Classification of COVID-19 with CT Images by CNNs

1 Introduction

As we know, the transmission rate of COVID-19 depends on the capacity to reliably identify infected patients with a low rate of false negatives. The COVID-19 presents several unique features on chest X-ray and computed tomography (CT) images, however, these characters are only moderately distinguishable for the human eye. In addition, a low rate of false positives is required to avoid further increasing the burden on the healthcare system by unnecessarily exposing patients to treatment if that is not required. Therefore, the CT image data can be used to train a classification model to help the healthcare system and auto-diagnose the pneumonia disease caused by the novel coronavirus.

The presented work mainly focuses on how to establish an efficient and accurate model to distinguish between COVID-19 and healthy patients. The image data for pneumonia and healthy patients are obtained from Kaggle and they are used to train the classification model. Since Convolutional Neural Networks (CNNs) have achieved expert-level performance in complex visual recognition tasks, the flexible model can extract intricate patterns directly from the CT images without the need to manually define the features by doctors.

Six different CNN architectures have been applied to capture the special characters on CT images and two different classification functions have been tried as the output layer. The results show that a detection accuracy of the disease about 92.27% is achieved by the final model.

2 Methodology

2.1 Data Preparation

The training dataset and testing dataset with 690 unspecified images were obtained from Kaggle. There are 5,266 X-Ray training images including 1,341 Normal X-Ray and 3,925 COVID-19 images. The original training data was randomly split into training set with 3949 images and validation set with 1317 images. (train:validation = 3:1).

2.2 Convolutional Neural Networks[1]

A Convolutional neural network (CNN) is a neural network with more convolutional layers and is mainly used for classification of image data. The idea for this method, is basically to extract the high level features of the input through applying filters in later layers. A convolutional neural network consists of one input layer and the hidden layers (including the output layer). In the hidden layers, the use of ReLU layer as the activation function is common, followed by additional convolutions such as pooling layers, and fully connected layers or normalization layers. These are referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. Explanation of the hidden layers used in this work can be found in Appendix A. Different combinations of these layers can be used as different models. The Figure 1 shows a classical example of the architecture of CNN, named ResNet50. On the left the ResNet50 architecture is shown[2]. The blocks with dotted line represent modules that are removed in some of the conducted experiments. On the middle, is the convolution block which changes the dimensions of the input, and on the right is the identity block which will not change the dimension of the input.

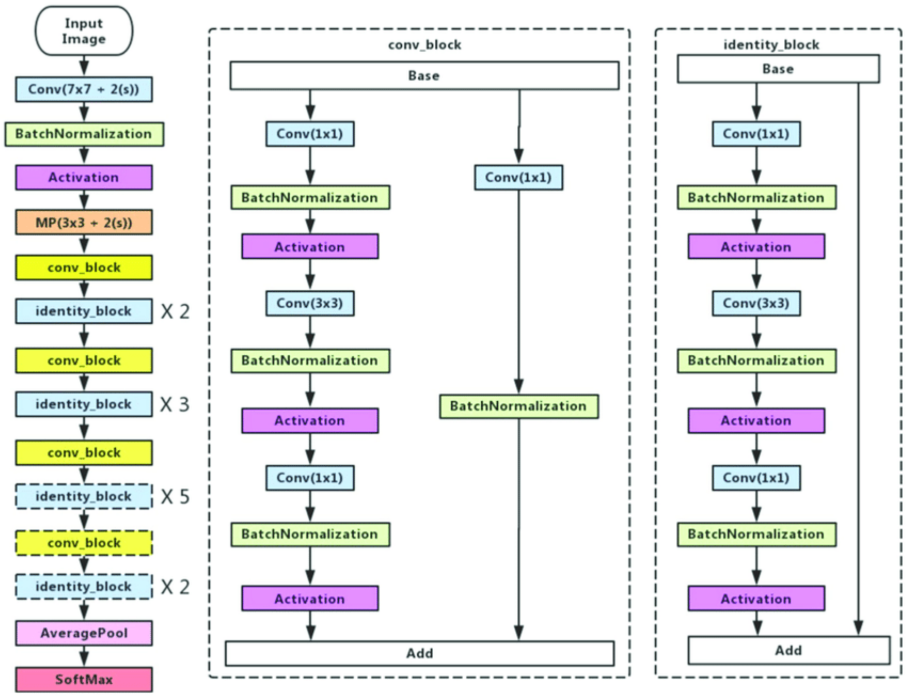

Figure 1: The Architecture of ResNet50[2]

