CoronaHack -Chest X-Ray-Dataset
A Comparative Study Using Different CNN architecture

Ka Wing Yan
University of California, San Diego
w2yan@ucsd.edu

Kai Akamatsu
University of California, San Diego
kakamatsu@ucsd.edu

Abstract

Computer vision technologies have the potential to transform the field of medicine. By utilizing a collection of 5,910 chest X-ray images, we compare the performance of five different convolutional neural network (CNN) architectures in classifying healthy and pneumonia-infected patients. We evaluate the accuracy of a simple CNN architecture, Resnet CNN, VGG16 CNN, UNET CNN, and pre-trained Resnet-18. The pre-trained Resnet-18 model outperformed simple CNN (74.68%), Resnet (83.17%), VGG16 (77.88%), and UNET (82.69%) with a final testing set accuracy of 84.94%. This study demonstrates the potential for Resnet CNNs to classify medical images to enhance healthcare.

1 Problem and Motivation

Problem Statement

Can Convolutional Neural Networks (CNNs) categorize chest X-ray images into two distinct categories: healthy and pneumonia-afflicted? This question explores the ability of CNNs to accurately differentiate between normal and pathological chest X-rays, potentially assisting in the automation of medical diagnoses and improving the speed and accuracy of identifying pneumonia in clinical settings.

1.1 Motivation

COVID-19 is an infectious disease caused by the SARS-CoV-2 virus. Globally, there has been a total of 702,412,742 COVID-19 cases with 6,974,185 deaths [9]. During the peak of this pandemic, many hospitals did not have enough space in the ICU to treat all COVID-19 patients effectively. The CDC estimated that as hospitals exceed max ICU bed capacity, approximately 80,000 excess deaths are expected [1]. This brings to light an urgent issue: we need a more efficient method to identify, diagnose, and contain COVID-19. Pneumonia, inflammation of the lungs, is a common complication of COVID-19 and physicians can often detect it with chest X-rays. This proposes an opportunity to utilize computer vision approaches to analyze chest X-ray data.

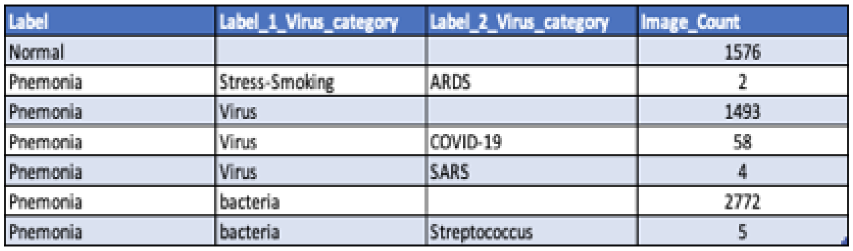
2 Dataset

Dataset summary

We utilized a collection of chest x-rays of healthy and pneumonia-affected patients [**?**].

- Format: 5910 JPEG images
- Classification: Normal vs Pneumonia / Virus vs Bacteria
- Dataset type: 90% for training and 10% for testing.

In this study, we are interested in classifying normal and pneumonia patients.

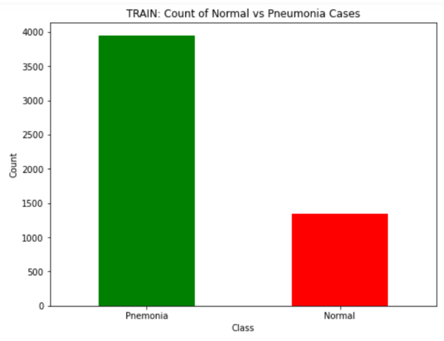


Label	Label_1_Virus_category	Label_2_Virus_category	Image_Count
Normal			1576
Pnemonia	Stress-Smoking	ARDS	2
Pnemonia	Virus		1493
Pnemonia	Virus	COVID-19	58
Pnemonia	Virus	SARS	4
Pnemonia	bacteria		2772
Pnemonia	bacteria	Streptococcus	5

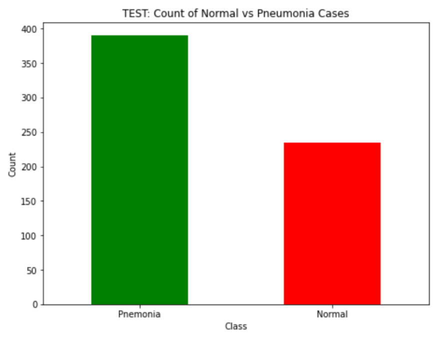
Figure 1: Dataset Overview

Dataset details

We downloaded the dataset from the CoronaHack -Chest X-Ray-Dataset project on Kaggle [4]. The dataset was already split into the training and testing folds. Of the 5910 images, 5286 were in the training fold and 624 were in the testing fold. We investigated the number of normal and pneumonia images in each of the training and testing folds. In the training dataset, 3944 images were from pneumonia-affected patients and 1342 images were from normal patients (Figure 2a). In the testing dataset, 390 images were from pneumonia-affected patients and 234 images were from normal patients (Figure 2b). Although the ratio of pneumonia and normal images are disproportionate in both the training (75%) and testing (62.5%) datasets, we moved forward with this split because the ratios were relatively similar across the two folds. We set aside 20% of the training data to create a validation dataset.



