3D Lung Nodule Image Classification

Haoyin Ni hn21@rice.edu
Chengming Dong cd57@rice.edu
Code Github Link: https://github.com/dongccmg/COMP-576-Final-Project

Abstract

With the continuous development and progress of medical imaging technology and computer technology, medical image analysis has become an indispensable tool in medical research, clinical disease diagnosis, and treatment. In recent years, deep learning (DL), especially deep convolutional neural networks (CNNs), has rapidly developed into a research hotspot for medical image analysis, which can automatically detect the implied diagnostic features of diseases from medical image big data.

In our project, we use different deep-learning models to predict patients' sickness according to their 3D lung CT scans. We utilize several innovative ways such as data augmentation, base weight initialization, and prediction results averaging to improve accuracy and AUC score. In this report, we will introduce those methods as well as the architecture, advantages, and features of the different neural network models. A comprehensive comparison and analysis of them and the impact of their hyperparameters is also conducted. The main purpose is to find out which model and its hyperparameter combination could reach the AUC score, in other words, closest to the correct results for each patient.

Novelty & Highlights

- Applying 3 different ways of 3D image data augmentation to increase the data volume.
- Implementing all the deep learning models in 3D structure.
- Pre-training and loading the best base weight to the model before final training to improve running time efficiency and prediction accuracy.
- Hyperparameter tuning on DenseNet and ResNet.
- Implementing the pre-activation architecture and identity mapping on ResNet.
- Averaging multiple prediction results to minimize the error for unpredictable and vague samples.

1. Background & Motivation

Medical image analysis has been widely used in supporting the clinical auxiliary screening, diagnosis, treatment decision, and guidance for major diseases, such as malignant tumors, brain disorders, mental disorders, and cardiovascular diseases. The main application areas of current research on medical image analysis include image classification and recognition, localization and detection, and lesion segmentation. So far, scholars all over the world have carried out a series of deep learning research work mainly for medical image analysis tasks of

different imaging principles such as MRI, CT, X-ray, ultrasound, PET, and pathological optical microscopy [1].

Lung cancer is the most common form of cancer that leads to death in advanced health care [1]. It is widely acknowledged that lung cancer is one of the most threatening malignant tumors to human health and life. The key to reducing lung cancer mortality and improving survival quality is early diagnosis and treatment. Lung nodules are an early manifestation of lung cancer, and their detection rate has increased with the development of computed tomography (CT) technology. However, the significantly increased CT data has also increased the burden on physicians to read the images. Physicians will likely experience visual fatigue and an inability to concentrate when reviewing films for long periods, which may lead to the underdiagnosis of lung nodules [2]. The application of artificial intelligence (AI) technology for the initial screening of large volumes of CT images and the flagging of suspicious lesions can help physicians reduce workload and improve diagnostic accuracy [3,4].

Deep learning has proven to be a popular and powerful approach in many areas of medical imaging diagnosis. In recent years, neural networks, rebranded as “deep learning,” began beating traditional AI in all kinds of critical tasks: speech recognition, image characterizing, and natural language processing. Deep learning also improves the precision of computer vision, which is conducive to the performance of CT image detection and classification.

Challenges and Limitations:

In the feature extraction stage of traditional lung nodule detection methods, manually designing the image features of lung nodules in terms of morphology and texture is often necessary. The designed feature sometimes could not include all nodule features.

Artificial regulation of characteristics in traditional methods has great limitations which cannot completely cover all nodules with characteristics, and the detection effect of nodules is not ideal[9].

Traditional machine learning methods are sometimes ineffective in uncovering the rich information contained in CT images because of the tedious steps involved in selecting features manually[10]. The deep learning algorithm is a new field of machine learning methods. By simulating the human brain to build hierarchical models, it has powerful automatic feature extraction capability and efficient feature representation, which provides lung nodule detection problems with new ideas.

2. Related Work

Various initiatives are frequently developed aiming at increasing the accuracy of lung cancer diagnosis using deep learning models.

Hongyang Jiang et al. [5] propose an effective lung nodule detection scheme based on multigroup patches cut out from the lung images, which are enhanced by the Frangi filter. The scheme reached a sensitivity of 80.06% with 4.7 false positives per scan and a sensitivity of 94% with 15.1 false positives per scan.

Devinder Kumar et al. [6] present a methodology using the stacked autoencoder which achieves an accuracy rate of 75.01% on the National Cancer Institute (NCI) Lung Image Database Consortium (LIDC) dataset.

Johnsirani Venkatesan [7] proposed a non-Gaussian Convolutional Neural Networks [NG-CNN] architecture containing a feature extractor and classifier, which has been applied to training, validation, and test dataset. Their methodology outperforms the other competing methods and gives a test accuracy of 94.97% and an AUC of 0.896.

GS Tran et al. [8] evaluated their CNN model with a focal loss on the LIDC/IDRI dataset extracted by the LUNA16 challenge. Their architecture is proven to have an accuracy of 97.2%, a sensitivity of 96.0%, and a specificity of 97.3%.

3. Dataset

The project is using a private dataset from real-world hospitals. The dataset includes 465 labeled samples. For every sample, it includes

- A 3D image of the patient's lung. The image size is 100 * 100 * 100.
- Outline of the lung marked by doctors.
- The binary label. 1 stands for sickness, while 0 stands for health.

The task is to design and train our model using the labeled samples. Then we run the model with the input of 100 unlabeled samples. The output of every sample should be a number between 0 and 1. For example, 0.83 means the patient's probability of sickness is 83% and the patient's probability of health is 17%. The dataset has true labels of the 100 samples and will rate our answer.

To make full use of the data while reducing the computational burden, we pre-process both the train and test dataset by doing dot product on the 'voxel' and 'seg' array and then take its central part in the size of 32x32x32

Visualization

Each patient corresponds to a datapoint in the dataset, which is saved in an npz file. Each npz file is a lesion containing two 3D NumPy arrays, 'voxel' and 'seg', respectively.

'voxel' saves the data from the CT scan.

```
tmp['voxel']
```

```
array([[[187, 192, 190, ...,  12,  34,  15],
        [192, 191, 194, ...,   1,  19,  32],
        [189, 193, 191, ...,   3,   0,  26],
        ...,
        [140, 189, 198, ..., 167, 163, 165],
        [173, 206, 207, ..., 165, 168, 177],
        [150, 190, 186, ..., 167, 167, 186]],

       [[194, 194, 193, ...,   9,  34,  17],
        [198, 190, 192, ...,   1,  16,  31],
        [195, 190, 190, ...,   3,   0,  18],
        ...,
        [123, 173, 184, ..., 166, 162, 169],
        [163, 194, 202, ..., 166, 167, 175],
        [159, 196, 198, ..., 170, 169, 178]],

       [[194, 190, 191, ...,   7,  33,  19],
        [195, 188, 193, ...,   0,  10,  30],
        [193, 193, 190, ...,   4,   0,  12],
        ...,
        [ 98, 151, 162, ..., 164, 165, 169],
        [146, 177, 196, ..., 165, 165, 170],
        [150, 201, 204, ..., 167, 168, 172]],
```

Figure 1. Voxel array

'seg' is a boolean array, where the 'False' spots are not a nodule and the 'True' spots are a nodule. The role of the seg is to key out the nodules in the CT.

```
tmp['seg']
```

```
array([[[False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        ...,
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False]],

       [[False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        ...,
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False]],

       [[False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        ...,
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False]],

       ...,
```

