AMERICAN UNIVERSITY OF BEIRUT

Pneumonia Detection Using Capsule Networks

by

Ali Yaghi Zaiter

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Science
to the Department of Computer Science
of the Faculty of Arts and Science
at the American University of Beirut

Beirut, Lebanon
February 2020

AMERICAN UNIVERSITY OF BEIRUT

Pneumonia Detection Using Capsule Networks

by
Ali Yaghi Zaiter

Approved by:

Dr. Shady Elbassuoni, Assistant Professor Computer Science	Advisor [illegible]
Dr. Mohamed El Baker Nassar, Assistant Professor Computer Science	Member of Committee [illegible]
Dr. Amer Abdo Mouawad, Assistant Professor Computer Science	Member of Committee [illegible]

Date of thesis defense: February, 2020

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Zaiter Ali Yashi

Last	First	Middle

[✓ Master's Thesis] ◯ Master's Project ◯ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☒ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One ___ year from the date of submission of my thesis, dissertation or project.**
Two ___ years from the date of submission of my thesis , dissertation or project.
Three X years from the date of submission of my thesis , dissertation or project.

[illegible] 20/2/2020

Signature Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

Acknowledgements

Foremost, i would like to thank my thesis advisor, Dr. Shady Elbassuoni, for his support, guidance, and motivation. Dr. Elbassuoni has a unique pedagogical style that inspired my interest in machine learning and artificial intelligence, and since our first meeting to discuss the thesis idea, he was so encouraging and helped me throughout the stages of the project. His knowledge and comments were so helpful.

I would like also to thank the the rest of my thesis committee members, Dr. Mohamed Nassar and Dr. Amer Mouawad, for their follow-up on my work and their constructive questions.

I would like also to thank the Office of Information technology at AUB and my colleagues for the support and encouragement, especially the research computing team.

Finally, i would like to thank my family, my friends, and everyone who gave me the motivation to start in the first place, and also provided constructive feedback through out my thesis.

An Abstract of the Thesis of

Ali Yaghi Zaiter for Master of Science
Major: Computer Science

Title: Pneumonia Detection Using Capsule Networks

Pneumonia is an infection that inflames the air sacs known as alveoli in the lungs. Symptoms include fever, chest pain, cough, and shortness of breath. It affects approximately 450 million people globally (7% of the population) and results in about four million deaths per year [1].

Currently the best available method to diagnose pneumonia and other pathology types are chest X-rays. In this thesis, we present a machine learning model using capsule networks (CapsNet) to be able to detect pathology like pneumonia and other common disease types using the chest X-ray images.

CapsNet were recently proposed to address the shortcomings of the traditional convolutional neural networks (CNNs) in computer vision tasks. One of the main issues of CNNs is that they do not recognize orientation or relative spatial relationships between the different elements of the image, which is an important feature when detecting pneumonia from X-ray images. They also require a large amount of training data, which is not the case for our problem. To train and evaluate our CapsNet approach, we use a dataset of 5,855 X-ray images [2] of infected and healthy individuals (we refer to this dataset as the pneumonia dataset), and then use a larger dataset [3] of 112,120 X-ray images to detect more common disease types (we refer to this dataset as the ChestX-ray14 dataset).

Using CapsNet we were able to achieve 0.94 accuracy on the pneumonia dataset. And on the large dataset, ChestX-ray14, we were able to outperform the ResNet-50 model of Wang et. al (2017) [3] (the creators of the dataset) on average AUROC and on most disease types including pneumonia.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pneumonia is a respiratory infection of the air sacs in the lungs. These air sacs exist in clusters at the end of the breathing tube. Pneumonia causes these tiny air sacs to be inflamed and filled up with fluid. The symptoms include coughing, low energy, fever or chills, and difficulty in breathing. It is typically caused by viruses, fungi, or bacterial infection. The infection could be caused by direct contact, like hand shaking, or by inhaling droplets in the air from coughing or sneezing.

People diagnosed with pneumonia should take it seriously, even though most people recover using antibiotics and rest. 1 out of 5 adults will need a hospital visit and some may require intensive care unit (ICU) stay if they have severe infection. Severe infections may even cause death. According to the world health organization, pneumonia accounts for 15% of all deaths of children under 5 years old, killing 808,694 children in 2017 [8]. Even though it is more common with the elderly people and young children, a person can be infected at any age.

Currently pneumonia is being diagnosed through symptoms, physical examination, blood tests, and chest X-rays images. Multiple attempts were made to use machine learning to automatically detect pneumonia and other disease types using chest X-rays.

A publicly available pneumonia dataset [2] consisting of 5,855 X-ray images labeled with either pneumonia or no-pneumonia was used for a binary classification problem. Another dataset, ChestX-ray14, was introduced by Wang et. al (2017) [3]. It is the largest publicly available X-ray dataset, consisting of 112,120 x-ray images belonging to 14 disease types (including pneumonia). The dataset was constructed using natural language processing techniques to label the X-ray images based on the reports associated with each image. Wang et. al (2017)[3] used the dataset they introduced, ChestX-ray14, for training a convolutional neural network to detect the 14 disease types given an X-ray image. They used a ResNet-50 based model, pre-trained using ImageNet, and they achieved good results with average AUROC 0.74 and 0.66 AUROC for pneumonia. Another attempt was made by Yao et al. (2017) [6], to tackle the same problem on the ChestX-ray14 dataset. They used DenseNet with long short-term memory (LSTM) model and they achieved better results than Wang et. al (2017)[3], with average AUROC 0.8 and 0.71 AUROC for pneumonia. Another attempt, CheXNet [7], was made on the same dataset, ChestX-ray14, was made using 121-layers convolutional neural network, and they achieved state-of-the-art results with 0.84 average AUROC and 0.77 AUROC for pneumonia.

All the above attempts were made using variations of the regular convolutional neural networks. In this thesis, we propose a totally different approach based on newly introduced concepts. Our approach is based on capsule networks that were introduced by Geoffrey Hinton and his team [4].

On October 2017 a completely new neural network model was introduced by Geoffrey Hinton and his team. They published two papers "Dynamic Routing Between Capsules" [4] and "Matrix capsules with EM routing" [5] describing a model that aims to address the shortcomings of the traditional convolutional neural networks (CNNs).

Geoffrey Hinton is referred to by some as the "Godfather of Deep Learning". He was one of the researchers who introduced the back-propagation algorithm and the first to use it for learning word embeddings.

The main issue of CNNs is that they do not recognize orientation or relative spatial relationships between the different elements. For example, in figure 2.8 the orientation and spatial relationships between the nose, eyes, mouth, etc is irrelevant to regular CNNs. And thus, a CNN may consider both as a face.

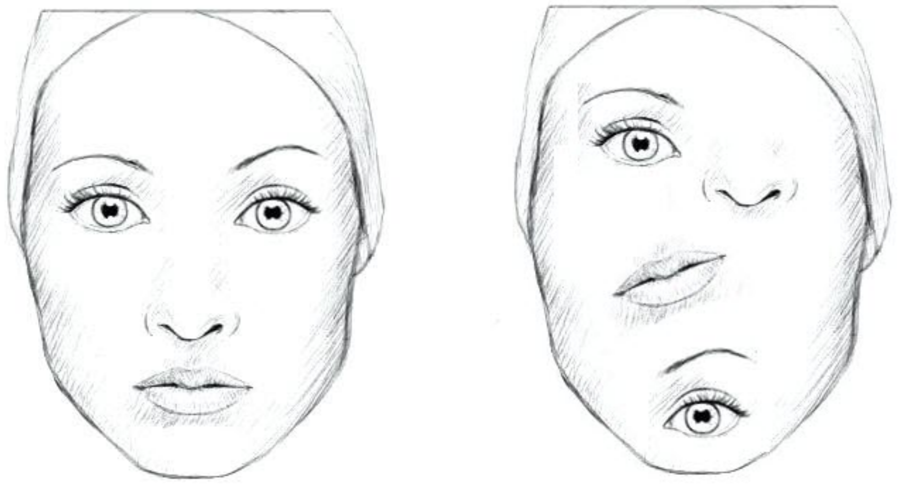


Figure 1.1: CNN Face Detection

Capsule networks can recognize the spatial relationship between the different parts of an image. A capsule outputs a vector instead of a scalar as opposed to a neuron, the vector allows the capsule to learn more than the probability of the existence of an entity. It can learn the instantiation parameters of an entity, which gives it a better tolerance for variations of the same entity and will require much less training data (variations are now being solved in CNN through augmentation). In the published papers using the smallNORB dataset [5], capsules were able to reduce the number of test errors by 45%.

Our aim in this thesis is to apply CapsNet to the detection of pneumonia using chest X-Ray Images. We will be testing our approach on two datasets. The first dataset [2] consists of 5,855 images that belong to healthy and infected people. The second dataset is the ChestX-ray14 [3], which consists of 112,120 chest X-ray images from 30,805 unique patients that are infected with one ore more of 14 different disease types.

On the pneumonia dataset [2], we were able to achieve 0.94 accuracy using CapsNet, which is better than the most voted kernel on Kaggle [9]. Using the ChestX-ray14 dataset [3], we were able to outperform Wang et. al (2017) [3] with an average AUROC of 0.7536 compared to 0.7451, and 0.6788 pneumonia AUROC compared to 0.6580 achieved by Wang et. al (2017) [3].

Chapter 2

Preliminaries

2.1 Machine Learning

Machine learning is a subset of artificial intelligence where algorithms are built to learn from the data and information. In today's life, machine learning allowed computers to interact with humans through voice recognition and natural language processing, drive cars autonomously, fraud detection, computer vision and to do many other tasks.

The below are major milestones through the history of machine learning:

- 1950, Alan Turning created the Turing Test which is basically a test to determine if a machine can exhibit intelligent behavior similar to human beings. In the test, an interrogator (a human being) is given the task to determine which of two entities is a machine and which is a human, the machine passes the test if it succeeds in fooling the interrogator into thinking its the human being.

- 1957, Frank Rosenblatt invented the perceptron, the first neural network.

- 1959, Arthur Samuel came up with the term "Machine Learning", he was one of the pioneers in the field of artificial intelligence and best known for his work on computer checkers, which was a true demonstration for artificial intelligence.

- 1967, the nearest-neighbor algorithm was invented, it was used to solve the traveling salesman problem.

- 1970, Seppo Linnainmaa introduced reverse mode of automatic differentiation (AD), describing a general method for automatic differentiation of complex nested functions. That was used later for applying back propagation.