(a) Training Dataset



(b) Testing Dataset

Figure 2: Comparison of Training and Testing Datasets

3 Method

All of our methods outlined below were performed in Python using the PyTorch package [7].

3.1 Data Augmentation

We utilized five forms of data augmentation (Table 1):

- **Resizing:**
  This transformation resizes the input image to 256x256 pixels. Resizing is a common practice in image processing for neural networks, as it ensures that all images have the same dimensions.
- **Flipping:**
  Randomly flips the image horizontally/vertically (with a default 50% chance). Horizontal flipping is a popular data augmentation technique because it increases the diversity of the training data without losing relevant features or altering the labels of the images. This can help reduce overfitting and improve the model's generalization by simulating variability in the dataset.
- **Cropping:**
  This transformation crops the central part of the image, making it 256x256 pixels if it isn't already. This can be used to remove extraneous parts of the image that are less likely to contain the object of interest, focusing the model's attention on the central part of the image.
- **Converting to tensors:**
  This transformation converts the image, which is typically read in as a PIL Image or a NumPy array, into a PyTorch tensor. This is necessary because PyTorch models expect input data in the form of tensors.
- **Normalize:**
  Normalizes each channel of the image (RGB) to have a specific mean and standard deviation, provided in the code. Standardizing the range of data values ensures that each feature contributes equally to training and helps speed up the convergence of the neural network. The specific values for mean and standard deviation are usually calculated from the entire training dataset and are hard coded.

Transformation	**Description**
`T.Resize(256)`	Resizes the input images to 256x256 pixels.
`T.RandomHorizontalFlip()`	Applies a random horizontal flip to the images with a default probability of 0.5.
`T.RandomVerticalFlip()`	Applies a random vertical flip to the images with a default probability of 0.5.
`T.CenterCrop(256)`	Crops the central part of the image to make a square of 256x256 pixels.
`T.ToTensor()`	Converts the images from PIL Image or NumPy arrays into PyTorch tensors.
`T.Normalize(...)`	Normalizes each color channel of the images. The means are (0.5071, 0.4867, 0.4408) and the standard deviations are (0.2675, 0.2565, 0.2761).

Table 1: Data augmentation transformations

Data augmentations can improve the generalization performance of CNNs by increasing the effective size of the training data and introducing variations in the training data that mimic scenarios in the testing dataset.

3.2 Why CNN?

Below we outline some key reasons why we chose to utilize CNNs:

- **Specialization in Image Processing**: CNNs are tailored for processing pixel data, making them ideal for image recognition tasks.
- **Feature Extraction Capabilities**: These networks are adept at automatically detecting critical features from images, essential for accurate medical diagnoses.
- **Proven Effectiveness**: CNNs have a strong track record in medical image analysis, demonstrating their reliability and accuracy.
- **Robustness to Variations**: They handle variations in images, such as angles, lighting, and body sizes.
- **Transfer Learning Capabilities**: Leveraging pre-trained models enhances performance, especially with limited data.

3.3 Simple CNN

Layer	In-channels	Out-channels	Kernel-size	Padding	Stride	Activation Function
conv1	3	32	3	1	1	ReLU
pool	-	-	2	0	2	-
conv2	32	64	3	1	1	ReLU
pool	-	-	2	0	2	-
flatten	-	-	-	-	-	-
fc1	262144	512	-	-	-	ReLU
fc2	512	num_classes	-	-	-	-

Table 2: CNN Architecture for Pneumonia Classification

Hyperparameter	Value
Number of Classes	2 (Normal, Pneumonia)
Learning Rate	0.001
Number of Epochs	10
Batch Size	32
Optimizer	Adam
Weight Decay	1e-4
Data Augmentation	Yes
Early Stopping	Yes
Early Stopping Patience	5
Scheduler	Yes (Learning Rate Scheduler)

Table 3: Hyperparameters used in the CNN models

We have structured the Simple CNN model as a series of two convolutional $\rightarrow$ max-pooling layers, leading into a fully connected layer and an activation function. The Adam optimizer [5], known for its efficiency in handling sparse gradients on noisy problems, was employed with an initial learning rate of 0.001. To optimize the learning process further, we implemented a learning rate scheduler that adjusts the rate based on the training epoch. The training process was set for a total of 10 epochs; however, the early stop mechanism was triggered after 6 epochs to prevent overfitting and to ensure the model's generalization.