Figure 2. Seg array

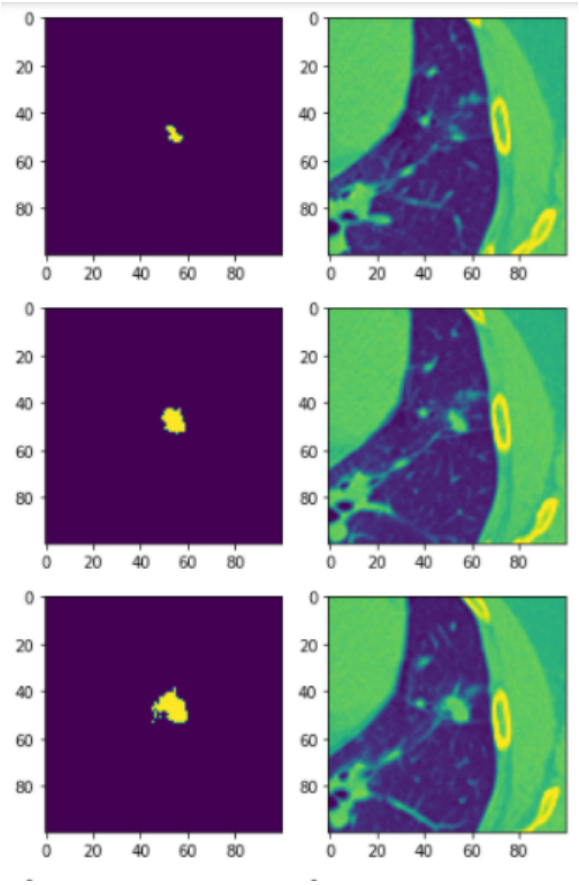

Figure 3. Nodule Locations and CT Scan Images

4. Data Augmentation

We are provided with only 465 samples. The dataset is too small so we need more data to train our model. We implemented three data augmentation methods for 3D image processing.

Rotation

We can rotate the original image by 90, 180, and 270 degrees to get three new images. Labels of the new images remain the same as the original ones.

Flip

We can flip the original image along the x-axis, y-axis, and z-axis to get three new images. Flipping along the diagnoses will make the size of the new image different from others. Labels of the new images remain the same as the original ones.

Mix up

We pick two random samples and mix them up. Considering the following definition:

X: the image of the first sample
x: the label of the first sample
Y: the image of the second sample
y: the label of the second sample
Z: the image of the new sample
z: the label of the new sample

Mix images: Z = 0.5 * X + 0.5 * Y
Mix labels: z = 0.5 * x + 0.5 * y

The samples created by rotation and flip are not necessarily new because they are just linear transformations of the original sample and features remain the same. But mixing up will create new samples because it introduces non-linearity. We are going to quadruple the dataset using these three methods.

5. Deep Learning Models

5.1 CNN

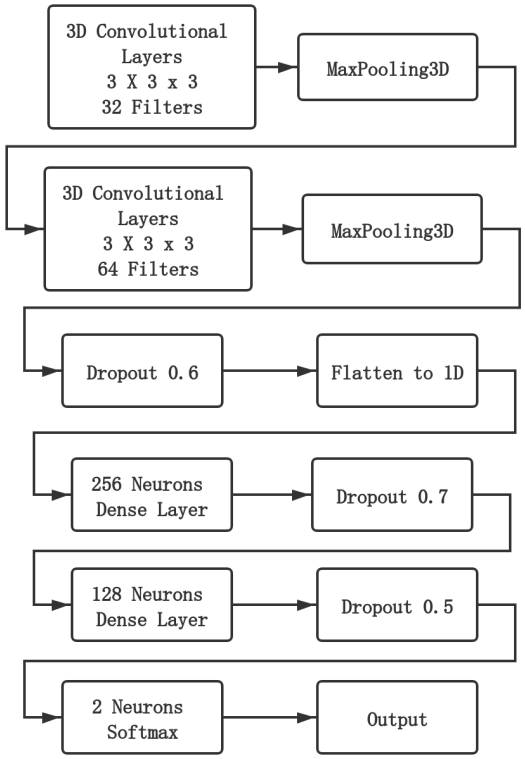

Figure 4. CNN Model Architecture

CNN is the first deep learning model we came up with since it is known for its accuracy and good performance in image recognition and classification. The CNN model we played with in assignment 2 shed light on the implementation of our own CNN. The main difference is that CIFAR is a 2D dataset while our dataset is 3D. Thus, we need to create a 3D-CNN, which is actually very similar to the 2D version. It also has the feature extractor and the ANN classifier that performs in the same manner as 2D-CNN.

The 3D-CNN performs 3D convolution instead of 2D convolution. In a 3d Maxpool (2x2x2 kernel), we look for the maximum element in a width 2 cube. This cube represents the space delimited by the 2x2x2 zone from the input. The number of operations is multiplied by the size of the filters used no matter how many layers of Maxpool or Convolution. The sequential API from Keras is used for building the model.

3D-CNN is our simplest and shallowest model. It runs relatively fast but also has the lowest AUC score during the experiment.

5.2 Inception Net

Inception network is an important milestone in convolutional neural networks[12]. Before Inception, the most popular convolutional neural networks simply stacked more and more convolutional layers, making the network deeper and deeper. This made the network more and more complex with more and more parameters, which led to a tendency of overfitting and increased computation. In contrast, the Inception network considers the parallel computation of multiple convolutional kernels, which greatly extends the width of the network.

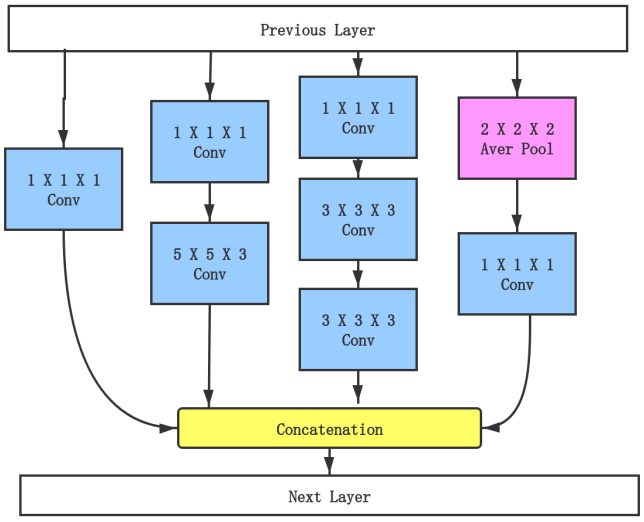

Figure 5. Inception Module Architecture

The core idea of Inception Net is sparse connectivity[12]. Because in biology neural connections are usually sparse. The most important feature of the Inception Net is the extensive use of Inception modules. Each Inception module has an architecture as shown in Figure 5.

The Inception module has different sizes of convolution, which represents the features of objects of different sizes captured by convolution kernels of different sizes. Specific sizes such as 1x1x1, 5x5x3, and 3x3x3 are used only for convenience.

In the module, the number of output channels of the module is the sum of the number of output channels of the four branches. This superposition inevitably increases the computational volume of each convolution in the Inception module. Therefore, after several modules, the computation volume may explode. The solution to this problem is to add an additional 1x1x1 convolutional layer before the 3x3x3 and 5x5x3 convolutional layers to limit the number of input channels fed to the convolutional layers. Another 1x1x1 convolution layer also comes after the average pooling layer not before. This is because the pooling layer is designed to extract the original features of the image, and once it is connected after the 1x1x1 convolution layer, it loses its original intent.

Compared with the naive CNN, Inception Net balances the width and depth of the network and improves the generalization ability of the network, so the computational resources also strike a good balance. CNN often requires a large number of parameters and can be computationally intensive, but the Inception network uses a modular design that allows it to learn more compact models with fewer parameters.

5.3 ResNet

As discussed in the two previous models, when the network depth increases, the accuracy of the network should increase simultaneously. Of course, we have to pay attention to the overfitting problem at the same time. But one critical problem with increasing network depth is that these increased layers are signaled by parameter updates. Since the gradient is propagated from backward to forward, and after increasing the network depth, the gradient of the previous layers will be small. This means that these layers basically stagnate in learning, which is also known as the gradient vanishing problem. Another problem is that when the network goes deeper, the parameters' space grows larger and makes the optimization more difficult. Thus, simply increasing the network depth could result in a higher probability of errors. While the residual module in ResNet is designed to allow us to train deeper networks.

