 and 0.89 which are about the same. The precision of the model is 0.93, which is slightly lower than the previous results, while the F-1 score is 0.92.

Actual \Predicted	Pneumonia	Normal
Pneumonia	354	36
Normal	26	208

6.2.5 Further explanation for schema B analysis

As can be shown in the figure, our schema B also resulted in a graph where it showed an optimal capacity on the x-axis but it appeared at the second model. The loss value for the first model in this schema was far smaller than that of the previous one because the number of parameters for this schema is already big while the simple model A contains comparably small number of parameters compared to the other complex models.

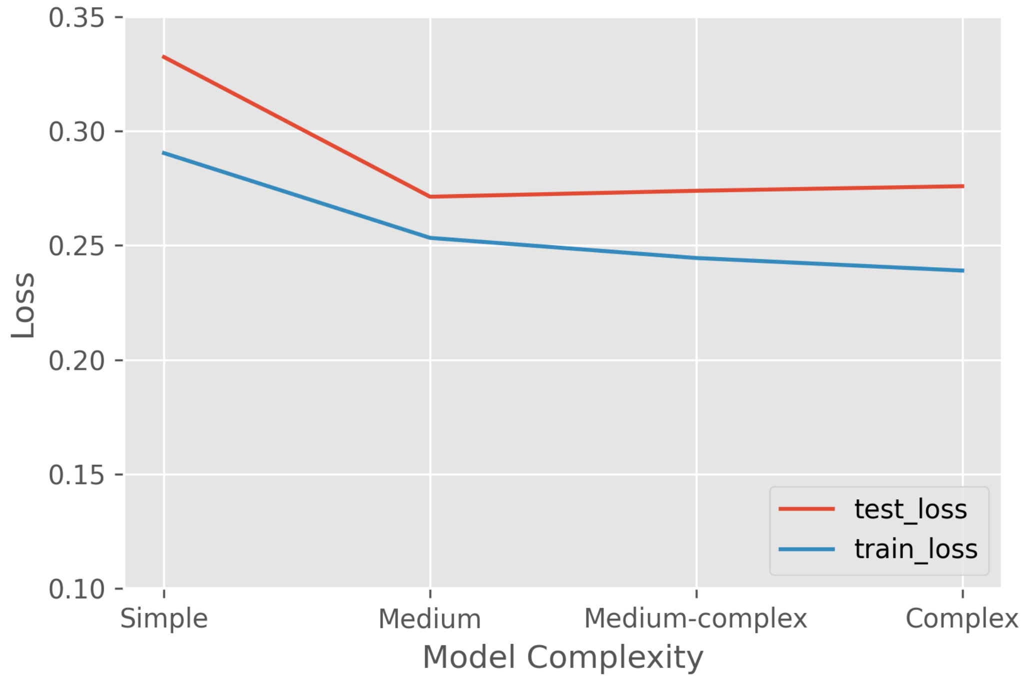

Figure 17: Train and test loss for simple to complex model B

The test loss of the medium model presented the lowest value among the four models and it slightly

increases in medium-complex model. The bias-variance tradeoff occurred at some point of model complexity after medium model. If we do not disregard the slight increase in test loss from medium to medium-complex model, the optimal model complexity can be somewhere between medium to medium-complex model.

7 Conclusion

The two experiments provide a practical implementation of deep learning image processing in the real world. In the bigger scope, those two different schemas are showing the outputs from variation in hyperparameters, while four models in each schema with the fixed hyper parameters are designed by structuring the neural network for different numbers of parameters. The graphs and confusion matrices for each experiment suggest that the medium-complex model in schema A and the medium model in schema B are the most desirable models among the four candidates in each experiment. The best model among the whole experiment might be the medium model in schema B as it produces preferable results in the confusion matrix as well as its optimal complexity, explained by the bias-variance tradeoff theory.