3.4 Resnet CNN

Layer Type	In-Channels	Out-Channels	Kernel Size	Stride	Padding	Additional Info
Convolution	3	64	7x7	2	3	Initial layer
Max Pooling	-	-	3x3	2	1	-
Two ResNet Blocks	-	64	3x3	1	1	Includes BatchNorm, ReLU
Two ResNet Block	-	128	3x3	1	1	Includes BatchNorm, ReLU
Two ResNet Block	-	256	3x3	1	1	Includes BatchNorm, ReLU
Two ResNet Block	-	512	3x3	2	1	Includes BatchNorm, ReLU
Adaptive Avg Pool	-	-	-	-	-	Output size: 1x1
Fully Connected	512	num_classes	-	-	-	-

Table 4: Architecture of the ResNet Model

Hyperparameter	Value
Number of Classes	2 (Normal, Pneumonia)
Learning Rate	0.0001
Number of Epochs	10
Batch Size	32
Optimizer	Adam
Weight Decay	None
Data Augmentation	Yes
Early Stopping	Yes
Early Stopping Patience	5
Scheduler	None

Table 5: Hyperparameters used in the Resnet models

We utilized the Resnet CNN architecture with 2 of the 64-channel output, 128-channel output, 256-channel output, and 512-channel output Resnet blocks [2]. Each Resnet block consists of two convolutional layers and a residual connection from the input of the block to the output. We utilized the Adam optimizer [5] with a learning rate 0.0001 to train the model. This model was trained for 10 epochs.

3.5 VGG16 CNN

We employed the VGG16 CNN architecture, a model known for its deep and homogeneous structure consisting primarily of 3x3 convolutional layers and max-pooling layers. The model was structured into five main blocks: the first two contain two convolutional layers with 64 and 128 channels respectively, followed by three blocks each containing three convolutional layers with increasing channel depths of 256, 512, and 512. Each convolutional layer is followed by a ReLU activation function to introduce non-linearity, and each block concludes with a max-pooling layer to reduce spatial dimensions and enhance feature extraction [8].

For the training regimen, we utilized the Adam optimizer [5], with a set learning rate of 0.001. The VGG16 model was subjected to a training duration of 10 epochs.

Layer Type	**In-Channels**	**Out-Channels**	**Kernel Size**	**Stride**	**Padding**	**Additional Info**
Conv2d + ReLU	3	64	3x3	1	1	Block 1
Conv2d + ReLU	64	64	3x3	1	1	Block 1
MaxPool2d	-	-	2x2	2	0	Block 1
Conv2d + ReLU	64	128	3x3	1	1	Block 2
Conv2d + ReLU	128	128	3x3	1	1	Block 2
MaxPool2d	-	-	2x2	2	0	Block 2
Conv2d + ReLU	128	256	3x3	1	1	Block 3
Conv2d + ReLU	256	256	3x3	1	1	Block 3
Conv2d + ReLU	256	256	3x3	1	1	Block 3
MaxPool2d	-	-	2x2	2	0	Block 3
Conv2d + ReLU	256	512	3x3	1	1	Block 4
Conv2d + ReLU	512	512	3x3	1	1	Block 4
Conv2d + ReLU	512	512	3x3	1	1	Block 4
MaxPool2d	-	-	2x2	2	0	Block 4
Conv2d + ReLU	512	512	3x3	1	1	Block 5
Conv2d + ReLU	512	512	3x3	1	1	Block 5
Conv2d + ReLU	512	512	3x3	1	1	Block 5
MaxPool2d	-	-	2x2	2	0	Block 5
Linear + ReLU + Dropout	512 * 8 * 8	4096	-	-	-	Classifier
Linear + ReLU + Dropout	4096	4096	-	-	-	Classifier
Linear	4096	num_classes	-	-	-	Classifier

Table 6: Layer architecture of the VGG16 model

Hyperparameter	Value
Number of Classes	2 (Normal, Pneumonia)
Learning Rate	0.001
Number of Epochs	10
Batch Size	32
Optimizer	Adam
Weight Decay	1e-4
Data Augmentation	Yes
Early Stopping	Yes
Early Stopping Patience	5
Scheduler	None