ResNet challenges traditional neural network architectures in three ways:

- ResNet bypasses residual layers by introducing the shortcut, which allows data to flow directly to any subsequent layer. This is in contrast to traditional, pipeline architectures, where networks sequentially process low-level features to high-level features.
- ResNet has a very large number of layers, more than 1000. In architecture like AlexNet, the number of layers is two orders of magnitude smaller.
- Experimentally, we found that removing a single layer from a trained ResNet

does not affect its prediction performance. In contrast, the removal of layers in a trained network such as CNN results in a great loss of prediction performance.

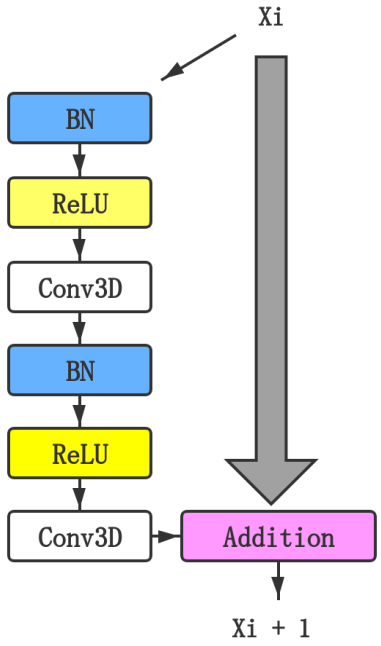

Figure 6. Residual Block Architecture

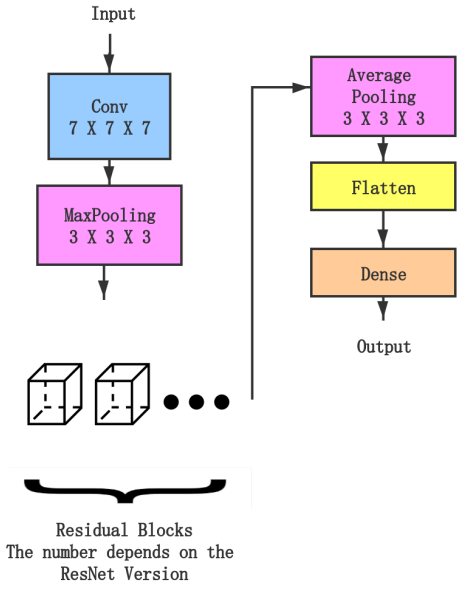

Figure 6. ResNet Architecture

ResNet combines various existing network structures and fully utilizes their advantages. The biggest feature of ResNet is its use of residual blocks, which are building blocks for the network that allow it to learn more complex and deep representations of the data[13]. This allows the network to bypass the convolutional layers and directly propagate the input through the shortcut connection, making it easier for the network to learn deep and complex representations. The graph here shows the structure of a residual block in ResNet. Each block consists of a series of consecutive layers and a shortcut, which directly flows from the input to the output. The shortcut creates a simple equivalent mapping without generating extra parameters or increasing computational complexity. Then, those two layers will be added together and passed to the next residual block.

Identity Mapping

Traditional neural networks suffer from gradient vanishing or explosion when they grow deeper. While by learning several later layers of the traditional deep network into a constant mapping $H(x) = x$, the model could be transformed into a shallow network. In ResNet, we realize this identity mapping by designing the network to be like

$$H(x) = F(x) + x \Rightarrow F(x) = H(x) - x$$

Where $H(x)$ is the output of the network, x is the input, and $F(x)$ is the residual function that represents the difference between the input and the output. Identity mapping $H(x) = x$ is satisfied as long as the residual $F(x) = 0$. At this point, the stacked layer only does a constant identity mapping, at least the network performance does not degrade. In

fact, the residuals will not be 0. This also allows the stacked layer to learn new features based on the input features, thus having better performance.

Pre-Activation

During the implementation, in order to retain the identity mapping, we use a pre-activation structure that puts the batch normalization and activation before the convolution layer, so as to make the shortcut connection path clean.
According to the paper[14], pre-activation residual blocks have better performance. One possible reason is the smooth shortcut path makes the optimization simpler. Plus, batch normalization is executed for both convolutional layers of inputs.

5.4 DenseNet

The DenseNet paper won the Best Paper Award with over 2000 citations in 2017 CVPR. It can achieve fewer parameters and higher accuracy compared with ResNet with dense connections among layers.

In ResNet, one layer takes its previous two layers as its input. In DenseNet, however, one layer takes additional outputs from all the preceding layers and passes on its own feature maps to all subsequent layers. It can be considered a more advanced version of ResNet and performs well in finding hidden and diversified features.

A DenseNet model comprises multiple dense blocks and transmit blocks (Figure 7).

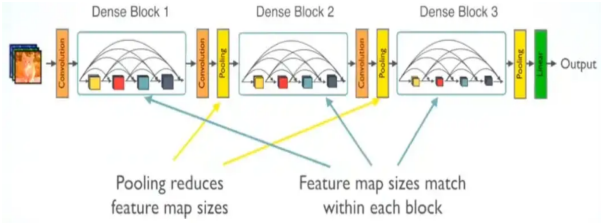

Figure 7. DenseNet Architecture

A dense block is to extract features from the input images. In a dense block, each layer

1) Obtains additional inputs from all preceding layers
2) Passes on its own feature maps to all subsequent layers.
3) Dimension remains the same to improve the computation efficiency.

A layer uses features from all preceding layers. Therefore, the output gives more smooth decision boundaries. That's saying, the result is not so absolute. For a binary classification task, the result tends to locate around 0.5 rather than 0 or 1. As a result, it performs well when training data is insufficient. As the name suggests, the layers are densely connected so the entire network can be thinner and more compact to reduce the number of parameters.

DenseNet connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - our network has L(L+1)/2 direct connections. For each layer, the feature maps of all preceding layers are used as inputs, and its own feature maps are used as inputs into all subsequent layers.

The error signal can be easily propagated to earlier layers more directly. This is a kind of implicit deep supervision as earlier layers

can get direct supervision from the final classification layer.

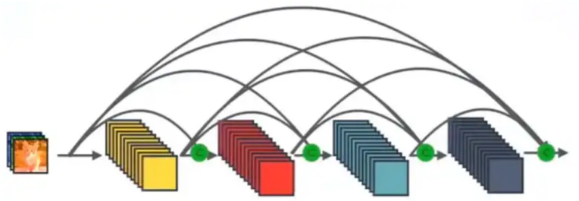

Figure 8. Dense Block Architecture

A transmit block serves as a connector between two adjacent dense blocks. It contains a convolution layer and a pooling layer, which can change the dimension of the input and reduce the feature map size. The last transmit block contains a global average pooling layer and a softmax classifier.

6. Experiment & Results

We conduct our experiment under the *Google Colab* environment. The advanced GPU and high RAM provided by *Google Colab Pro* are used to accelerate the training. The default version of Tensorflow 2.9.2 and Keras 2.9.0 serves as our main technique to build our own models.

We then use the *roc_auc_score* from *scikit-learn* to evaluate the performance of each model. as the testing measurement to evaluate the performance of each model. In binary classification, the goal is to predict one of two possible classes, in our case "0" or "1". The AUC score is calculated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold values. The resulting curve is known as the receiver operating characteristic (ROC) curve. The AUC score is the area under the ROC curve. A model with a perfect AUC score of 1.0 has perfectly predicted all positive and negative classes, while a model with an AUC score of 0.5 has performed no better than random guessing.

$$AUC = \frac{\sum_{ie\ positiveClass} rank_i - \frac{M(1+M)}{2}}{M \times N}$$

Expected Result

Since the dataset has binary labels, the accuracy is at least 50%. Even if your model is doing nothing but flipping a coin, the expected accuracy is 50%. Binary classification looks easy. But in this case, only experienced doctors using a sophisticated machine can give a correct answer so we don't think our model will reach a good accuracy such as 99.9%. Therefore, for this dataset, we believe an accuracy of 70% is good enough to win the competition.