For schema A, some unappealing results in confusion matrices can be implied by the volatility in the test loss values which can be explained by the essence of binary cross entropy. For example, even though the smoothed line suggests that the mean values of test loss for the medium-complex model in schema A is lower than that of the medium model in schema B, the classification power of medium model in schema B is better than that of medium-complex model in schema A. This volatility might be caused by a high learning rate fixed at 0.01.

In terms of the computational cost, it turned out that the early stopping could be employed for the better experiment as the monitored results for the later epochs were not showing significant changes. Moreover, the improvement of the results might be possible with a better GPU or machine.

To sum up, the solid approach to achieve the moderate Convolutional Neural Network model with high performance is to start testing out different experiments with different hyperparameters. Thorough investigation for the choice of hyperparameter in accordance with the nature of data and the given environment must be preceded by the implementations of the experiments. Afterall, determination of the best fit model is possible with careful and holistic analysis of optimal model complexity.

Deep learning algorithm is extremely complicated and its application to the real world problem appears different in most of the new cases. Hence, our best bet is to conduct experiments in different settings for the best approach to the solution in a flexible way.

8 Python code

The whole codebases were created and tested after the careful investigation of a published book and learned from the content in the book [14].

It was rearranged to a modular code for the applications in the data science fields of industry. The following link is the Github repository which connects to a collection of the source code to build this application:

https://github.com/jpark143-jp/jpark143-jp.github.io/tree/master/COVID-19_CNN

9 References

[1] Goodfellow, Ian, et al. *Deep Learning*. The MIT Press, 2016, pp. 330–372.

[2] Zhang, Aston, et al. “The Cross-Correlation Operation.” *Dive into Deep Learning*, pp. 231–233.

[3] Géron, Aurélien. “CNN Architectures.” *Hands-on Machine Learning with Scikit-Learn & Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, Inc., Sebastopol, CA, 2019, pp. 365–375.

[4] Zheng, Alice. “Evaluation Metrics/Log-Loss.” *Evaluating Machine Learning Models*, O'Reilly Media, Inc., 2015, pp. 7–10.

[5] Hastie, Trevor, and Robert Tibshirani. “Bias-Variance Tradeoff.” *An Introduction to Statistical Learning with Applications in R*, Springer, New York, 2021, pp. 33–36.

[6] Moolayil, Jojo. “Hyperparameter Tuning.” *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python*, Apress, Berkeley, CA, 2019, pp. 142–151.

[7] Kermany, Daniel; Zhang, Kang; Goldbaum, Michael (2018), “Large Dataset of Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images”, Mendeley Data, V3, doi: 10.17632/rscbjbr9sj.3

[8] Reed, James C. *Chest Radiology: Patterns and Differential Diagnoses*. Elsevier Masson, 2019.

[9] Shorten, Connor, and Taghi M. Khoshgoftaar. "A Survey on Image Data Augmentation for Deep Learning." *Journal of Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0197-0.

[10] Venkatesan, Ragav, and Baoxin Li. "Image Representation Basics." *Convolutional Neural Networks in Visual Computing: A Concise Guide*, CRC Press, Taylor & Francis Group, CRC Press Is an Imprint of the Taylor & Francis Group, an Informa Business, Boca Raton, 2018, pp. 3–10.

[11] Pattanayak, Santanu. "VGG16." *Pro Deep Learning with Tensorflow: A Mathematical Approach to Advanced Artificial Intelligence in Python*, Apress, Berkeley, CA, 2017, pp. 209–210.

[12] Aghdam, Hamed Habibi, and Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks a Practical Application to Traffic-Sign Detection and Classification*, Springer International Publishing, Cham, 2018, pp. 79–128.

[13] Michelucci, Umberto. "Training Neural Networks." *Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks*, Apress, Berkeley, CA, 2018, pp. 137–178.

[14] Manaswi, Navin Kumar. "CNN in Tensorflow." *Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition with TensorFlow and Keras*, Apress, 2018, pp. 97–114.