Table 7: Hyperparameters used in the Vegg16 models

3.6 UNET CNN

Layer Type	In-Channels	Out-Channels	Kernel Size	Stride	Padding	Additional Info
Conv2d + ReLU	3	64	3x3	1	1	Encoder Block 1
MaxPool2d	-	-	2x2	2	0	-
Conv2d + ReLU	64	128	3x3	1	1	Encoder Block 2
MaxPool2d	-	-	2x2	2	0	-
Conv2d + ReLU	128	256	3x3	1	1	Encoder Block 3
MaxPool2d	-	-	2x2	2	0	-
Conv2d + ReLU	256	512	3x3	1	1	Encoder Block 4
MaxPool2d	-	-	2x2	2	0	-
Conv2d + ReLU	512	1024	3x3	1	1	Encoder Block 5
MaxPool2d	-	-	2x2	2	0	-
ConvTranspose2d + ReLU	1024	512	2x2	2	0	Decoder Block 1
Conv2d + ReLU	1024	512	3x3	1	1	Concat with Encoder Block 4
ConvTranspose2d + ReLU	512	256	2x2	2	0	Decoder Block 2
Conv2d + ReLU	512	256	3x3	1	1	Concat with Encoder Block 3
ConvTranspose2d + ReLU	256	128	2x2	2	0	Decoder Block 3
Conv2d + ReLU	256	128	3x3	1	1	Concat with Encoder Block 2
ConvTranspose2d + ReLU	128	64	2x2	2	0	Decoder Block 4
Conv2d + ReLU	128	64	3x3	1	1	Concat with Encoder Block 1
Global Avg Pool	-	-	-	-	-	Before final classification
Conv2d	64	n_class	1x1	1	0	Final classification layer

Table 8: Layer architecture of the adapted UNet model

Hyperparameter	Value
Number of Classes	2 (Normal, Pneumonia)
Learning Rate	0.001
Number of Epochs	10
Batch Size	32
Optimizer	Adam
Weight Decay	1e-4
Data Augmentation	Yes
Early Stopping	Yes
Early Stopping Patience	10
Scheduler	Yes (Learning Rate Scheduler)

Table 9: Hyperparameters used in the UNET models

Transfer Learning Resnet 18

We obtained pre-trained Resnet 18 weights from torchvision models. In order to adjust the Resnet 18 model to binary classification, we replaced the last fully connected layer to have an output size of two. We created two variations of the transfer learning Resnet 18 model.

In our first model, which we call 'Fine-tune Resnet18 with freeze,' we freeze all of the layers except the last Resnet block and fully connected layer. To train this model, we used the Adam optimizer [5] with a learning rate 0.0001 and a learning rate scheduler. This model was trained for 6 epochs due to an early stop.

In the second model, which we call 'Fine-tune Resnet18 without freeze,' we don't freeze any of the Resnet weights. To train this model, we used the Adam optimizer [5] with a learning rate 0.00001. This model was trained for 7 epochs due to an early stop.

4 Results

Before we trained the CNN models, we visualized the chest X-ray images to see if there are clear differences between Normal and Pneumonia-labeled images. Although the distinctions are often subtle, pneumonia-infected patients have large white clouds surrounding the lung tissues (Figure 3). These are denser clusters of pus-filled alveoli, called infiltrates [6]. This confirms that there are visible differences in the chest X-rays between normal and infected patients that CNN models can model.