6.1 Training Techniques

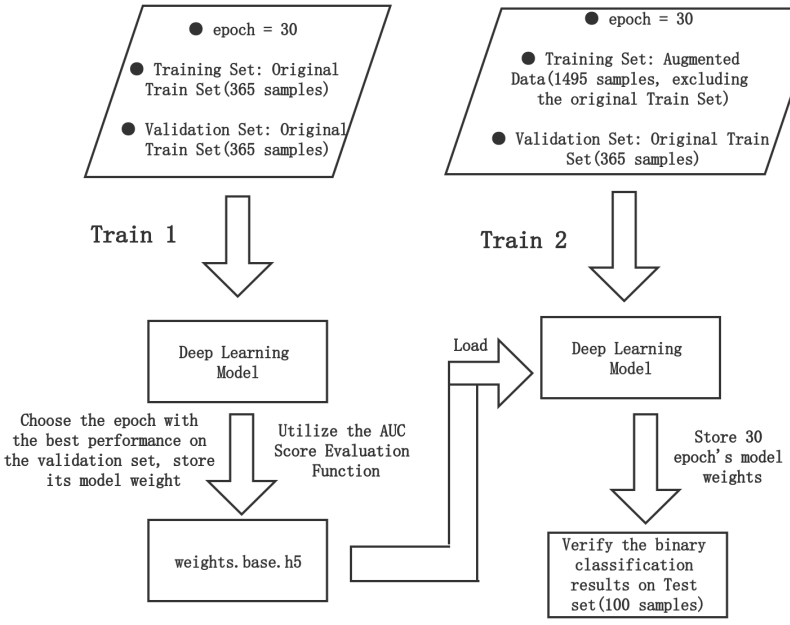

Figure 9. Training Steps

Our original dataset has 465 samples. Each of them has a label of 0 or 1 provided by real pulmonologists. We split it into a train set of 365 and a test set of 100. We also use the data augmentation technique in section 4 to triple the train set. We use batch size 16 for the entire experiment.

Pre-training

At first, we only train each model using the augmented data once just like most machine learning procedures. However, the result is not very good and the running time is pretty long. We infer that sometimes data augmentation could distort or degrade the original data, making it more difficult for the model to learn from. Additionally, applying too many transformations can make the data too different from the real-world data that the model will encounter at test time, leading to poorer performance on unseen data. Since our project is to classify medical images, geometric transformations could have the potential to change the nature of the medical picture, which leads to a great impact on the prediction results.

However, our train set is only 3 times the test set, which is definitely not enough. In order to obtain a compromise but a better result, we decide to pre-train the data. In the first train, we use the original train set and validate it using the same 365 samples based on the AUC score. We store the model's weight after each epoch in an h5 file. After the first training, we select the epoch with the highest AUC score as the base weight which initializes the model in our second training. Although this weight may exhibit certain overfitting on the train set, it won't be so severe that it performs extremely poorly on the test set due to slight overfitting. Then we replace the train set with our 1495 samples after data augmentation (without the original dataset) and train the weight-loaded model for 30 epochs again. The result will be verified on the 100 test set in the pre-training, and it can help the model to learn more quickly and accurately than if it were trained from scratch.

Pre-training allows the model to make use of the knowledge that has been learned by the previous training run and can help it to learn more accurately. By initializing the weights, our model could be better suited to the task at hand. Moreover, the training time on the large augmented dataset could also be greatly reduced.

6.2 Testing Techniques

Ensemble Learning

After training, we load each epoch's weight into the corresponding model again and make a prediction on the 100 test samples respectively. Then we output the results into a CSV file. In the beginning, we only use the last epoch's weight to generate the result. However, the score of the 30th epoch is not always satisfactory. Sometimes it's only between 0.5-0.6. We thought this might be caused by overfitting. However, after printing out the AUC score for all 30 epochs, we didn't find any trend of score changes after each epoch of training. The irregular fluctuations of the AUC score also deny the conclusion that the middle epochs around 15 perform the best. In order to improve the score, we determine to average the results of two models.

This technique is also known as *Ensemble Learning*. It is originally a traditional

machine learning method both applied for supervised and unsupervised learning. One advantage of Ensemble Learning is that it can improve the overall performance of the model by combining the strengths of multiple models. For example, if some models are better at capturing certain patterns in the data, while others are better at capturing different patterns, the final prediction will be more accurate than any of the individual models.

Another advantage of Ensemble Learning is that it can reduce overfitting. Overfitting occurs when a model is too complex and learns patterns in the training data that do not generalize to unseen data. By combining the predictions of multiple models, ensemble learning can help to reduce overfitting and improve the model's performance on new data.

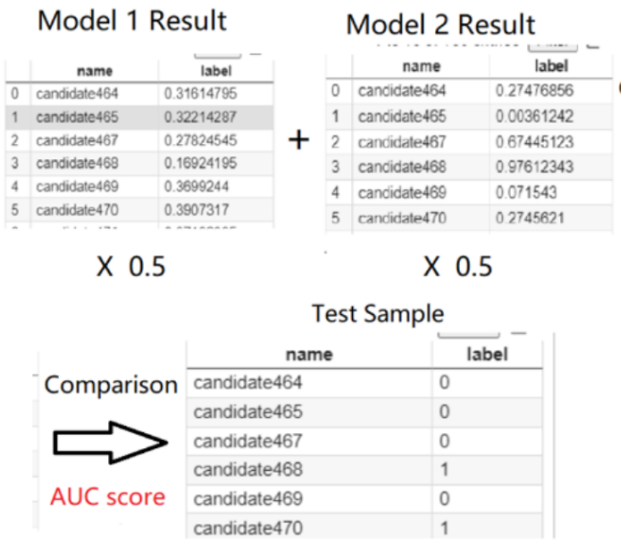

Figure 10. Averaging labels from 2 results

Take the above figure as an example. For candidate 464, the label value predicted by model 1 is around 0.316. The result from model 2 is around 0.275. We multiply each of these two labels by a coefficient of 0.5 and add them together to generate a new label(around 0.296) for candidate 464, and compare it with the true label 0 in the test set, which means candidate 464 should be healthy. We do such a mixup on all 100 samples and use the AUC score to evaluate the new result. We try all the combinations of the 30 results and recalculate their scores to find the best one. This score is potentially the final and the best score the current model could achieve.

6.3 Results

CNN

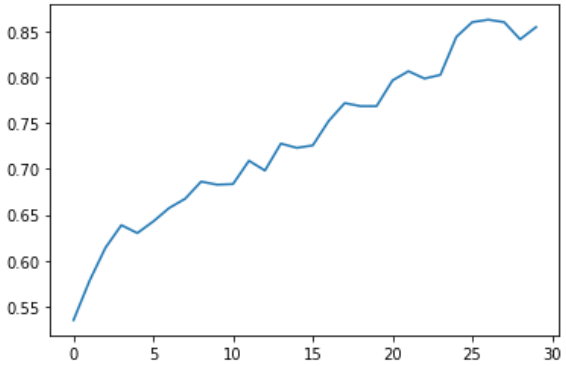

Figure 11. Categorical Accuracy Plot of CNN

Model	Training Accuracy	Final AUC_Score	Running Time
CNN	94.31%	0.5968	3min 06sec

Table 1 . CNN Performance Table

CNN is our most straightforward model. It is the only model which got an AUC_Score under 0.6, but CNN's running time is also the shortest.

The high training accuracy also indicates that our dataset is very tricky and our task is very difficult. There are many more examples of one class than the other, in

other words, the train set is heavily imbalanced. When the model is evaluated on a new dataset or in a real-world scenario, it may not perform as well because it is not able to accurately predict the minority class.