- 1996, IBM Deep Blue was the first computer to win against world champion (Garry Kasparov) in chess.
- 1998, Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner introduced LeNet-5, the first convolutional neural network, which propelled deep learning.
- 2006, Geoffrey Hinton coined the term ”Deep Learning”
- 2010, Dan Claudiu Ciresan and Jurgen Schmidhuber published one of the very first implementations of GPU neural networks
- 2012, AlexNet, a deeper convolutional neural network than LenNet-5 designed by Alex Krizhevsky won ImageNet Large Scale Visual Recognition Challenge (ILSVRC).
- 2014, Google introduced the first inception model, GoogLeNet, a 22 layers deep that won ILSVRC.
- 2015, ResNets were introduced, they helped in solving the vanishing gradient problem by using identity shortcut connections (forward the same value one or more layers).
- 2016, The first time a computer program defeats a professional player (Lee Sedol) in Go, which is considered a much harder game than chess.
- 2019 Lee Sedol announced his retirement arguing that he can no longer be the best player in Go due to the dominance of AI and he describes them as entities that cannot be defeated.

2.2 Artificial Neural Networks (ANN)

Artificial neural networks tried to simulate the human brain through the creation of neurons and connections among them. The initial model of brain interaction was introduced by Donald Hebb in 1949 in his book ”The Organization of Behavior. Hebb wrote ”When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell” [10]. The Hebbian theory led to one of the oldest learning algorithms, the Hebbian training, which can be formulated as below:

$$w_{ij}[n+1] = w_{ij}[n] + \eta x_i[n]x_j[n] \tag{2.1}$$

Where η is a learning rate, and $x_i[n]$ and $x_j[n]$ are, respectively, the outputs of the ith and jth elements at time step n.

The hebbian learning algorithm basically increases the weight between two nodes if they are in agreement.

Later back propagation became the dominant approach for learning artificial neural networks. Figure 2.1 shows how the neuron works by multiplying each input with a weight and then applies some activation function to the sum with a bias.

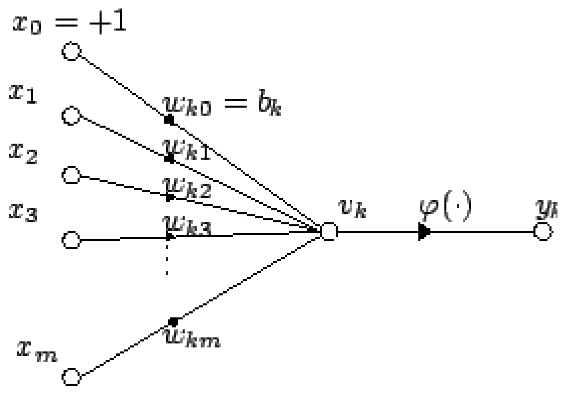


Figure 2.1: Artificial Neuron
Source: Wikipedia [11]

Where v_k is the weighted sum and φ is some activation function.

$$y_k = \varphi(\sum_{i=0}^{i=m} w_{ki}x_i) \tag{2.2}$$

2.3 Convolutional Neural Networks (CNN)

In 1959 two neuro-physiologists, David Hubel and Torsten Wiesel, published one of the most influential papers in the computer vision field. The paper, "Receptive fields of single neurons in the cat's striate cortex" [12], tried to observe neuronal activity in the cats brain based on what it sees. They placed electrodes in the primary visual cortex area of the brain of the cat and then they tried to observe any activity in the nerve cells while showing the cat different images (Figure 2.2). At first they could not make the nerve cells respond to anything, but after months of research while moving, one of the images a neuron got activated due to the movement of an edge when replacing an image with another. Hubel and Wiesel realized that neurons in the visual cortex respond to specific features of an image such as edges and angels.

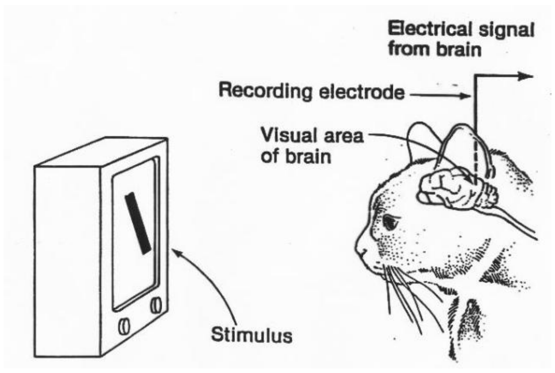


Figure 2.2: Hubel Experiment
Source: Good Psychology [13]

Later in 1980, a Japanese computer scientist, Kunihiko Fukushima, proposed a model inspired by the work of Hubel and Wiesel. The model, neocognitron [14], included multiple convolutional layers with filters that slide on 2D arrays and do some calculation, and the output was used as input for the next layers. The model was used for handwritten character recognition and it served as the inspiration for today's convolutional networks.

In 1998, a french scientist, Yann LeCun, introduced LeNet-5 [15] (Figure 2.3) and he applied back propagation to Fukushima's convolutional neural network architecture. LenNet-5 was the first modern convolutional network.

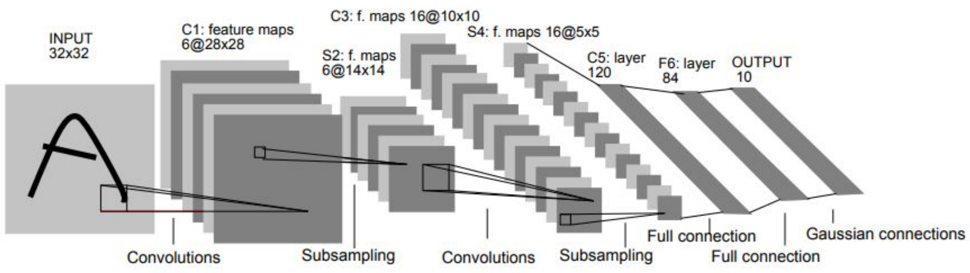


Figure 2.3: LeNet-5
Source: Original Paper [15]

Convolutional layers work as feature map detectors, it slides a filter, across the input tensor, and do an element-wise multiplication, then summation, and applies an optional activation function. Each filter will output one feature map, and a convolutional layer can consist of more than one filter.

Figure 2.4 shows a convolutional neural network filter being applied to an input tensor. And Figure 2.4 shows the element-wise multiplication.

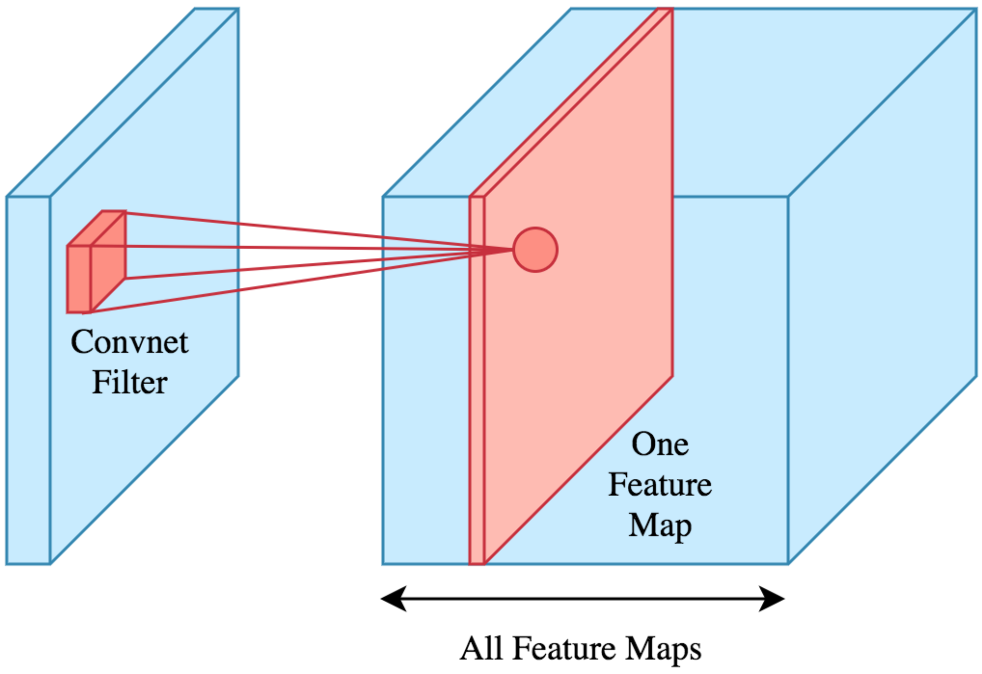


Figure 2.4: CNN Feature Maps

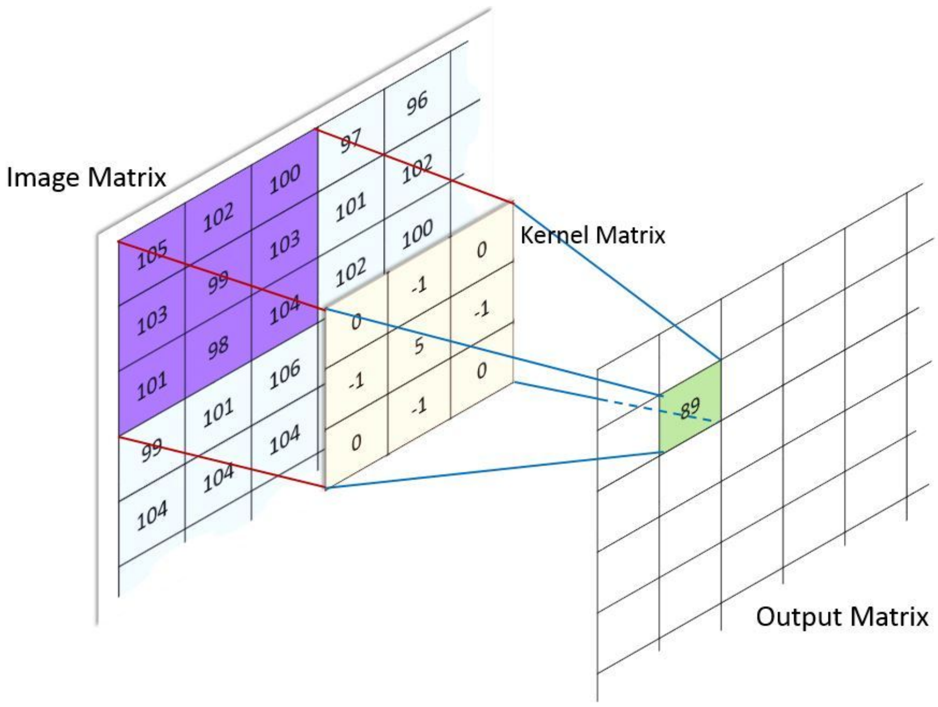


Figure 2.5: CNN Kernel Multiplication
Source: Brilliant [16]

2.4 K-Nearest Neighbors (KNN)

The KNN algorithm is a supervised machine learning algorithm that can be used for regression and classification problems.

In the context of a classification problem, the algorithm answers the question ”to which class a given input belongs given a set of labeled instances?”, the algorithm works by finding the nearest K neighbors, where K is predefined (e.g. 5) and the nearest neighbors are typically measured using euclidean distance. And then it returns the mode of the K neighbors. Figure 2.6 shows a simplified visualization of KNN.