2.3 Model Comparison

In the last layer of the architecture shown above, SoftMax function has been applied for classification. However, different architectures of CNN have different performances. Here, the comparison of six different CNN-based models is performed and the best result is reported with confusion matrix and relative statistics. Finally, the model is applied to the Kaggle test dataset to get the final testing score and verify the results. The models are ranked based on their final Kaggle score.

3 Implementation Details

3.1 Images Processing

Firstly, some data preparation methods were applied to set up. In order to avoid the over-fitting problem, data augmentation methods for training data were applied to increase the size of dataset by artificially generating data. The augmentation parameters which were used are as following, and were chosen based on references[3]:

- Normalization: setting the mean to zero and dividing by standard deviation for each sample and the whole dataset.
- Rotation: randomly rotating with an angle of 10°.
- Zooming: randomly zooming with a scale of 0.1.
- Shifting: randomly shifting images horizontally and vertically with a scale of 0.2
- Flipping: randomly flipping images horizontally.

Also, since the CT images have different sizes, resolutions and lighting conditions, in order to do visualization and train CNNs model more simply, all the images were resized to have the same scale. Different sizes were experimented to find the scale which performs better for each model, such as 150×150, 200×200, 250×250 and 350×350 pixels. The results show that when resizing the images to sizes smaller than 250×250, the accuracy for all models are relatively smaller and less stable.(See Appendix A) Also, by increasing the size bigger than 250×250, the performance of classification does not change a lot. In this case, all of the data was resized into 250×250 pixels for all the final models.

3.2 Model Architecture

As mentioned before, a total of six different architectures were used, three of which were chosen from known neural network models, ResNet50, AlexNet and VGG16, and three architectures were built layer by layer, CNN1, CNN2 and CNN3. The optimizer, the loss function and the metric were added to the end of the models. The loss function shows the difference between the predicted labels and the true labels after each optimization, and the way of calculation depends on the function. The metric to track the results for each step was chosen as accuracy which is the ratio of correct predictions to total predictions.
ResNet50 architecture, as the best model, was shown above. ADAM was chosen as the optimizer and categorical cross-entropy was set as loss to track the model fitting with accuracy output. The structures of other models, the loss functions and classifiers used are available in Appendix A.

3.3 Training and Testing

After stetting up the model, the optimizer and the loss function, the training and testing processes were carried out. For each model, learning rate and epoch number were manipulated to achieve better results. After completion of each training/testing process, the results of models were compared in three forms: a plot of "model accuracy" vs. "epoch number", a plot of "model loss" vs. "epoch number", and the confusion matrix of classification.
Another testing procedure was then conducted on the unknown-labels portion of the dataset, to obtain predicted labels for the 690 images. The predicted labels were used in Kaggle to assure the final results are reliable, and over-fitting is not a matter of concern.

4 Results

4.1 Model Comparison

Test scores acquired by Kaggle for the six models are shown in Table 1:

Model	CNN1	CNN2	CNN3	ResNet50	VGG16	AlexNet
Kaggle Score	83.82%	83.82%	86.47%	**92.27%**	83.82%	82.61%

Table 1: Kaggle scores for all models

It shows that the Kaggle score of ResNet50 is 92.27%, which achieves top 5 in the Kaggle Competition. It means that this model can help distinguish CT images between healthy people and COVID-19 patients with accuracy 92.27%.

4.2 Results of ResNet50

Since the performance of ResNet50 works best, more details of the fitting history of ResNet50 will be shown. Different epochs (= 10,15,20,25,30,40) have been tried to train the model. The result shows that if too many epochs are set up (more than 15) for ResNet50, it will become unstable and is easy to cause overfitting (see Appendix A). Also different learning rates were tried in ADAM optimizer, and the result shows that it can safely be set it as 0.0001 to avoid unstable fitting. In this case, epoch was chosen as 15 with the learning rate = 0.0001 to train the model. The Figure 2 shows that, as the epoch increases, the accuracy rates of training set and test(validation) set are close and stable above 90%. In addition, the model loss also decreases across epoch and is stable in a lower region, which indicates that it might not have overfitting issues.