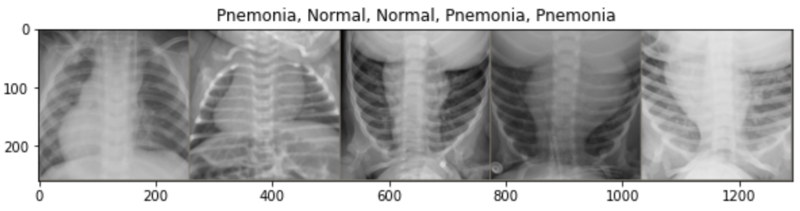
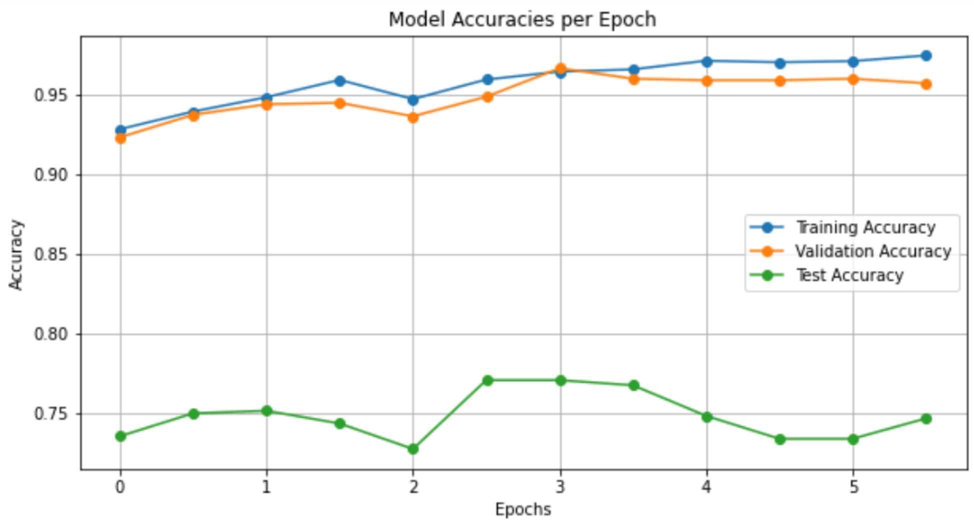
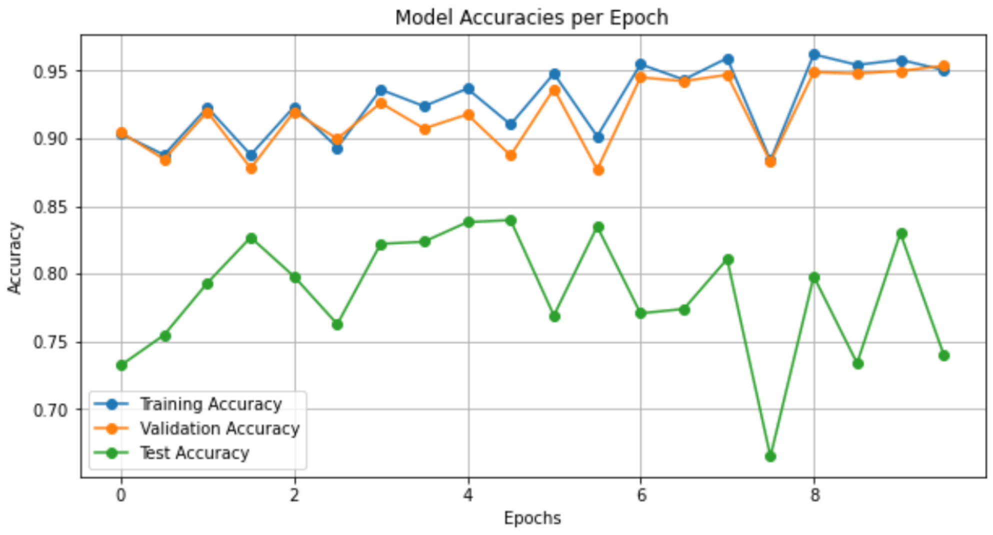


Figure 3: X-ray images

To compare the results of various CNN models, we recorded the training, validation, and testing accuracies at each epoch. Figures 4 and 5 show the accuracy of SimpleCNN, Resnet, VGG16, and UNET at each epoch. In all of these CNN models, the training and validation accuracies are much higher than the testing accuracies. This is expected because the testing dataset is a completely unseen dataset. All of the CNN models achieve a >90% accuracy on the validation dataset by the last epoch.