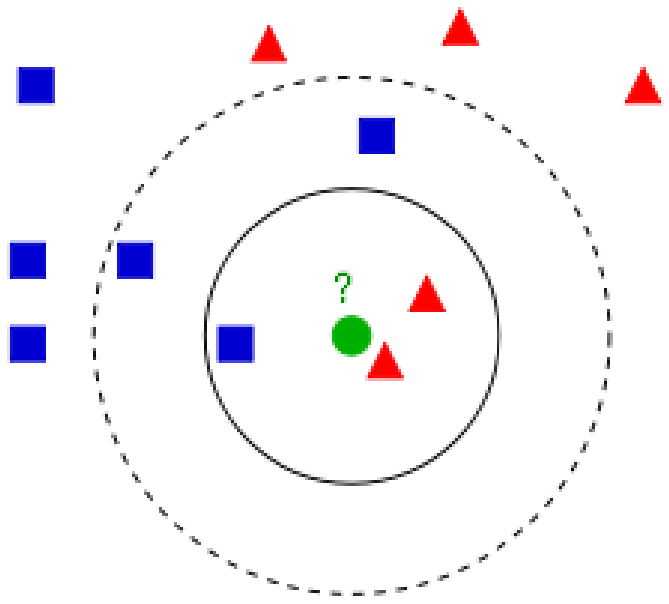


Figure 2.6: KNN Classification
Source: Wikipedia [17]

In the case of regression problem the algorithm simply returns the mean of the K neighbors.

2.5 Capsule Networks (CapsNet)

Neural networks have evolved through time form basic architectures to state-of-the-art networks like AlexNet, ResNets, and GoogLeNet. With the advancement in computational power and computer RAM, more complex networks that are deeper and able to process much more data became feasible to implement. However, these networks are variations of convolutional neural networks, some are deeper than others but all rely on the same building blocks, convolutional layers, dropouts, pooling, etc. . Capsule networks introduce new totally different building elements that tries to enhance the capability of neural networks in the domain of computer vision problems.

2.5.1 Convolutional Neural Networks Shortcomings

Lets assume we are creating a regular convolutional neural network and we train it to recognize the statue of liberty (Figure 2.7). To achieve that we would provide the neural network with thousands of images of the statue of liberty and train it on them. This is quite similar to how the human brain works. To make a person recognize something we provide him/her with images of that object. However, if we flip an image, the human brain still recognizes the object, but a convolutional neural network will fail the test unless it was trained on the flipped image too. Convolutional neural networks solve this issue through data augmentation, but this is basically creating variations of the images (flipping, zooming, cropping, etc.), and that does not sound like artificial intelligence, and that is not how the human brain works, and this makes the network require a lot of data.

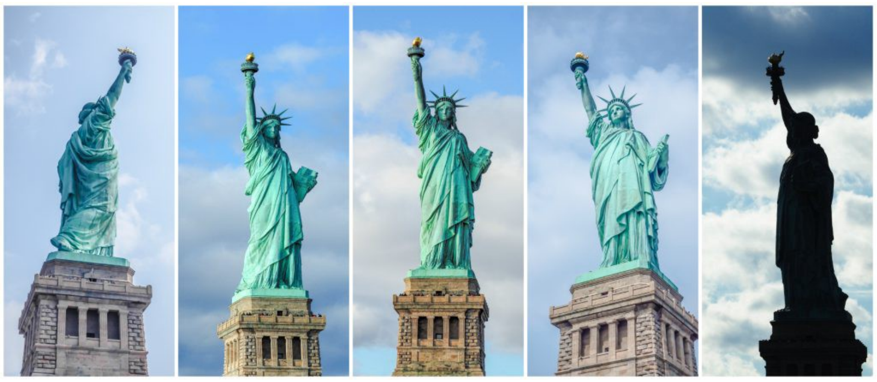


Figure 2.7: Statue of Liberty
Source: Medium [18]

Another problem with convolutional neural networks is pooling. Geoffrey Hinton says "The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster". Pooling operations, like max pooling, made convolutional neural networks achieve great results, but at the expense of losing valuable information.

Convolutional neural networks work by detecting low level features in the beginning and then uses those feature maps to detect higher level features in deeper layers. For example, a convolutional neural network may detect eyes, nose, and mouth in early CNN layers and then another layer detects a face. However, a convolutional neural network does not take into consideration the spatial relationship between those entities (Figure 2.8)

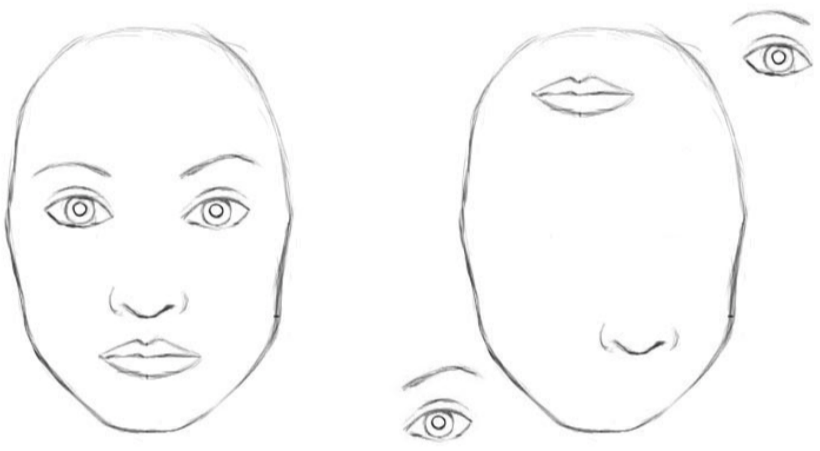


Figure 2.8: CNN Face Recognition
Source: Medium [18]

2.5.2 How Capsule Networks Work

Hinton says that the brain does the opposite of computer rendering which takes vectors of data and renders an object. Capsule networks aim to do the same thing as the brain, an inverse graphics operation. It constructs a part-whole hierarchy of parts where the instantiation parameters of a part are passed to the higher level part.

CapsNet achieves viewpoint invariance by learning the rotation and pose of the objects. In this way CapsNets require much less data for training, because we don't need to supply different variations of the same image.

CapsNets are equivariant, i.e. they learn objects that transform to each other, e.g. through rotation. Contrary to convolutional neural networks that are not rotation invariant. A CNN uses different neurons to detect variations of the same entity, which makes it bigger and require more data.

Figure 2.9 shows how a CNN uses different neurons to recognize the same face with different rotations.

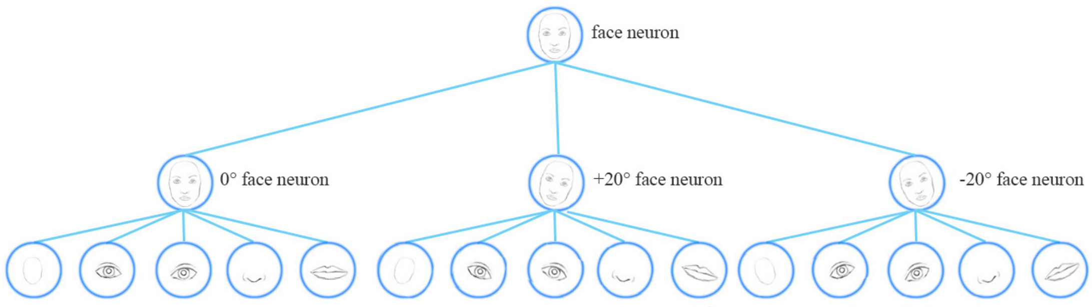
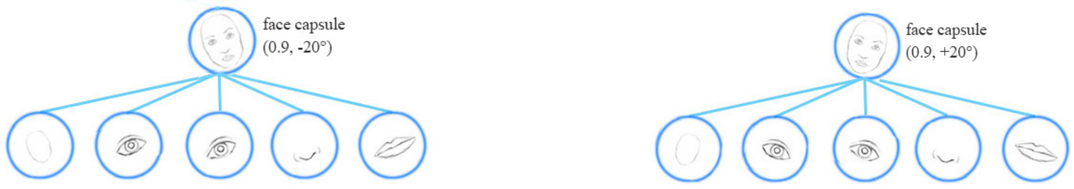


Figure 2.9: CNN Rotation
Source: GitHub [19]

A CapsNet encapsulates the instantiation parameters in the output vector to achieve equivariance. Figure 2.10 shows how one capsule is used to recognize the same face with different rotations.



(a) Left Rotation (b) Right Rotation (same capsule)

Figure 2.10: CapsNet Rotation
Source: GitHub [19]

The logic of CapsNet can be summarized as follows:

- Each capsule outputs a vector instead of a scalar
- The values in the vector denote different instantiation parameters of the object (pose, etc.)
- The length of the vector can be used to indicate the presence of the object, and we use a squashing algorithm to make sure the vector length in the range 0-1 (a probability)
- We multiply each input vector by a matrix and then do a weighted sum from the different lower capsule layers
- Finally, we use routing by agreement and increase the coupling coefficient if the result of the multiplication of the input vector and the weights matrix is in agreement with the output vector (using dot product)

Figure 2.11 shows how capsule networks work in comparison to neurons. The input vectors are multiplied by a weight matrix to obtain transformed vectors, $\hat{u}_{j|i}$, the result is multiplied by coupling coefficients, c_i, and then summed. The output vector, u_j, of capsule j is squashed to make sure its length in the range [0-1].

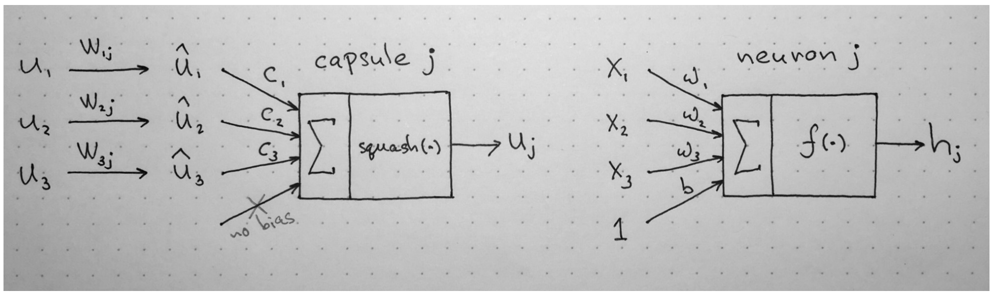


Figure 2.11: Capsule
Source: Online [20]

2.5.3 Squashing Algorithm

The length of the output vector of a capsule represents the probability of existence of some entity, thus it must be in the range 0-1. A squashing algorithm is used for this purpose as follows:

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|} \frac{s_j}{\|s_j\|} \tag{2.3}$$

Where v_j is the vector output of capsule j. s_j is the weighted sum over all "prediction vectors" $\hat{u}_{j|i}$ from the lower-level capsules:

$$s_j = \sum_i c_{ij} \hat{u}_{j|i} \tag{2.4}$$

$$\hat{u}_{j|i} = W_{ij} u_i \tag{2.5}$$

Where c_{ij} are the coupling coefficients that are determined by iterative dynamic routing by agreement.
u_i is the vector output of the lower level capsule.