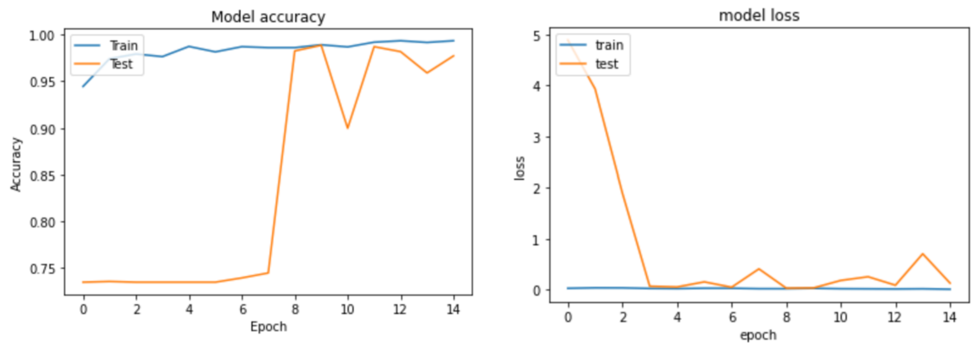

Figure 2: The Fitting History of ResNet50

Validation Set	Normal	Pneumonia
Normal	347	2
Pneumonia	19	949

Training Set	Normal	Pneumonia
Normal	1337	4
Pneumonia	96	3829

Table 2: The Confusion Matrices of ResNet50

Table 2 shows the confusion matrix in validation set (left) and the confusion matrix in training set (right). It can be used to calculate the accuracy, sensitivity and specificity (denote COVID-19 as positive) as done in Table 3:

-	Accuracy	Sensitivity: TP/(TP+FN)	Specificity: TN/(TN+FP)
Training Set	98.10%	97.55%	99.70%
Validation Set	98.41%	98.04%	99.43%

Table 3: Accuracy, sensitivity and specificity of ResNet50

From the table one can see the accuracy, sensitivity and specificity are very close and stable in both validation set and training set, which means that the ResNet50 model is not likely to be overfitted. Since the sensitivity is about 98%, it means one can precisely detect the COVID-19 patients using this model. It will avoid false negative issue which can lower the possibility of infection in latent period of COVID-19. In addition, the specificity is about 99%, which means that one can distinguish the normal patients with high accuracy. In this case, it will avoid false positive issue that deteriorate the shortage of medical resources.
In conclusion, the classification model trained by ResNet50 can be used to capture the features of X-ray images between normal patients and COVID-19 patients and then predict precisely in practice.

Reference

[1] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2] Ji, Qingge & Huang, Jie & He, Wenjie & Sun, Yankui. (2019). Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images. Algorithms. 12. 51. 10.3390/a12030051.

[3] https://www.kaggle.com/rajmehra03/a-comprehensive-guide-to-transfer-learning

[4] https://www.kaggle.com/aakashnain/beating-everything-with-depthwise-convolution#Model

A Additional Proof Details

A.1 Utilized Functions

This section is a brief explanation of how different utilized functions work.
Loss Functions: Two different loss functions were used in this work: Categorical CrossEntropy and Binary CrossEntropy. Both functions work based on calculating the difference between the probability distributions of the predicted model and the actual data. Loss functions were chosen based on references[3].
Classification Functions: The classification functions that were tested are SoftMax function and Sigmoid function. The former converts the outputs of the last layer into probabilities that add up to one, whereas the latter converts each of the outputs of the previous layer to a probability between zero and one, predicting what is the probability of the output belonging to a specific class. Classification functions were chosen based on the fact that two classes exist, and references[4].
Optimizer: The only optimizer used in this work is Adam. Adam scales weights and the learning rates of the parameters throughout the process by working similar to Stochastic Gradient Descent but using second moments of the gradients.
Additional layers: There are a number of additional layers, that were used in building the models. ReLU layer is the activation function that outputs the maximum of zero and the input. Maximum pooling returns the maximum value of its input region, and is an example of pooling layers. Normalization layers (e.g. Batch Normalization) normalize the inputs to a layer for each batch. The flatten layer converts everything into 1-D to create a single vector. The dense layer performs like a normal neural network layer with specified number of nodes. The dropout layer sets the output of specified edge neurons to 0, to prevent overfitting.