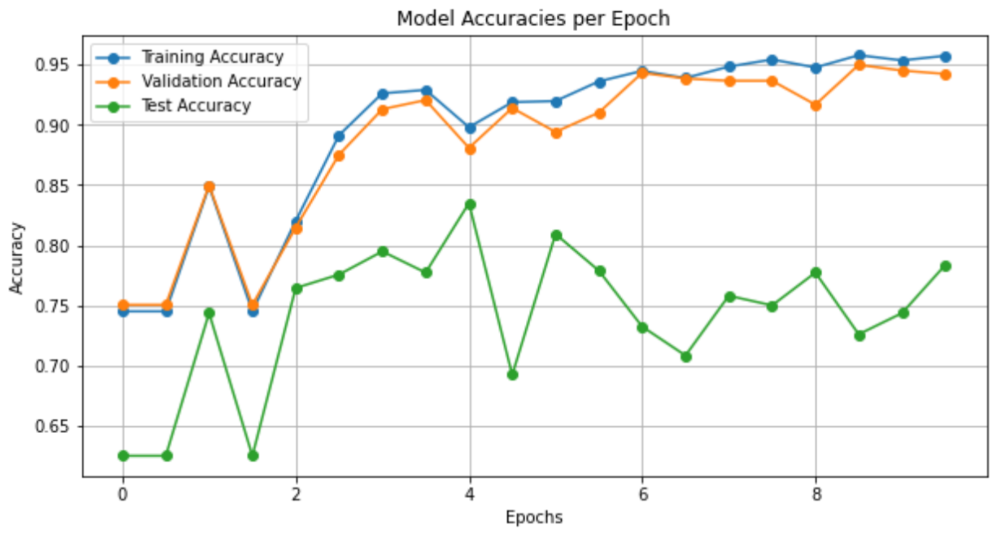
(a) Simple CNN



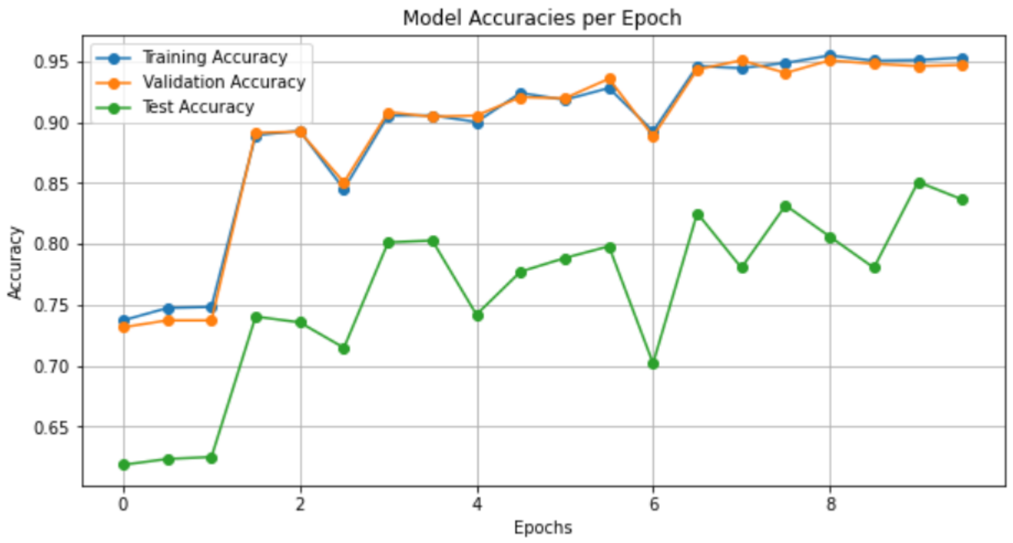
(b) Resnet CNN

Figure 4: Comparison of Simple CNN and Resnet CNN Architectures

The training and validation accuracy at the last epoch is similar across the Simple CNN, Resnet CNN, VGG16 CNN, and UNET CNN, reaching about 95% accuracy. However, the accuracy of predictions on the testing dataset varied between the architectures. The Simple CNN and VGG16 seemed to be the most overfit on the training data, only reaching a 74.68% accuracy and 77.88% accuracy on the testing dataset respectively (Figure 4a, 5a, and 7). The Resnet CNN and UNET CNN outperformed the alternative architectures with a 83.17% and 82.69% accuracy on the testing dataset respectively (Figure 4b, 5b, and 7)