2.5.4 Dynamic Routing By Agreement

Capsule networks use a dynamic routing by agreement to construct the part-whole relationship. Basically there is a coupling coefficient between a capsule and its next higher level capsule, this coupling coefficient increases when the two capsules are in agreement. Two capsules are considered in agreement if their output vectors have the same direction. The dot product of the two vectors is used to determine this agreement.

The sum of all coupling coefficients belonging to a low level capsule should be equal to one. And all the coupling coefficients are calculated using an iterative approach as shown in Algorithm 1.

Algorithm 1 Routing algorithm

```
1: procedure ROUTING($\hat{u}_{j|i}, r, l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $c_i \leftarrow softmax(b_i)$
5:         for all capsule $j$ in layer $l+1$: $s_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$
6:         for all capsule $j$ in layer $l+1$: $v_j \leftarrow squash(s_j)$
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $l+1$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i}.v_j$
8:     return $v_j$
```

2.6 Synthetic Minority Over-sampling Technique (SMOTE)

One of the major problems that might be faced when training a machine learning model is when one of the classes has much more instances than the others. This case is referred to as a class imbalance.

Imagine we have two classes with 80% of the instances belonging to one class. In this case a model which always predicts the majority class would have an 80% accuracy.

To solve the class imbalance problem there are 3 techniques:

1. Under-Sampling: this is done by removing instances from the majority class
2. Oversampling: this is done by adding additional instances to the minority class
3. Modifying the loss function to give a higher importance for the misclassification in the minority class

SMOTE is an oversampling technique that synthesis new instances given the existing ones. Basically what it does is that it loops through the exiting real instances and for each one it determines the k-nearest neighbors and then randomly picks one of those neighbors and then randomly returns a new instance on the line joining the original instance and the selected neighbor. Figure 2.12 shows a simplified visualization of SMOTE in 2 dimensions.

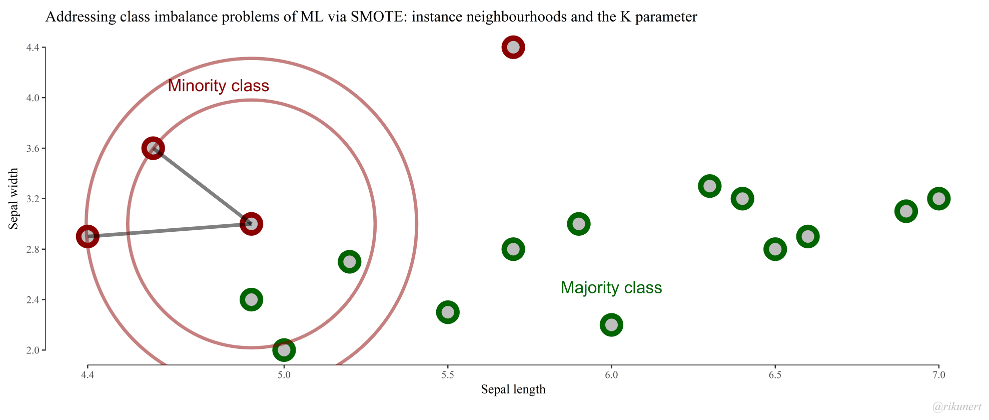


Figure 2.12: SMOTE
Source: Rich Data [21]

2.7 Contrast Limited Adaptive Histogram Equalization (CLAHE)

An image histogram is a representation of distribution of the intensities of values per channel. The x-axis represent the range of values of the intensities, from 0 to 255, and the y-axis represents the counts of each intensity. Figure 2.13 shows a sample histogram for each channel, at the x-axis are the intensities, and at the y-axis is the counts of pixels.

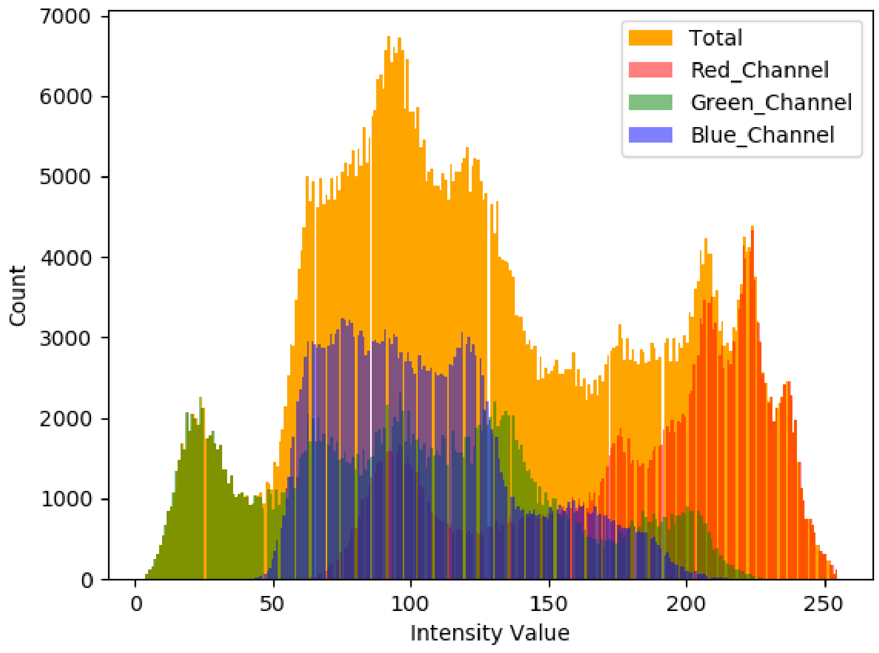


Figure 2.13: Image Histogram
Source: Towards Data Science [22]

Histogram equalization is a technique used to enhance the contrast of the image by spreading the intensities in the histogram (Figure 2.14):

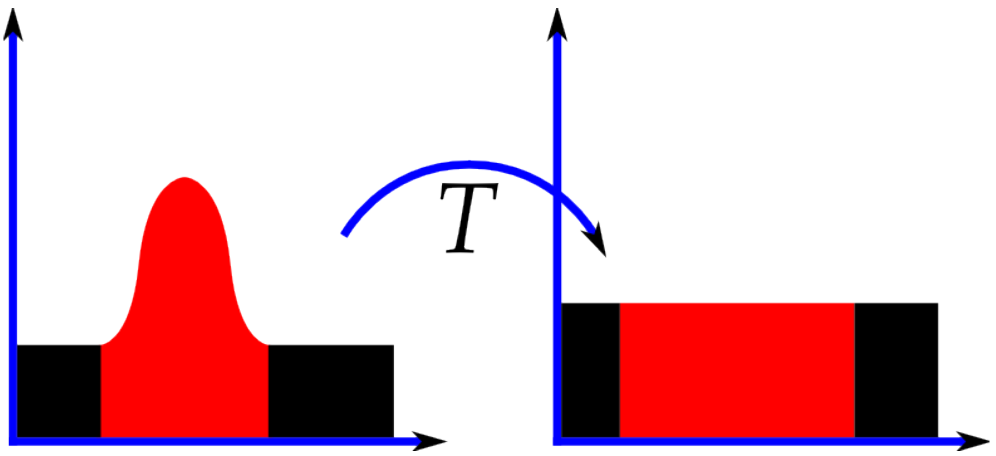


Figure 2.14: Histogram Equalization
Source: Wikipedia [23]

Histogram equalization usually enhances the global contrast of the image. To solve that the adaptive histogram equalization uses multiple histograms for different sections of the image. Figure 2.15 shows a sample picture with histogram equalization and with adaptive histogram equalization.

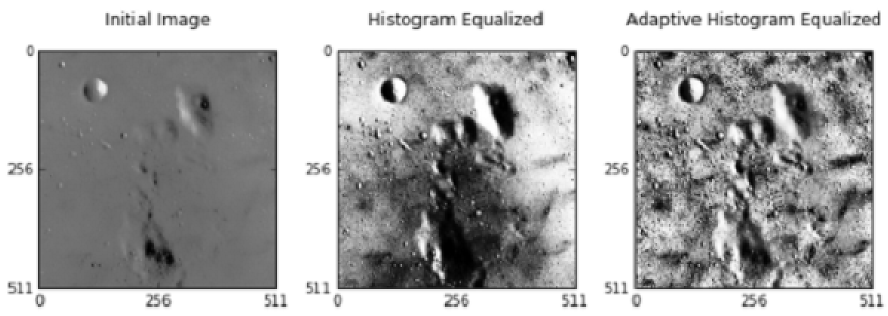


Figure 2.15: Adaptive Histogram Equalization
Source: Towards Data Science [24]

AHE (adaptive histogram equalization) has a tendency to amplify noise in relatively same regions of the image, contrast limited adaptive histogram equalization (CLAHE) tries to solve this by limiting the contrast amplification. Figure 2.16 compares a sample picture using the different techniques.

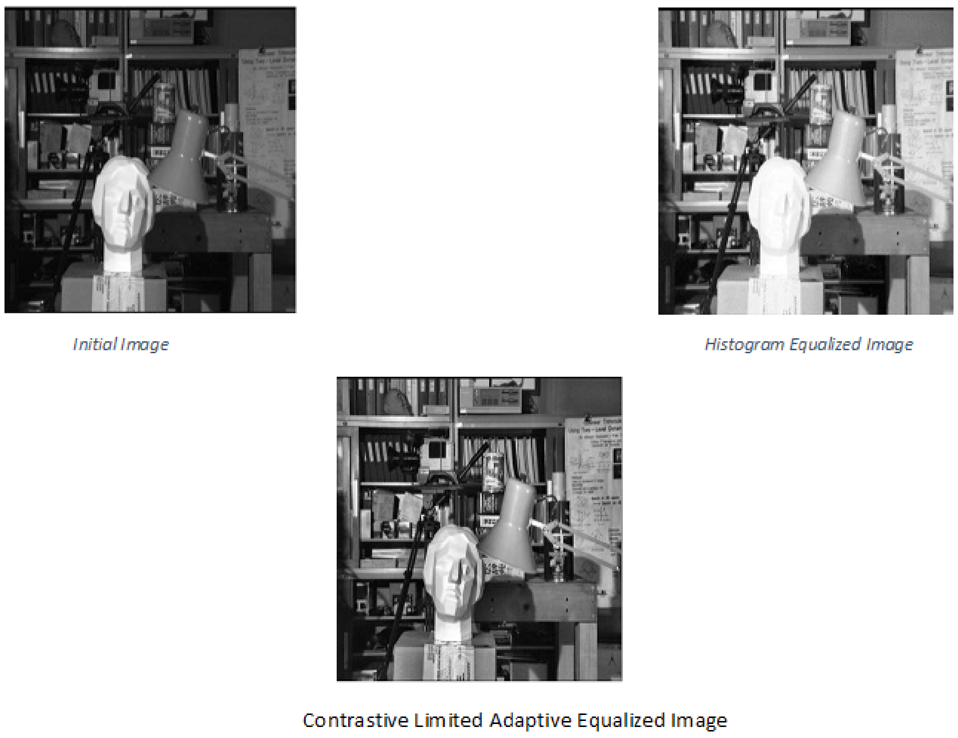

Initial Image
Histogram Equalized Image
Contrastive Limited Adaptive Equalized Image

Figure 2.16: Contrast Limited Adaptive Histogram Equalization
Source: Towards Data Science [24]

Chapter 3

Literature Review and Related Work

3.1 Dynamic Routing Between Capsules[4]

In this paper they introduce capsule networks and try to apply it to the MNIST dataset for handwritten digits recognition.