Inception Net

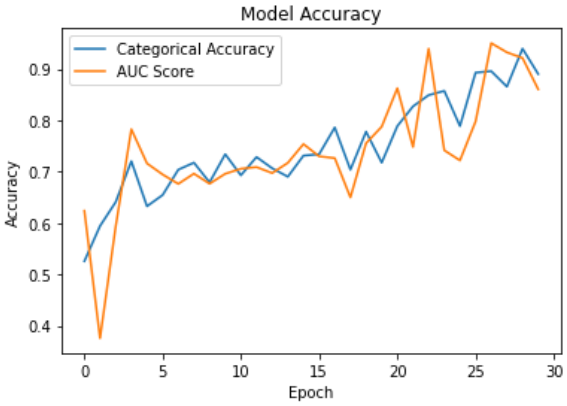

Figure 12. Model Accuracy Plot of Inception Net Train 1

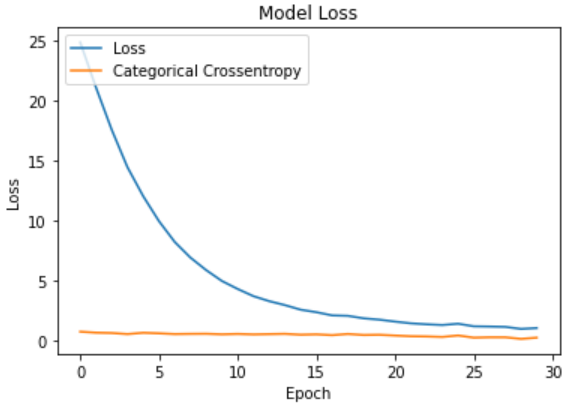

Figure 13. Model Loss Plot of Inception Net Train 1

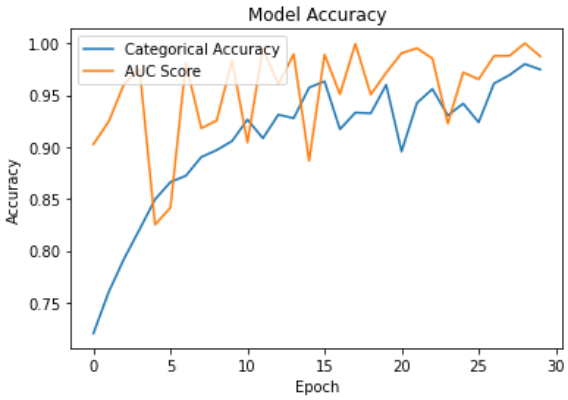

Figure 14. Model Accuracy Plot of Inception Net Train 2

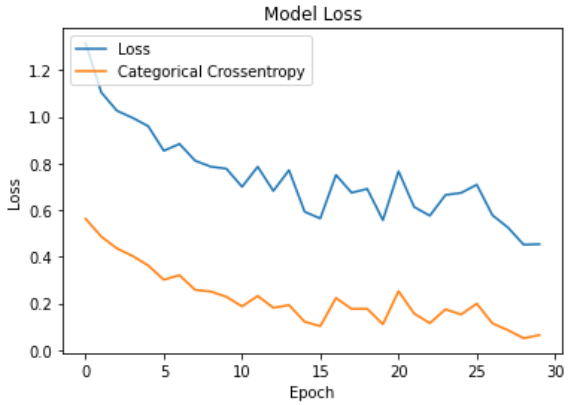

Figure 15. Model Loss of Inception Net Train 2

Inception	AUC_avg	AUC_max	Running Time
Train 1	0.7415	0.9505	4 min 16 sec
Train 2	0.9294	0.9995	6 min 17 sec

Table 2. Inception Net Training Table

Inception	AUC_avg	AUC_max	AUC_mix
Test 1	0.6206	0.658	0.6612
Test 2	0.5486	0.6515	0.6524

Table 3. Inception Net Testing Table

In the two tables, Train 1 stands for the pre-train using the original train set, and Train 2 stands for the second train using the augmented data. Test 1 tests the 30 models generated by Train 1. Test 2 tests the 30 models generated by Train 2.

From Table 2 we can tell that the second train has a significantly higher maximum and average AUC score maximum and AUC scores average than the first train. While training 3 times the data volume, the model with initialized weight only takes 1.5 times more time to complete training. All of those prove that the pre-training method is positively impacting training performance and increasing training efficiency.

From the 3rd column(AUC_max) and 4th column(AUC_mix) of Table 3, we conclude that for both tests, the mixed result has an even better AUC_score than the result of the best single model among 30 epochs. Both mixed scores are also way higher than the average. Thus, we can conclude that our Ensemble Learning method is working correctly to make more robust and accurate predictions.

Compared with CNN, Inception Net's running time is slightly longer. However, Inception Net has a better final AUC score of 0.6612, which clearly manifests the power of its novel inception module architecture.

ResNet

We tested 3 versions of ResNet, ResNet18, ResNet34, ResNet50, and ResNet101. The numbers in their names indicate the number of layers in the network. Another slight difference is that ResNet18 and ResNet34 use *basic block* while ResNet50 and ResNet101 use *bottleneck.*

The bottleneck block is more complex than the basic block and consists of three convolutional layers, with two shortcut connections. The first shortcut connection skips the first convolutional layer, while the second shortcut connection skips the second and third convolutional layers. This design allows the bottleneck block to learn more complex features because ResNet101 has more layers and is deeper than the other two.