A.2 Architecture of Tested Models

The architecture of built models is presented in Figure A, and the architecture of two other models (AlexNet and VGG16) are shown in Figure B and C.

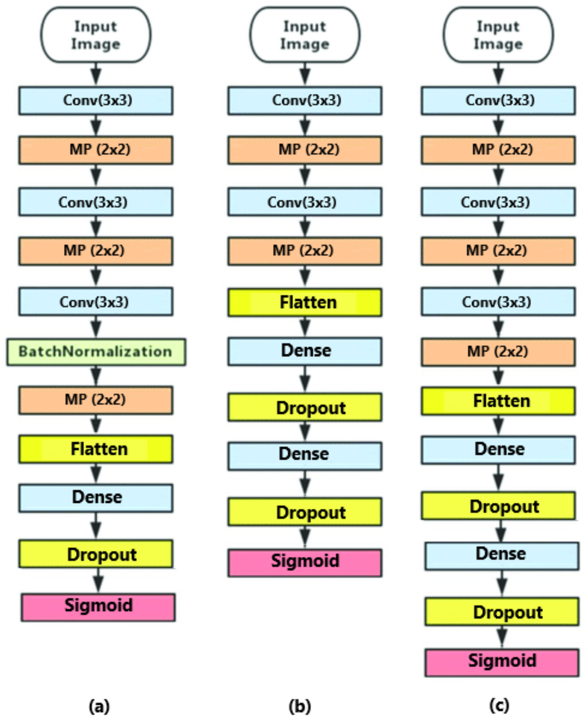

Figure A: Architectures of built models: (a) shows CNN1, (b) shows CNN2 and (c) shows CNN3

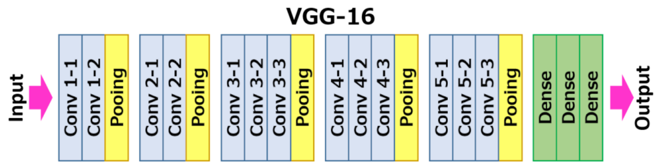

Figure B: Architectures of VGG16

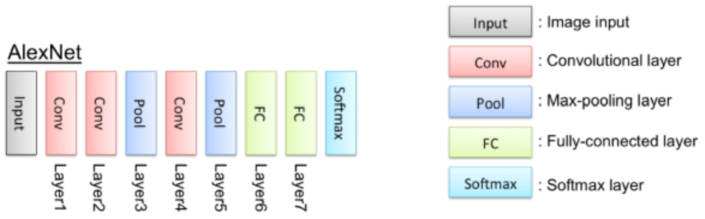

Figure C: Architectures of AlexNet

A.3 Case of Overfitting

ResNet50 was taken as example to track if there is any overfitting issue by comparing the Kaggle scores with different epoch numbers. The Table A shows that when epoch size increases more than 15, the Kaggle scores start to fluctuate and decrease a little. In this case, the epoch size 15 is appropriate.

Epoch	10	15	20	25	30	40
Kaggle Score	76.32%	92.27%	91.06%	90.09%	91.06%	84.78%

Table A: Kaggle scores for testing with different epoch

A.4 Case of Smaller Resizings

In another test, all the data was resized to 150×150 pixels, and they were used to train CNN2 and ResNet50. The Kaggle scores obtained for those cases are shown in Table B. It can be seen the values are smaller than the case of resizing to higher resolutions. In the Figure C, it shows that when resizing the images to sizes smaller than 250×250, the accuracy for all models are relatively smaller and less stable.

Model	CNN2	ResNet50
Kaggle Score	78.26%	76.57%

Table B: Kaggle scores for testing with smaller size

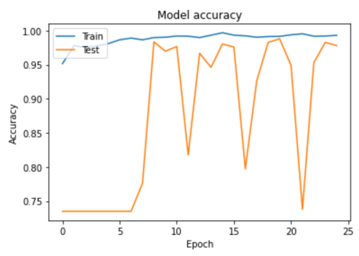

Figure D: Example of Sizes Smaller than 250×250