They show that a discriminatively trained capsule network achieves state-of-the-art performance on MNIST and is considerably better than a convolutional network at recognizing highly overlapping digits.

They used a model consisting of a convolution layer (9x9) and then two capsule layers (Figure 3.1):

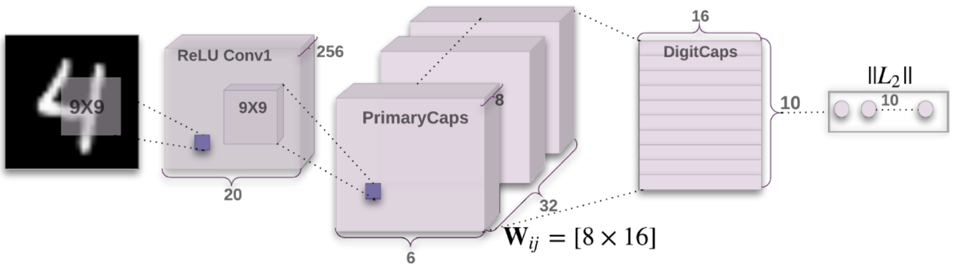


Figure 3.1: DigitsCap Model

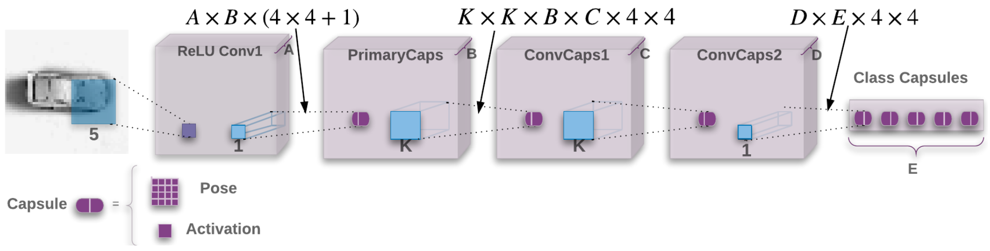
3.2 Matrix Capsules With EM Routing[5]

In this paper the authors describe a version of capsules in which each capsule has a logistic unit to represent the presence of an entity and a 4x4 matrix which could learn the relationship between the entity and the viewer (the pose).

They applied their approach on the smallNORB dataset, and they were able to reduce the number of test errors by 45% compared to the state-of-the-art.

The smallNORB dataset is intended for 3d objects recognition. It consists of around 50,000 images of 50 toys belonging to 5 generic categories: airplanes, trucks, cars, four-legged animals, and human figures. Each image is 96x96 pixels.

They used a 5x5 convolutional layer with 32 filters and ReLu activation function followed a primary capsule layer and then two other capsule layers and a final class capsule layer consisting of 5 capsules (Figure 3.2).



$A \times B \times (4 \times 4 + 1)$
$K \times K \times B \times C \times 4 \times 4$
$D \times E \times 4 \times 4$
ReLU Conv1
A
PrimaryCaps
B
ConvCaps1
C
ConvCaps2
D
Class Capsules
5
1
K
K
1
E
Capsule
=
Pose
Activation

Figure 3.2: SmallNORB Model

Where A is the number of filters in the convolution layer = 32. B is the number of primary capsule types =32, K is the kernel size =3, C is the number of capsule types in the first convolutional capsule layer =32, D is the number of capsule types in the first convolutional capsule layer =32. The last capsule layer consists of one capsule per class output (E=5)

3.3 ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases [3]

In this paper Xiaosong Wang and other team members from the department of radiology and imaging sciences at the national institutes of health, Bethesda, tries to take advantage of the tremendous number of x-ray images that are stored in their institute's Picture Archiving and Communication System (PACS) to train a neural network that is able to recognize certain disease types. These images are not labeled, however they are associated with radiological reports. Through natural language processing techniques applied to the radiological reports they were able to label each of the X-ray images. And then they used this newly presented dataset to build a computer-aided diagnosis (CAD) system using a convolutional neural network.

The dataset they built initially consisted of 108,948 frontal-view X-ray images belonging to 32,717 patients and each image was labeled with zero or more thoracic pathology: Atelectasis, Cardiomegaly, Effusion, Infiltration, Mass, Nodule, Pneumonia, and Pneumathorax.

Later, the dataset was expanded to include 6 additional thorax diseases(total 112,120 images): Consolidation, Edema, Emphysema, Fibrosis, Pleural Thickening, and Hernia.

For the model they tried pre-trained ImageNet models (AlexNet, GoogLeNet, VGGNet-16, and ResNet-50), ResNet-50 was found to give the best results. Figure 3.3 shows the ROC curve.

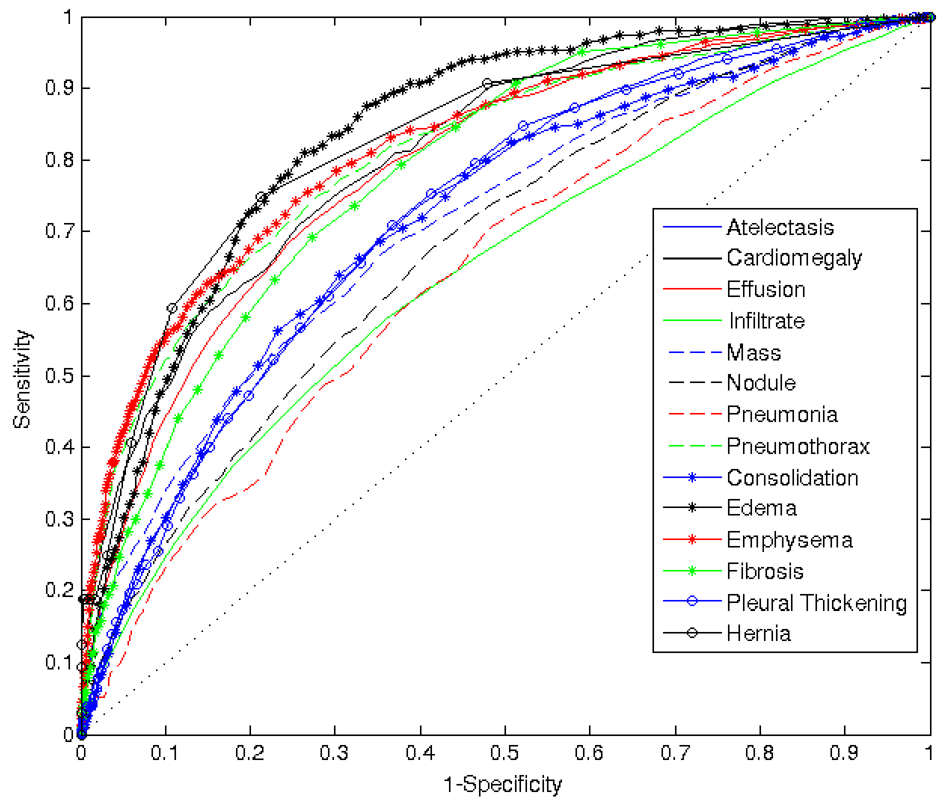


Figure 3.3: ChestX-ray14 ROC

Table 3.1 shows the results they achieved (AUROC):

ResNet-50	ChestX-ray8	ChestX-ray14
Atelectasis	0.7069	0.7003
Cardiomegaly	0.8141	0.8100
Effusion	0.7362	0.7585
Infiltration	0.6128	0.6614
Mass	0.5609	0.6933
Nodule	0.7164	0.6687
Pneumonia	0.6333	0.6580
Pneumothorax	0.7891	0.7993
Consolidation	-	0.7032
Edema	-	0.8052
Emphysema	-	0.8330
Fibrosis	-	0.7859
Pleural Thickening	-	0.6835
Hernia	-	0.8717

Table 3.1: Wang et al. Results

3.4 Learning to diagnose from scratch by exploiting dependencies among labels [6]

In this paper they try to build on the dataset ”ChestX-ray14” that was introduced by Wang et al.(2017) [3] and come up with better results using a model developed from scratch (Figure 3.4) instead of transfer learning of pre-trained ImageNet models.

Two key points in their approach:

1. Using DenseNet as opposed to ResNet that was used by Wang et al.(2017) [3]
2. Taking advantage of the inter-dependencies between labels for making better predictions using LSTM (Long short-term memory)

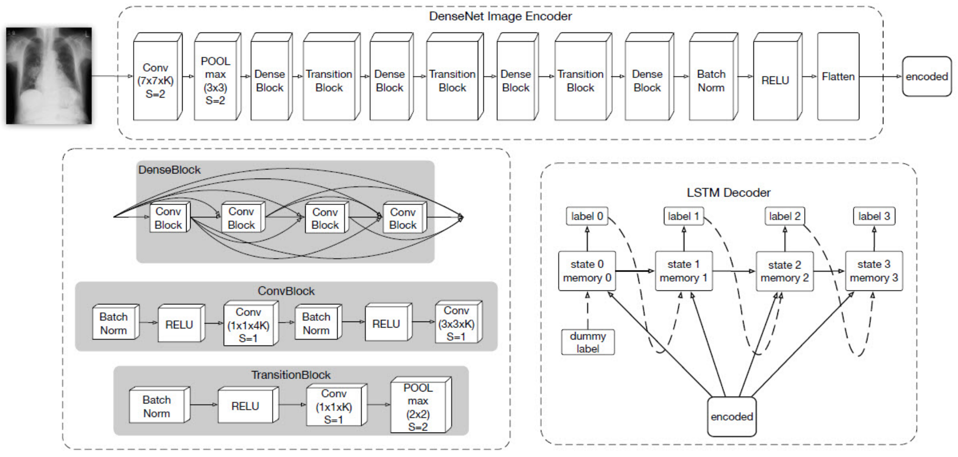


Figure 3.4: LSTM Model

Table 3.2 shows the results they achieved (AUROC):

Pathology	AUROC
Atelectasis	0.772
Cardiomegaly	0.904
Effusion	0.859
Infiltration	0.695
Mass	0.792
Nodule	0.717
Pneumonia	0.713
Pneumothorax	0.841
Consolidation	0.788
Edema	0.882
Emphysema	0.829
Fibrosis	0.767
Pleural Thickening	0.765
Hernia	0.914

Table 3.2: Yao et al. Results

3.5 CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning [7]

In this paper they aim to develop and algorithm that can detect pneumonia from chest X-ray images at a level exceeding practicing radiologists.

To achieve that they used a model, ChestXNet, consisting of 121-layer convolutional neural network trained on ChestX-ray14 [3].