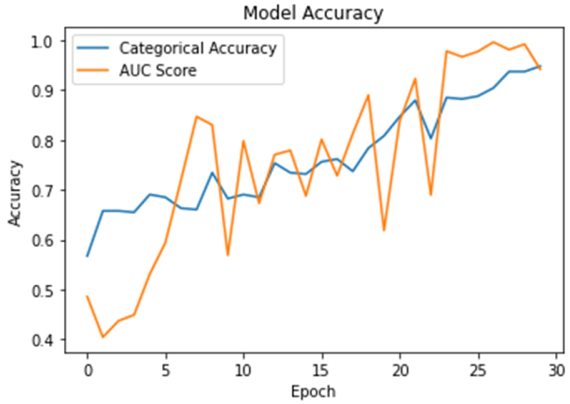

Figure 16. Model Accuracy Plot of ResNet18 Train 1

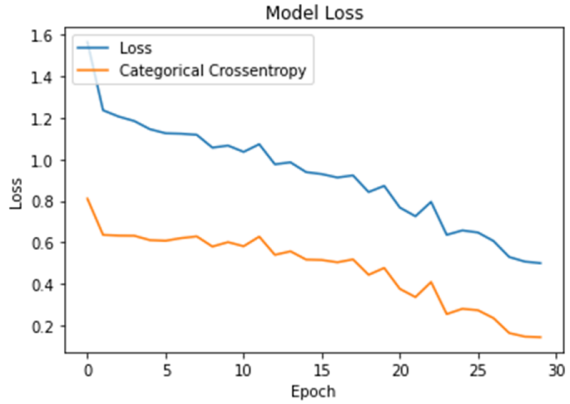

Figure 17. Model Loss Plot of ResNet18 Train 1

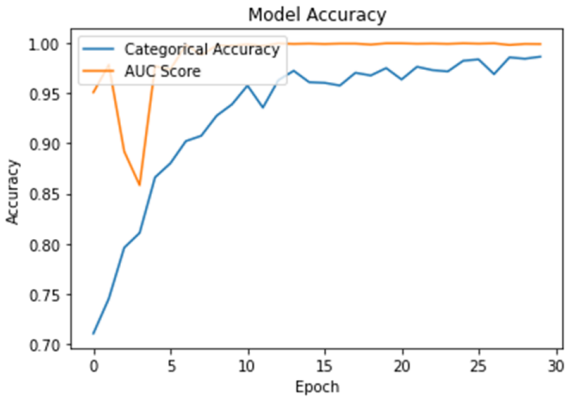

Figure 18. Model Accuracy Plot of ResNet18 Train 2

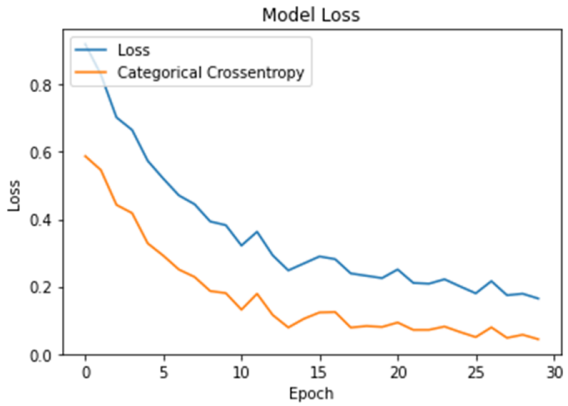

Figure 19. Model Loss Plot of ResNet18 Train 2

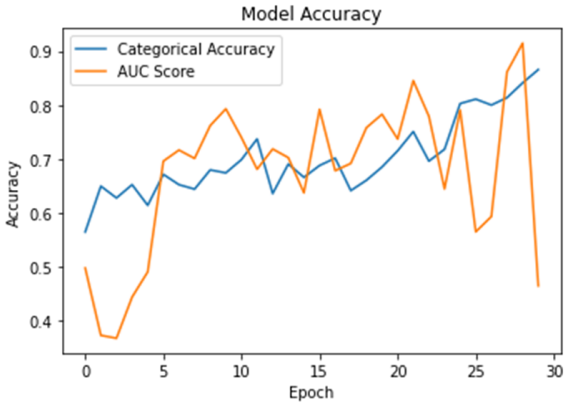

Figure 20. Model Accuracy Plot of ResNet34 Train 1

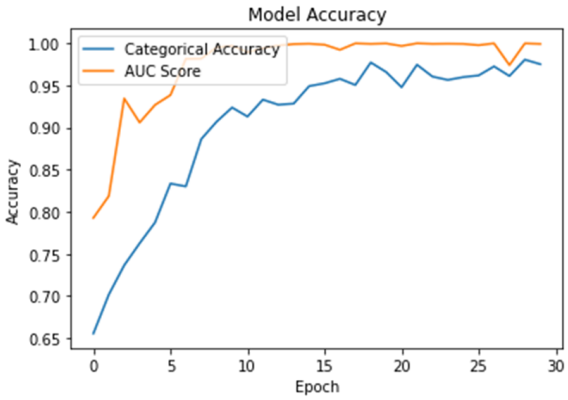

Figure 21. Model Accuracy Plot of ResNet34 Train 2

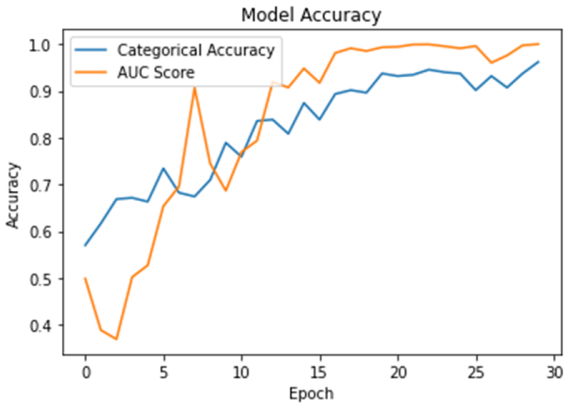

Figure 22. Model Accuracy Plot of ResNet50 Train 1

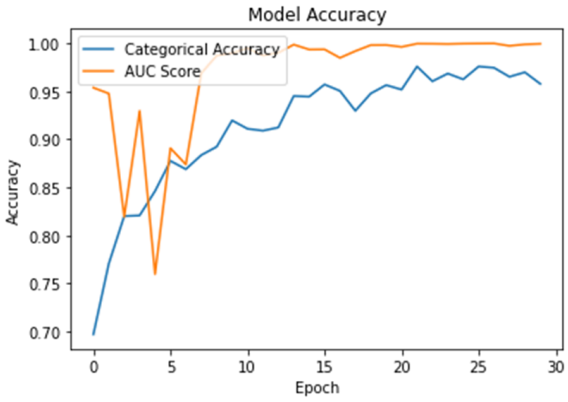

Figure 23. Model Accuracy Plot of ResNet50 Train 2

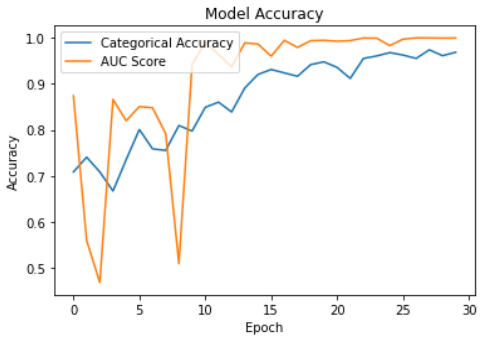

Figure 24. Model Accuracy Plot of ResNet101 Train 2

Model	AUC_ avg	AUC_ max	Running Time
Res18 Train 1	0.7568	0.9861	4 min 42 sec
Res18 Train 2	0.9867	1.0	7 min 38 sec
Res34 Train 1	0.6737	0.915	5 min 32 sec
Res34 Train2	0.9736	1.0	9 min 04 sec
Res50 Train 1	0.636	0.9872	5 min 30 sec
Res50 Train 2	0.9736	1.0	12 min 55 sec
Res101 Train1	0.8363	0.9985	7 min 23 sec
Res101 Train2	0.909	1.0	14 min 13 sec

Table 4. ResNet Training Table

Model	AUC_ avg	AUC_ max	AUC_mix
Res18 Test 1	0.5595	0.6796	0.6904
Res18 Test 2	0.5931	0.6450	0.6624
Res34 Test 1	0.5373	0.667	0.6776
Res34 Test 2	0.5818	0.6268	0.6578
Res50 Test 1	0.49	0.6244	0.6256
Res50 Test 2	0.5604	0.642	0.648
Res101 Test 1	0.5264	0.6266	0.6588
Res101 Test 2	0.5653	0.652	0.65

Table 5. ResNet Testing Table

With more models with different hyperparameters tested, we could be more confident in the effectiveness of our innovative training and testing techniques. as in Inception Net's experiment. According to Table 4, all the models' *AUC_avg* from *Train 2* are significantly higher than *Train 1*. All four models reach *AUC_avg* over 0.9 and *AUC_max* of 1.0 in *Train 2.* This means pre-training is effective. From Table 5, all the models' *AUC_avg* from *Test 2* are significantly higher than *Test 1*, which means the models with initiated weight also perform well on the test set compared to the empty models. At the same time, *AUC_mix* is always slightly higher than *AUC_max,* which indicates that trying the different combinations of two results and averaging their labels do improve the best possible prediction.

Comparing different versions of ResNet, we get the following results:

- *AUC_avg Train 2*

Res18 > Res34 = Res50 > Res101

- *Running Time Train 2*

Res18 < Res34 < Res50 < Res101

- *AUC_avg Test 1*

Res18 > Res34 > Res101 > Res50

- *AUC_avg Test 2*

Res18 > Res34 > Res101 ≈ Res50

- *AUC_mix Test 1*

Res18 > Res34 > Res101 > Res50

- *AUC_mix Test 2*

Res18 > Res34 > Res101 ≈ Res50

From the above analysis, the result has good uniformity and we could draw some conclusions with a certain trend. The average AUC score of the 30 epochs is the

most persuasive factor about the performance of a certain model. Res18 achieves the best *AUC_avg* on all experiments, including Train1, Train2, Test1, and Test2. Res101 gets a higher AUC during testing, but a lower AUC during training compared with Res50. In general, with the increase of layers, the ResNet model is definitely more computationally expensive based on running time, and maybe more susceptible to overfitting which causes a drop in performance. Res18 is considered to be the best ResNet model among the four due to its smaller number of parameters and faster inference times.

The best final AUC score in the ResNet experiment is **0.6904** from Res18's *Test 1,* which is higher than 0.6612 from Inception Net. Given that the running time of those two models is very close, we confer that ResNet is a better model than Inception Net on our task.

DenseNet

For DenseNet, we mainly tune the parameter "growth rate" of the network. It is represented as "k" in our code. The growth rate determines the number of feature map channels generated after each 1x1 and 3x3 convolution in the DenseBlock. These feature maps need to concatenate the previous feature map as input for the next layer. The feature maps output by each layer are added to this global state of the block, which can be understood as the "collective knowledge" of the block. The growth rate determines the proportion of those new features to the global state. This hyperparameter could affect the overall capacity of the network and its ability to learn complex features. Thus, the growth rate is the most important hyperparameter in DenseNet.