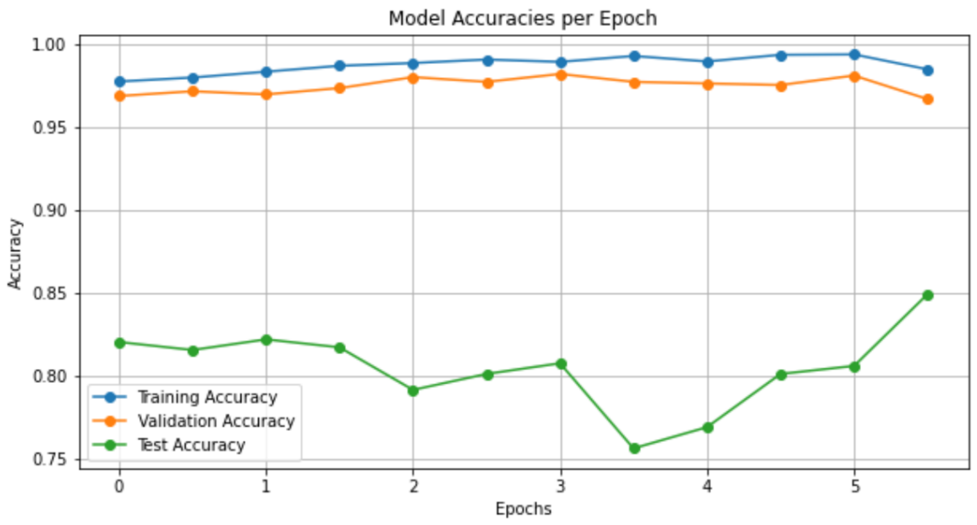
(a) VGG 16 CNN



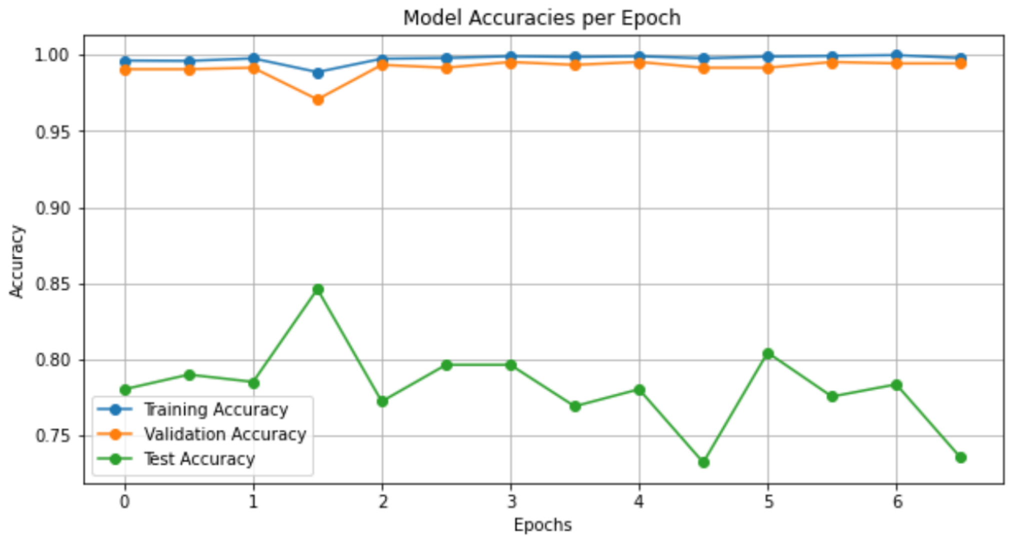
(b) UNET CNN

Figure 5: Comparison of VGG16 CNN and UNET CNN Architectures

Both of the Resnet 18 transfer learning models achieved a >98% validation accuracy by the last epoch (Figures 6). We hypothesized that freezing some many layers and only training the last Resnet block will result in a less overfit model that generalizes better on the testing data. Indeed, the fine-tuned Resnet 18 model with freeze achieved a higher testing accuracy (84.94%) than without the freeze (73.56%) (Figure 7).



(a) Fine-tune Resnet 18 with Freeze



(b) Fine-tune Resnet 18 without Freeze

Figure 6: Comparison of Transfer Learning Models

In order to evaluate the performance of each model in real medical practice we compared the accuracies achieved on the final testing dataset. The fine-tune Resnet 18 model with freeze achives the highest test dataset accuracy of 84.94% (Figure 7).

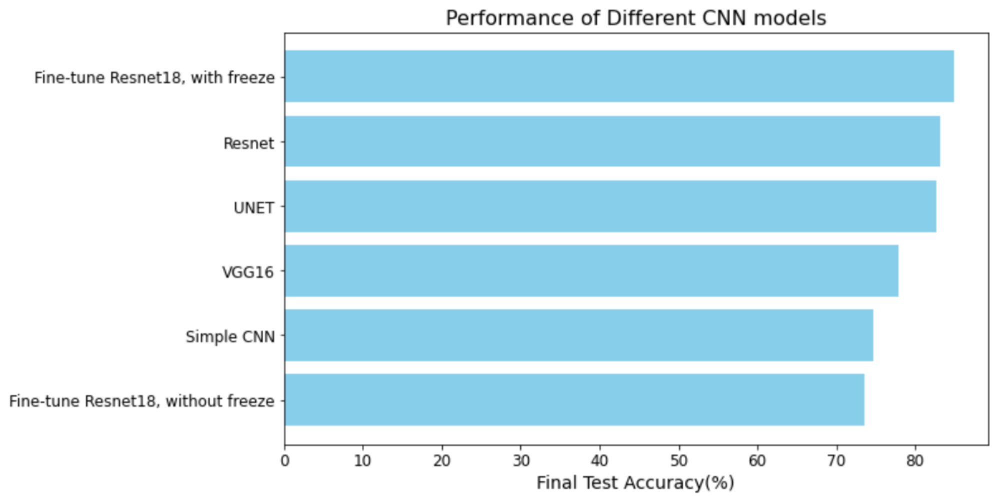


Figure 7: Comparison of all of the models

5 Discussion

This study compared the performance of five different Convolutional Neural Network (CNN) architectures—Simple CNN, ResNet CNN, VGG16 CNN, UNET CNN, and fine-tuned ResNet—in classifying chest X-ray images into normal and pneumonia-affected categories. Our results revealed the pre-trained ResNet-18 model, when fine-tuned, outperformed the other architectures, achieving the highest testing set accuracy of 84.94%. This aligns with existing research suggesting that pre-trained models, due to their extensive prior learning, can be highly effective in medical image classification tasks, particularly when fine-tuned to specific datasets [3].