Table 3.3 shows the results they achieved (AUROC):

Pathology	AUROC
Atelectasis	0.8094
Cardiomegaly	0.9248
Effusion	0.8638
Infiltration	0.7345
Mass	0.8676
Nodule	0.7802
Pneumonia	0.7680
Pneumothorax	0.8887
Consolidation	0.7901
Edema	0.8878
Emphysema	0.9371
Fibrosis	0.8047
Pleural Thickening	0.8062
Hernia	0.9164

Table 3.3: ChestXNet Results

Chapter 4

Pneumonia Detection

In this chapter we will try to tackle the problem of pneumonia detection using a binary classifier. Our dataset [2] which we will refer to as the pneumonia dataset is a publicly available dataset on Kaggle, many attempts were made to use machine learning to obtain a model that is as accurate as possible. We will try to utilize the concepts of capsule networks and see how it performs in comparison to the other attempts using regular convolutional neural networks.

4.1 Dataset

The pneumonia dataset [2] we use consists of 5,855 images belonging to people that were diagnosed with pneumonia and others with no pneumonia detected. Its a binary classification problem. Figure 4.1 shows a bar graph of the counts of the two classes we have, pneumonia with 4272 images and no pneumonia with 1583 images.

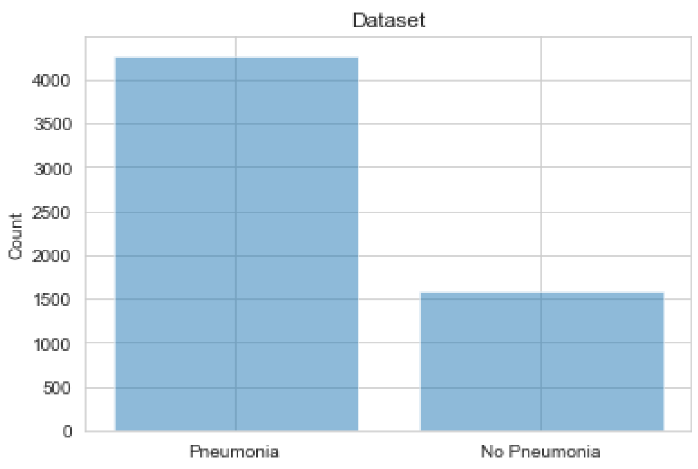


Figure 4.1: Pneumonia Dataset Counts

Figure 4.2 shows the distribution of the the two classes in terms of percentage of the total count.

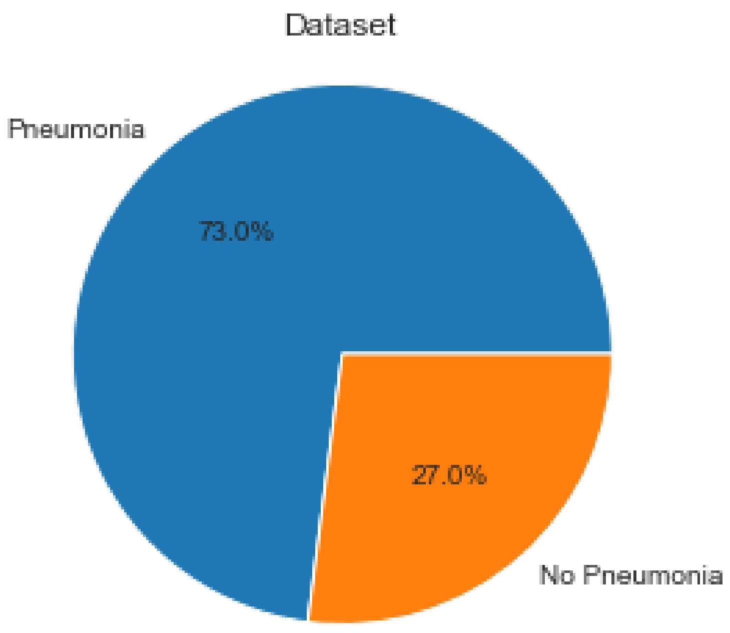


Figure 4.2: Pneumonia Dataset Percentages

4.2 Images Preprocessing

Before starting to work on the model, we had to apply pre-processing to the images. Below are the steps applied:

1. The images were resized to 128x128 pixels

2. We applied Contrast limited adaptive histogram equalization (CLAHE) to the images

3. The data was split as shown in table 4.1 and figure 4.3:

Training	4,184(71.46%),
Validation	624 (10.66%)
Testing	1,047 (17.88%)

Table 4.1: Pneumonia Dataset Split

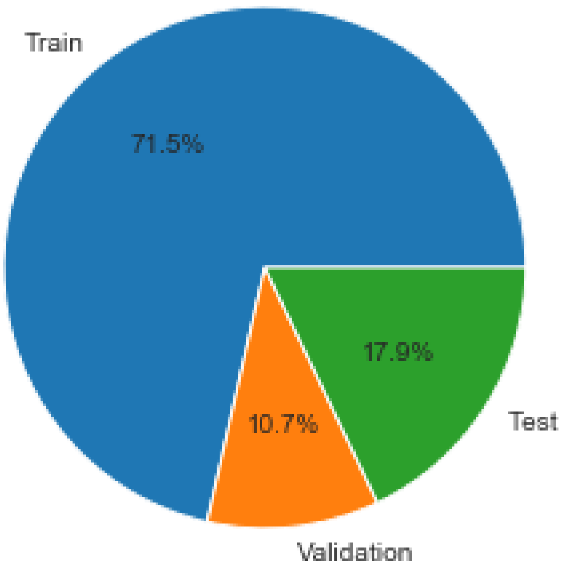


Figure 4.3: Pneumonia Dataset Split

Figure 4.4 shows some randomly selected images with pneumonia after pre-processing:

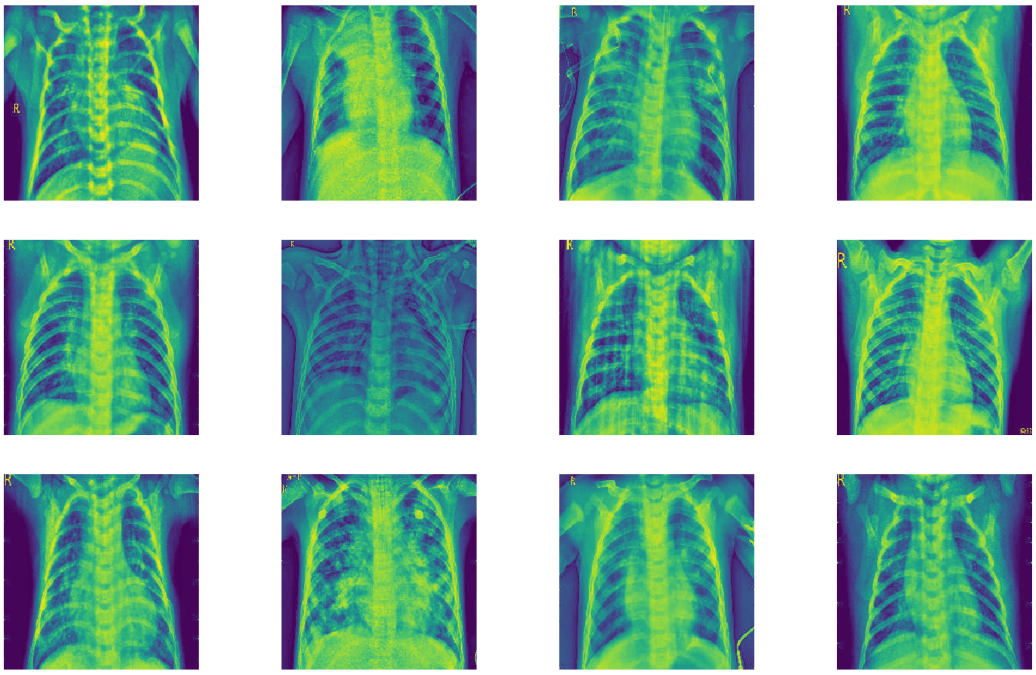


Figure 4.4: Pneumonia Dataset Sample

Figure 4.5 shows some randomly selected images without pneumonia after pre-processing:

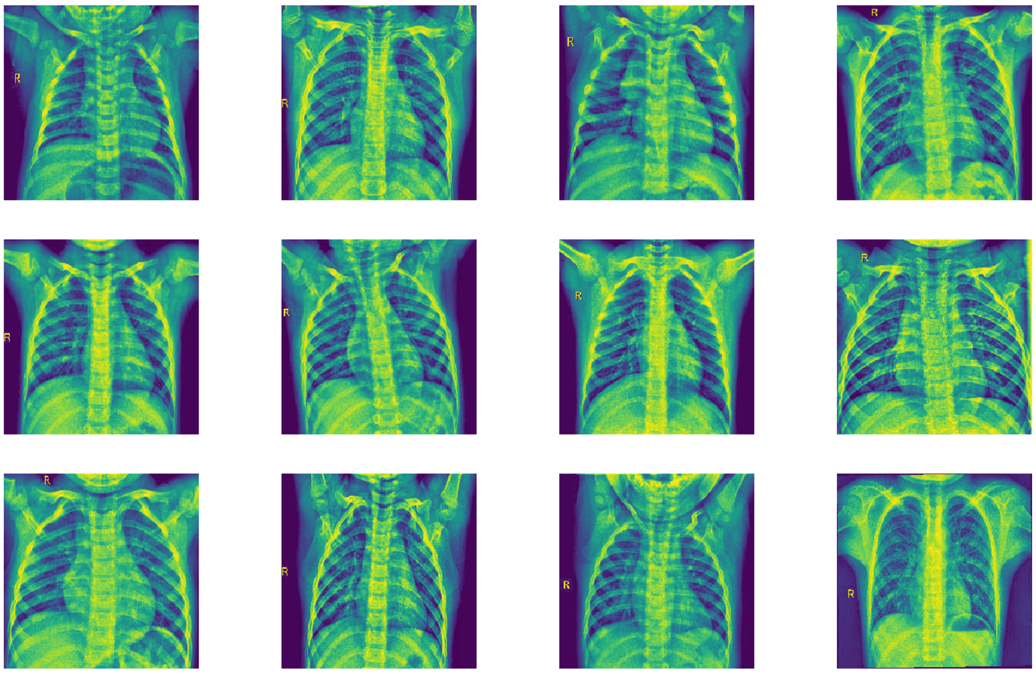


Figure 4.5: Pneumonia Dataset Sample

4.3 The Capsule Network

The capsule network we used can be described as below:

- Two convolutional layers followed by two capsule layers, with a decoder.
- The two convolutional layers are regular CNN layers that do feature map extraction.
- The first capsule layer, takes output from the last convolutional layer and produces vectors (squashed with 6 dimensions).
- The last capsule layer outputs a single vector whose length represents the probability of detecting pneumonia, and the values of the vector represent different instantiation parameters (8 dimensions).
- The decoder network consists of dense layers, it takes the output vector of the last capsule layer and reproduces the image.
- We used Adam optimizer, binary-accuracy, and margin loss. [4]