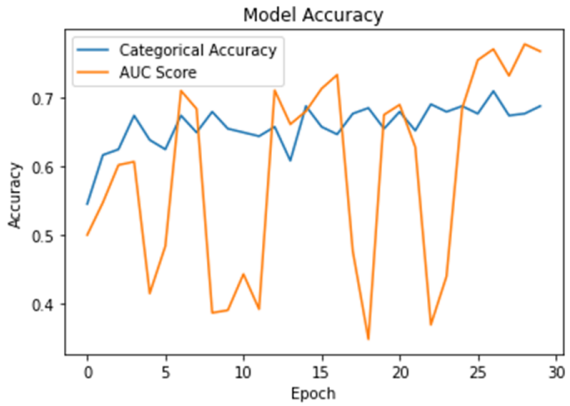

Figure 25. Model Accuracy Plot of Dense k = 64 Train 1

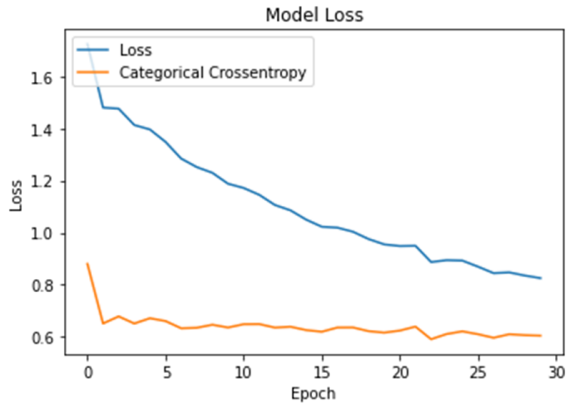

Figure 26. Model Loss Plot of Dense k = 64 Train 1

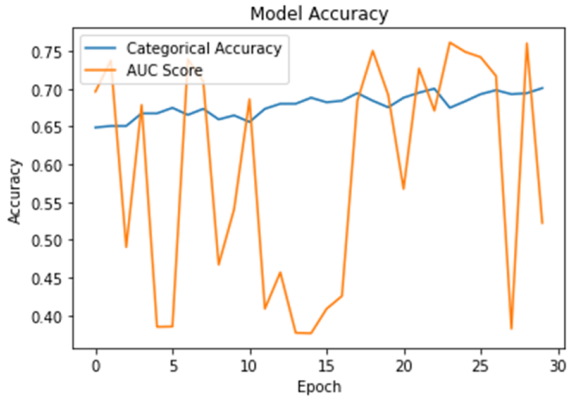

Figure 27. Model Accuracy Plot of Dense k = 64 Train 2

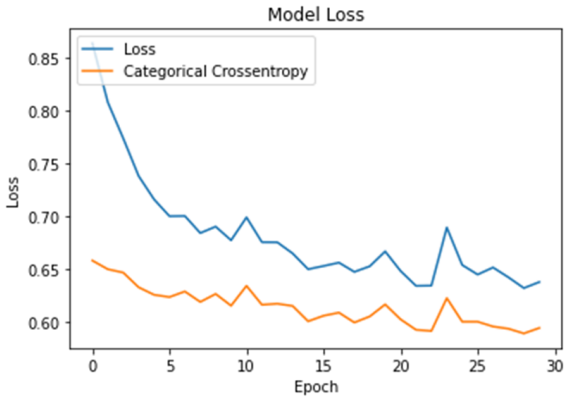

Figure 28. Model Loss Plot of Dense k = 64 Train 2

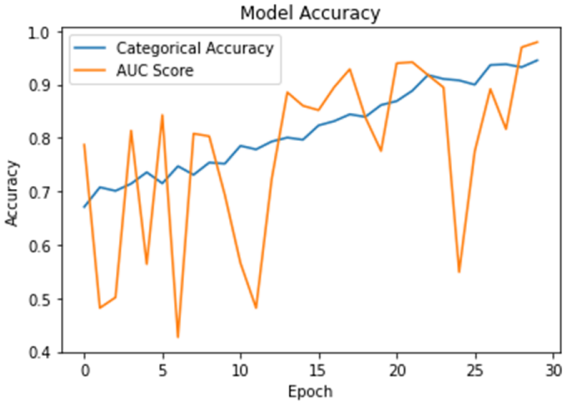

Figure 29. Model Accuracy Plot of Dense k = 16 Train 2

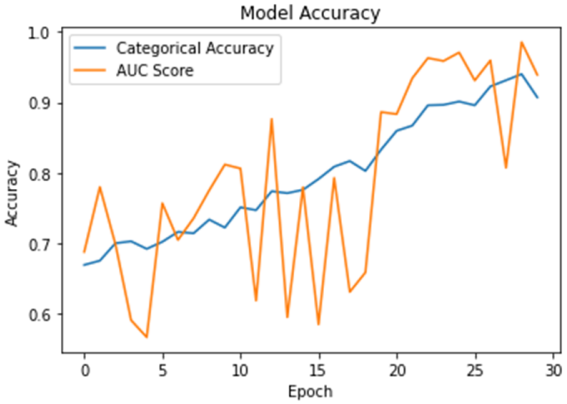

Figure 30. Model Accuracy Plot of Dense k = 8 Train 2

Model	AUC_avg	AUC_max	Running Time
Dense k = 8 Train 1	0.6656	0.799	13 min 09 sec
Dense k = 8 Train 2	0.769	0.9862	17 min 44 sec
Dense k = 16 Train 1	0.6697	0.8534	21 min 13 sec
Dense k = 16 Train 2	0.7735	0.9792	28 min 26 sec
Dense k = 64 Train 1	0.5412	0.8124	25 min 32 sec
Dense k = 64 Train 2	0.5898	0.7615	36 min 34 sec

Table 6. DenseNet Training Table

Model	AUC_avg	AUC_max	AUC_mix
Dense k = 8 Test 1	0.5366	0.6696	0.6732
Dense k = 8 Test 2	0.5503	0.686	0.68
Dense k = 16 Test 1	0.5354	0.6724	0.678
Dense k = 16 Test 2	0.514	0.661	0.679
Dense k = 64 Test 1	0.5804	0.6728	0.692
Dense k = 64 Test 2	0.5976	0.7012	0.7108

Table 7. DenseNet Testing Table

Comparing different versions of DenseNet, we get the following results:

- *AUC_avg Train 2*

(k = 64) < (k = 8) < (k = 16)

- *Running Time Train 2*

(k = 8) < (k = 16) < (k = 64)

- *AUC_avg Test 1*

(k = 8) ≈ (k = 16) < (k = 64)

- *AUC_avg Test 2*

(k = 16) < (k = 8) < (k = 64)

- *AUC_mix Test 1*

(k = 8) < (k = 16) < (k = 64)

- *AUC_mix Test 2*

(k = 8) ≈ (k = 16) < (k = 64)

It seems that the growth rate doesn't have much correlation with the training accuracy of the DenseNet model. When k = 64, the model gets the lowest *AUC_avg* on both training procedures.

However, with the increase of "k", we find that DenseNet is doing gradually better in testing. "k = 8" and "k = 16" models have pretty close performance on testing. The AUC scores retrieved from k = 64, no matter in which testing procedure, is always around 0.05 - 0.1 greater than the other two growth rate. The only two AUC scores which break the bottleneck of 0.7 in all our experiments both come from Dense k = 64 during *Test 2.* **0.7108** is also the best AUC score we reach with all the models and combinations of techniques.

Since DenseNet alleviates the vanishing-gradient problem, strengthens feature propagation, encourages feature reuse, and substantially reduces the number of parameters[11], it is not surprising that this model outperforms the other 3 in testing.

DenseNet and ResNet Comparison

1. Running Time

DenseNet is significantly more computationally heavy than ResNet. The model with the lowest growth rate k = 8 in DenseNet takes 15-20 minutes to complete a single train, which is almost the same as training the most complex ResNet, Res101.