The Simple CNN, despite its less complex architecture, showed commendable performance, suggesting that even basic models can provide significant insights when dealing with distinct image characteristics between normal and pneumonia-infected patients. However, the disparity in performance between the Simple CNN and more advanced models like ResNet and VGG16 underscores the importance of deep architectural features in capturing complex patterns within medical imagery.

Notably, the UNET CNN, typically used for segmentation tasks, performed well in this study. Its ability to capture both coarse and fine feature information underscores its potential for classification tasks in medical imaging.

The study also addresses an important topic: data imbalance between normal and pneumonia images. This situation poses challenges for model training, as imbalanced datasets can lead to biased predictions favoring the majority class. Despite this, our models achieved high validation accuracies and acceptable testing accuracies, suggesting that our data augmentation and model training strategies effectively mitigated some of the issues arising from dataset imbalance.

Finally, we recapitulate the strength of transfer learning in various computer vision applications. Fine-tuning the last few layers of a pre-trained ResNet model resulted in an impressive testing dataset accuracy.

5.1 Pros and Cons in Different CNN

Simple CNN

Pros:

- Simple architecture makes it easy to understand and implement.
- Requires fewer computational resources, making it faster to train.
- Less prone to overfitting compared to more complex models.

Cons:

- Limited in capturing complex patterns due to simpler structure.
- Lower performance on highly complex image classification tasks.
- May not generalize well to new, unseen data compared to deeper networks.

ResNet CNN

Pros:

- Utilizes residual connections to enable training of very deep networks.
- Improves gradient flow and alleviates the vanishing gradient problem.
- Often achieves superior performance on various image recognition tasks.

Cons:

- More complex to understand and implement compared to simpler models.
- Pre-trained models available, making it quick to implement for transfer learning.
- Requires more computational resources for training and inference.
- Large number of layers can lead to higher memory usage and overfitting.

VGG16 CNN

Pros:

- Simple and uniform architecture makes it easy to modify and understand.
- Pre-trained models available, making it quick to implement for transfer learning.
- Proven effectiveness on a range of image processing tasks.

Cons:

- Very deep network, leading to high computational cost.
- Large model size due to the number of parameters.
- Can be more prone to overfitting compared to models with regularization strategies.

UNET CNN

Pros:

- Specialized for image segmentation tasks with its encoder-decoder structure.
- Capable of capturing multi-scale contextual information.
- Efficient for medical image segmentation and other similar applications.

Cons:

- Primarily designed for segmentation.
- Complex architecture requires significant computational resources.
- Training can be challenging due to the intricate structure.

5.2 Future Directions

Instead of classifying healthy and pneumonia patients, we are also interested in exploring if we can distinguish viral and bacterial pneumonia. Enhancing the specificity of pneumonia classification could significantly improve the utility of CNN in medical practices.

6 Authors' Contribution

Ka Wing Yan: Visualization,Simple CNN, Vgg16,Unet , building up the training model and implenment the scheduler, optimizer,early stop, Data augmentation and accuarcy model build up.
Kai Akamatsu: Background research, data loading, data cleaning, ResNet CNN, transfer learning, final result visualization, final report.

References

[1] Centers for Disease Control and Prevention (CDC). Impact of hospital strain on excess deaths during the covid-19 pandemic — united states, july 2020–july 2021. https://www.cdc.gov/mmwr/volumes/70/wr/mm7046a5.htm, 2021.

[2] Kaiming He et al. Deep residual learning for image recognition. https://arxiv.org/abs/1512.03385, 2015.

[3] Nandhini Santhanam Mahboubeh Jannesari Mate E. Maros Thomas Ganslandt Hee E. Kim, Alejandro Cosa-Linan. Transfer learning for medical image classification: a literature review. *BMC Medical Imaging*, 2022.

[4] Kaggle. Coronahack -chest x-ray-dataset. https://www.kaggle.com/datasets/praveengovi/coronahack-chest-xraydataset, NA.

[5] Diederik P Kingma et al. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2014.

[6] Ming-Yen Ng, Elaine Yuen Phin Lee, Ya-Seng Arthur Yang, Ching-Yao Yang, Chao-Yu Tsai, Li-Ya Huang, Yu-Cheng Wang, Pei-Yun Yang, Jin-Shing Chen, and Tzung-Hai Yen. Imaging profile of the covid-19 infection: Radiologic findings and literature review. *Radiology: Cardiothoracic Imaging*, 2020.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. https://arxiv.org/abs/1409.1556, 2014.

[9] WORLDOMETER. Covid-19 coronavirus pandemic. https://www.worldometers.info/coronavirus/#google_vignette.