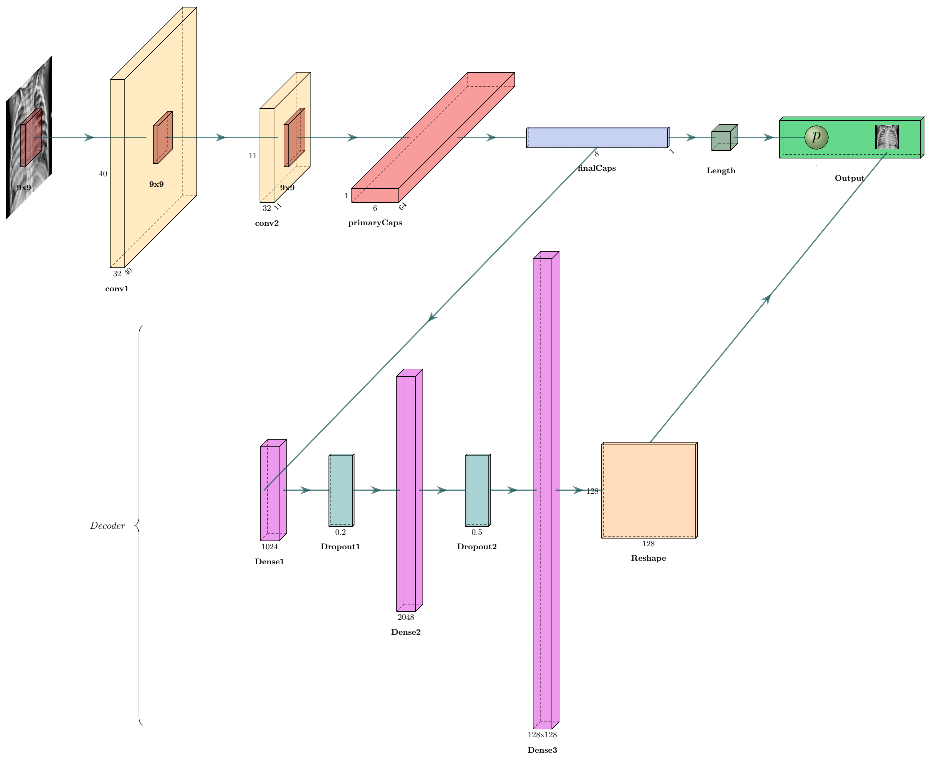
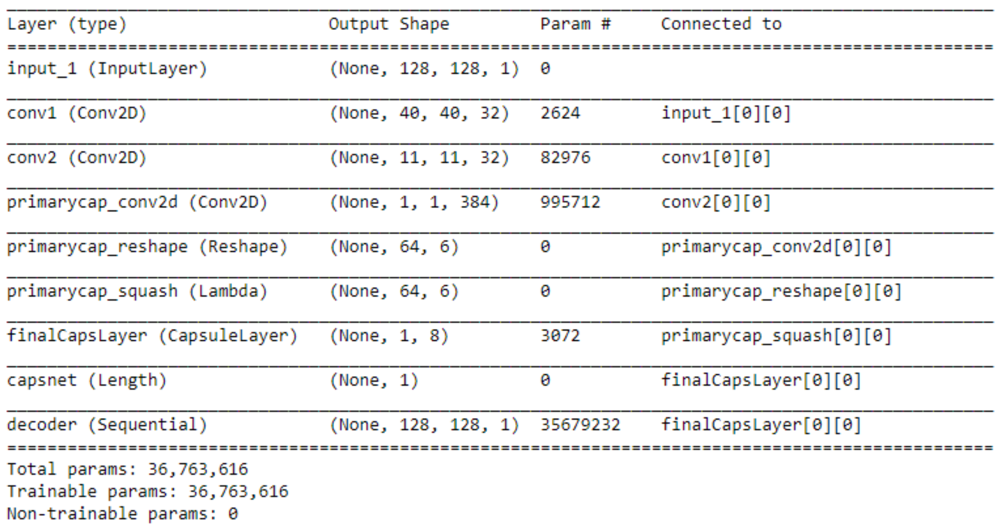


Figure 4.6: Pneumonia Capsule Network



```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_1 (InputLayer)            (None, 128, 128, 1)  0
________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 40, 40, 32)   2624        input_1[0][0]
________________________________________________________________________________________________
conv2 (Conv2D)                  (None, 11, 11, 32)   82976       conv1[0][0]
________________________________________________________________________________________________
primarycap_conv2d (Conv2D)      (None, 1, 1, 384)    995712      conv2[0][0]
________________________________________________________________________________________________
primarycap_reshape (Reshape)    (None, 64, 6)        0           primarycap_conv2d[0][0]
________________________________________________________________________________________________
primarycap_squash (Lambda)      (None, 64, 6)        0           primarycap_reshape[0][0]
________________________________________________________________________________________________
finalCapsLayer (CapsuleLayer)   (None, 1, 8)         3072        primarycap_squash[0][0]
________________________________________________________________________________________________
capsnet (Length)                (None, 1)            0           finalCapsLayer[0][0]
________________________________________________________________________________________________
decoder (Sequential)            (None, 128, 128, 1)  35679232    finalCapsLayer[0][0]
================================================================================================
Total params: 36,763,616
Trainable params: 36,763,616
Non-trainable params: 0
```

Figure 4.7: Pneumonia Capsule Network Summary

4.4 Margin Loss

We used a margin loss function, which basically works in a similar manner to the sum of squared error but it returns zero if the error is within a specific margin:

$$L = T\max(0, m^{+} - \|v\|)^{2} + \lambda(1 - T)\max(0, \|v\| - m^{-})^{2} \tag{4.1}$$

T is the true label associated with the input image (1 or 0)
m^{-} and m^{+} are the margins, we used 0.1 and 0.9 respectively
λ is a constant,we used 1.0
$\|v\|$ is the length of the last capsule output vector.

4.5 The Decoder

Given the idea that a capsule outputs a vector that represents the instantiation parameters of some entity, then we can in a way or another reconstruct the image given that vector. For example, if we are building a capsule network to recognize handwritten digits, and a capsule outputs a vector that represents the instantiation parameters of an image of the digit 8 for example, then we should be able to reconstruct an image of that digit given those parameters, which could yield an image with longer, bolder, or other varied version of the digit 8.

The below describes the decoder we used with the X-ray images:

- It consists of a sequence of dense and dropout layers, and the last layer is an activation layer that just reshapes the output from a vector to a 128x128 matrix, which is the input image size.

- During training its used as a loss function that helps in regularizing the model.
- We reconstruct the image from the output vector of the last capsule layer, and then compute the loss compared to the original input image. We used sum of squared error.
- To make sure that the loss in the decoder does not dominate the loss of the model we multiply it with some weight, we used 0.0002, this is a hyper parameter that was tweaked as part of the model training.

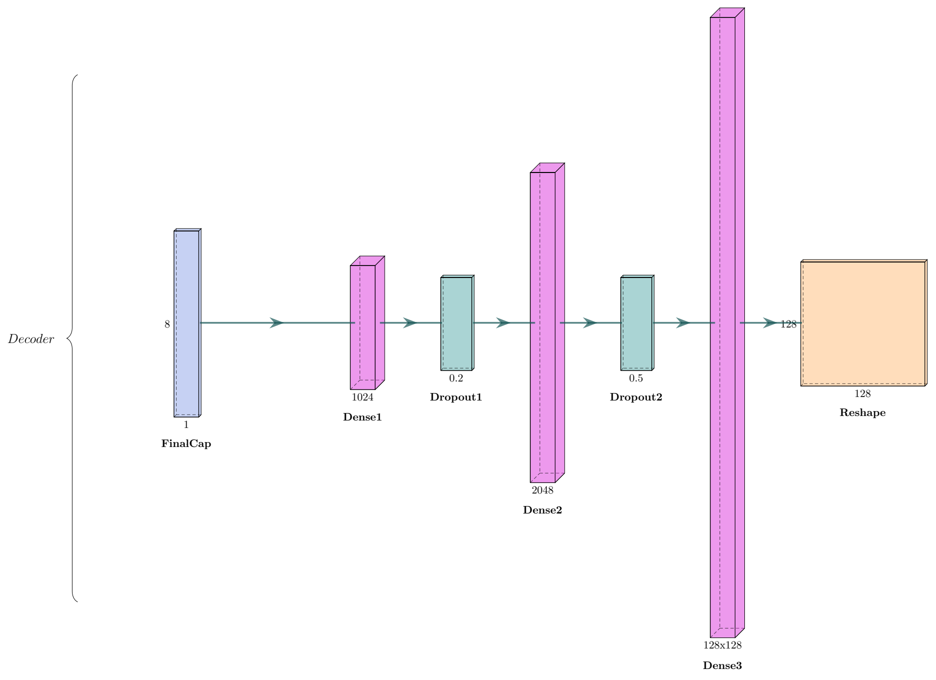


Figure 4.8: Capsule Network Decoder

To demonstrate how an image can be reconstructed from a vector, we took the output vector (8 dimensions) and we altered the values of each dimension by the values [-0.7, -0.5, 0, 0.5, 0.7] and plotted the reconstructed image.

Figure 4.9 shows a sample image and figure 4.10 shows the reconstructed images using the decoder of that image. There are 8 rows in the reconstructed images because our output vector is of 8 dimensions. As can be seen, by altering the instantiation parameters a modified version of the image is being reconstructed.