Even though k = 64 gives us the best prediction in all the models, the second training takes around 40 minutes even with GPU. The huge demand for computational resources of DenseNet also prevents us from experimenting with a larger growth rate than 64.

This could partly resort to the reason that DenseNet uses a more complex connectivity pattern between their layers. In a DenseNet, each layer is connected to every other layer in a feedforward fashion, whereas in a ResNet, the layers are connected in a more simple and efficient pattern and each layer uses fewer filters.

2. Train Accuracy

With our novel pre-training technology, both models get a more accurate training accuracy. However, DenseNet's AUC score on the validation data is substantially lower than ResNet's. From the model accuracy figures of DenseNet, we could easily tell that the AUC score fluctuates dramatically and could not converge after 30 epochs. While in Figure 18, ResNet18's AUC score converges after 5 epochs and maintains at a high level after that. Thus, we infer that ResNet has better training performance than DenseNet on this certain dataset. ResNet also takes more advantage of pre-training.

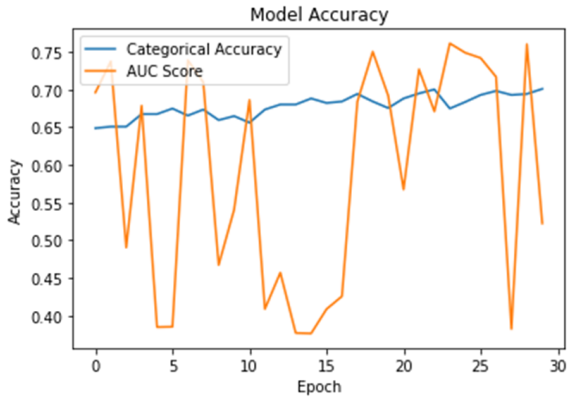

Figure 31. Model Accuracy Plot of Dense k = 64 Train 2

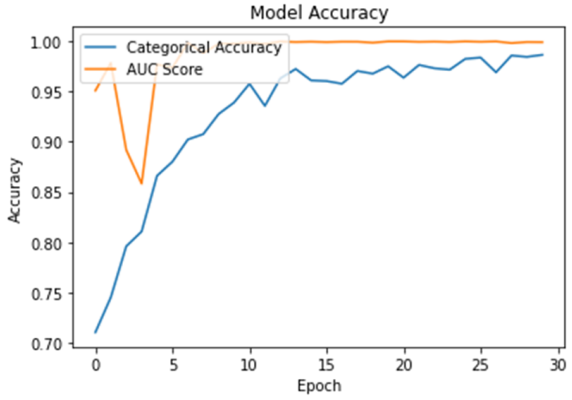

Figure 32. Model Accuracy Plot of ResNet18 Train 2

7. Conclusions & Future work

Our project takes several hundred 3D images of the CT scan from the patient's lungs. We then train our multiple deep-learning models and use them to predict whether another set of patients, whose lung scan images as the testing data, have malicious lung nodules or not. We then compare all the testing sample results with the real(true) answer provided by physicians. We use the AUC score to determine the performance of different models. The overall goal of our project is to improve the accuracy, which means we want our model to be as consistent as possible with the judgments made by physicians.

In order to overcome the insufficiency of the train set, we first use 3 different methods of data augmentation. We find that the augmented dataset could alleviate the speed of overfitting. We implement 4 different deep-learning models with 3D structures. For ResNet and DenseNet, we also set the hyperparameters changeable. Then we train each model using the original dataset for the first time and find the weight which produces the best AUC score on the validation data. This weight is then used as the initial weight to train on the data-augmented samples. Finally, we test our trained models with 100 unseen samples split from the original dataset to evaluate each model's performance. The technique to average the results from different epochs, which is borrowed from Ensemble Learning, is used to further increase the final AUC score.

From the detailed extrapolation of the experiment results, we proved that the training and testing techniques we used not only improve both the training and testing accuracy but also reduce the consumption of computing resources. The two more advanced and complicated neural networks, ResNet and DensNet, outperform Inception Net, which still beats naive CNN. ResNet is better during training and DenseNet is better during testing. The best AUC scores that different ResNet and DenseNet versions achieved are from 0.65 to 0.70. DenseNet with a growth rate of 64 reached the highest score of 0.7108.

If we have more time, we would continue researching in the following areas:

- Try additional data augmentation methods, including further tuning of *mixup*'s coefficient instead of taking the average. The goal is to further mitigate overfitting while improving model performance on the test set.
- Train the model more than two times, and mix up the result of more than two epochs' model. We can also tune the proportion of each result involved. Because when the number of training is large enough, the effect of a certain number of prediction errors is negligible. For samples that are very difficult to predict, the results will eventually stabilize at around 0.5, which will not produce crucial errors.
- Do more hyperparameter tuning on the models such as activation function, loss function, and batch size, to further improve the model and solve overfitting.
- Find more cutting-edge deep learning models that allow us to take more parameters and solve more difficult image classification problems.
- Test our techniques and models on a larger open-source dataset so that we can possibly compare our results with other people's work.

Reference

1. Torre LA, Bray F, Siegel RL, et al. Global cancer statistics, 2012. CA Cancer J Clin. 2015;65(2):87–108. doi: 10.3322/caac.21262.

2. Sui Y, Wei Y, Zhao D. Computer-aided lung nodule recognition by SVM classifier based on combination of random undersampling and SMOTE. Comput Math Methods Med. 2015;2015:368674. doi: 10.1155/2015/368674.

3. Sahiner B, Chan HP, Hadjiiski LM, et al. Effect of CAD on radiologists' detection of lung nodules on thoracic CT scans: analysis of an observer performance study by nodule size. Acad Radiol. 2009;16(12):1518–1530. doi: 10.1016/j.acra.2009.08.006.

4. Xia Y, Lu S, Wen L, et al. Automated identification of dementia using FDG-PET imaging. Biomed Res Int. 2014;2014:421743. doi: 10.1155/2014/421743.

5. H. Jiang, H. Ma, W. Qian, M. Gao and Y. Li, "An Automatic Detection System of Lung Nodule Based on Multigroup Patch-Based Deep Learning Network," in IEEE Journal of Biomedical and Health Informatics, vol. 22, no. 4, pp. 1227-1237, July 2018, doi: 10.1109/JBHI.2017.2725903.

6. D. Kumar, A. Wong and D. A. Clausi, "Lung Nodule Classification Using Deep Features in CT Images," 2015 12th Conference on Computer and Robot Vision, 2015, pp. 133-138, doi: 10.1109/CRV.2015.25.

7. Johnsirani Venkatesan, Nikitha, ChoonSung Nam, and Dong Ryeol Shin. "Lung nodule classification on CT images using deep convolutional neural network based on geometric feature extraction." Journal of Medical Imaging and Health Informatics 10.9 (2020): 2042-2052.

8. Tran GS, Nghiem TP, Nguyen VT, Luong CM, Burie JC. Improving Accuracy of Lung Nodule Classification Using Deep Learning with Focal Loss. J Healthc Eng. 2019 Feb 4;2019:5156416. doi:

10.1155/2019/5156416. PMID: 30863524; PMCID: PMC6378763.

9. Jiang W, Zeng G, Wang S, Wu X, Xu C. Application of Deep Learning in Lung Cancer Imaging Diagnosis. J Healthc Eng. 2022 Jan 3;2022:6107940. doi: 10.1155/2022/6107940. PMID: 35028122; PMCID: PMC8749371.

10. Sverzellati N, Silva M, Calareso G, et al. Low-dose computed tomography for lung cancer screening: comparison of performance between annual and biennial screen. Eur Radiol, 2016, 26(11): 3821-3829

11. Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger Densely Connected Convolutional Networks Computer Vision and Pattern Recognition (cs.CV); Machine Learning (cs.LG)

12. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

13. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

14. He, K., Zhang, X., Ren, S., & Sun, J. (2016, October). Identity mappings in deep residual networks. In European conference on computer vision (pp. 630-645). Springer, Cham.