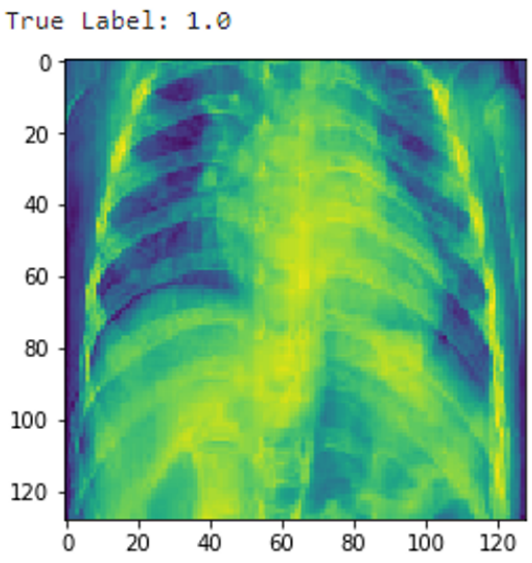


Figure 4.9: Decoder Input Image

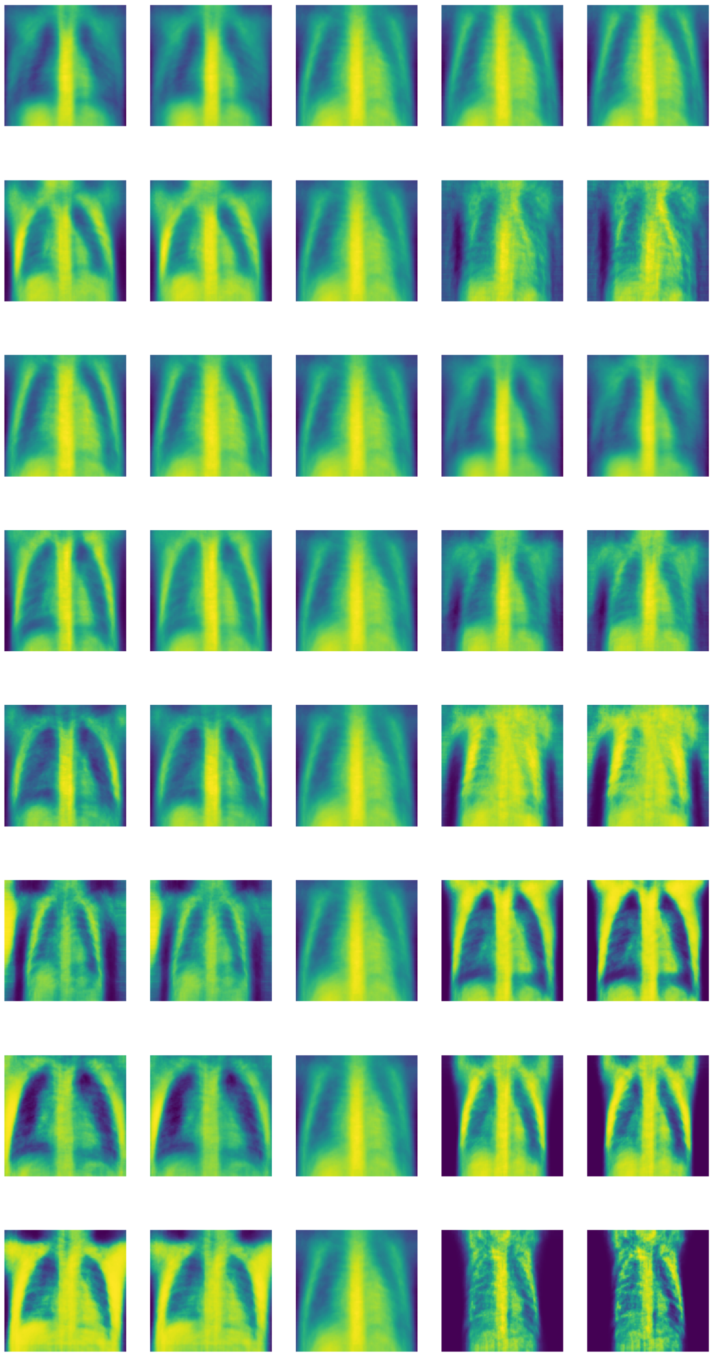


Figure 4.10: Decoder Reconstructed Images

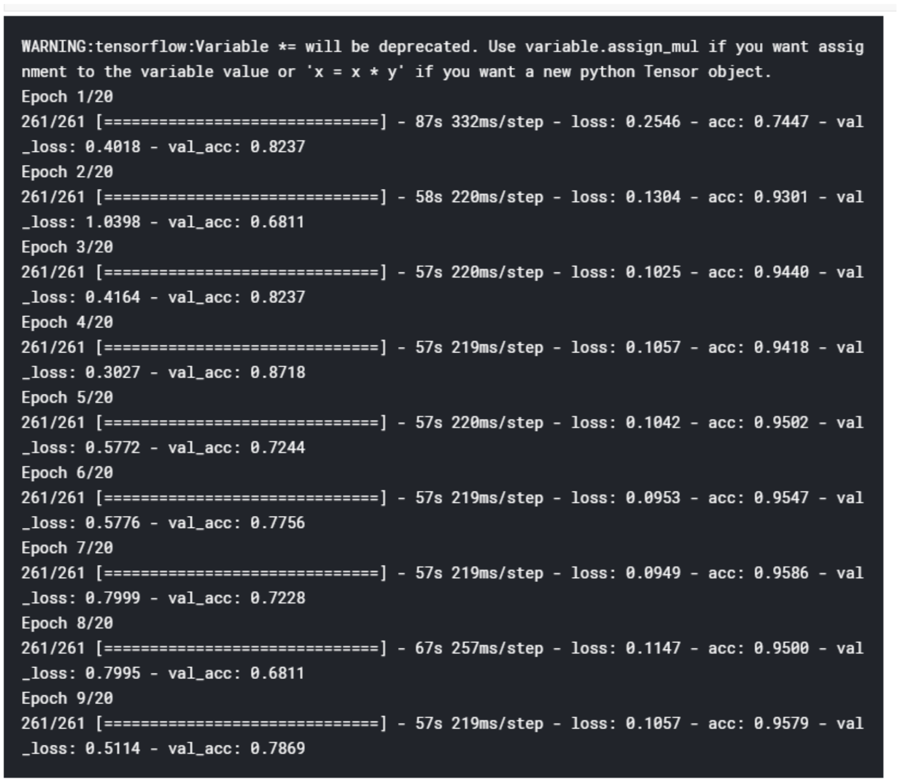
4.6 Results

To see how good our capsule network is performing we took the most voted kernel on Kaggle [9] that works on the same dataset and we compared the results.

4.6.1 CNN Results

The CNN model [9] was run on our split data and the following results were recorded on the test dataset. Figure 4.11 shows the epochs while training the CNN model [9].

1. Accuracy: 0.9054
2. Recall: 0.8756
3. Precision: 0.9971
4. F1-score: 0.9324



```
WARNING:tensorflow:Variable *= will be deprecated. Use variable.assign_mul if you want assig
nment to the variable value or 'x = x * y' if you want a new python Tensor object.
Epoch 1/20
261/261 [==============================] - 87s 332ms/step - loss: 0.2546 - acc: 0.7447 - val
_loss: 0.4018 - val_acc: 0.8237
Epoch 2/20
261/261 [==============================] - 58s 220ms/step - loss: 0.1304 - acc: 0.9301 - val
_loss: 1.0398 - val_acc: 0.6811
Epoch 3/20
261/261 [==============================] - 57s 220ms/step - loss: 0.1025 - acc: 0.9440 - val
_loss: 0.4164 - val_acc: 0.8237
Epoch 4/20
261/261 [==============================] - 57s 219ms/step - loss: 0.1057 - acc: 0.9418 - val
_loss: 0.3027 - val_acc: 0.8718
Epoch 5/20
261/261 [==============================] - 57s 220ms/step - loss: 0.1042 - acc: 0.9502 - val
_loss: 0.5772 - val_acc: 0.7244
Epoch 6/20
261/261 [==============================] - 57s 219ms/step - loss: 0.0953 - acc: 0.9547 - val
_loss: 0.5776 - val_acc: 0.7756
Epoch 7/20
261/261 [==============================] - 57s 219ms/step - loss: 0.0949 - acc: 0.9586 - val
_loss: 0.7999 - val_acc: 0.7228
Epoch 8/20
261/261 [==============================] - 67s 257ms/step - loss: 0.1147 - acc: 0.9500 - val
_loss: 0.7995 - val_acc: 0.6811
Epoch 9/20
261/261 [==============================] - 57s 219ms/step - loss: 0.1057 - acc: 0.9579 - val
_loss: 0.5114 - val_acc: 0.7869
```

Figure 4.11: CNN Epochs

Figure 4.12 shows the confusion matrix.

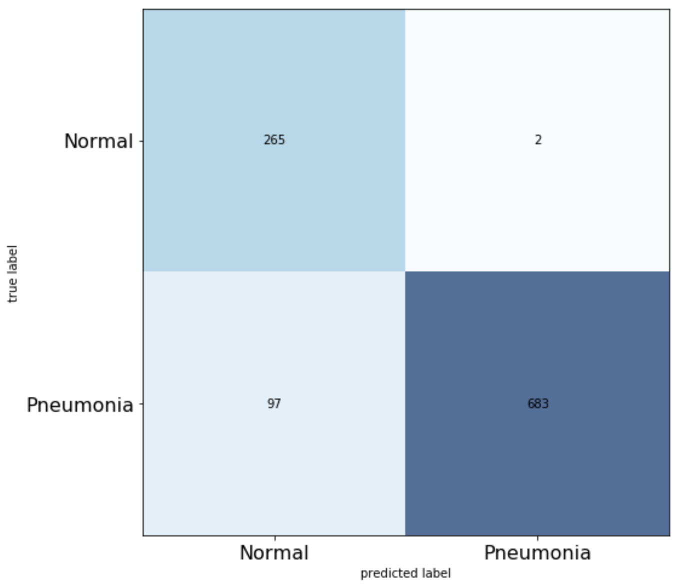


Figure 4.12: CNN Confusion Matrix

4.6.2 CapsNet Results (Our Results)

Using capsule networks we were able to achieve much better results than CNN in terms of accuracy and recall. Recall is very important in our case study and in the medical domain in general (higher recall means lower false negatives). Table 4.2 shows the recorded statistics using our capsule network on the test dataset in comparison with the results of CNN:

	CNN	CapsNet
Accuracy	0.9054	0.9475
Recall	0.8756	0.9385
Precision	0.9971	0.9905
F1-score	0.9324	0.9638

Table 4.2: CapsNet Results

Figure 4.13 shows the confusion matrix associated with the capsule network we used.

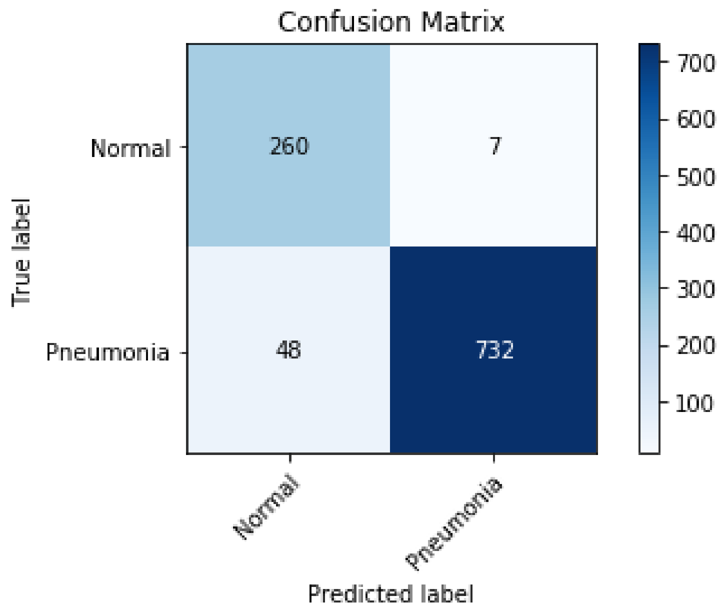


Figure 4.13: Pneumonia Confusion Matrix

Figure 4.14 shows the learning curve of the model, the loss is a weighted sum of the capsnet loss (capsnet outputs a probability, the length of the output vector) and the decoder loss which is the sum of squared error between the input image and reconstructed one. The decoder acts as a regularizer.

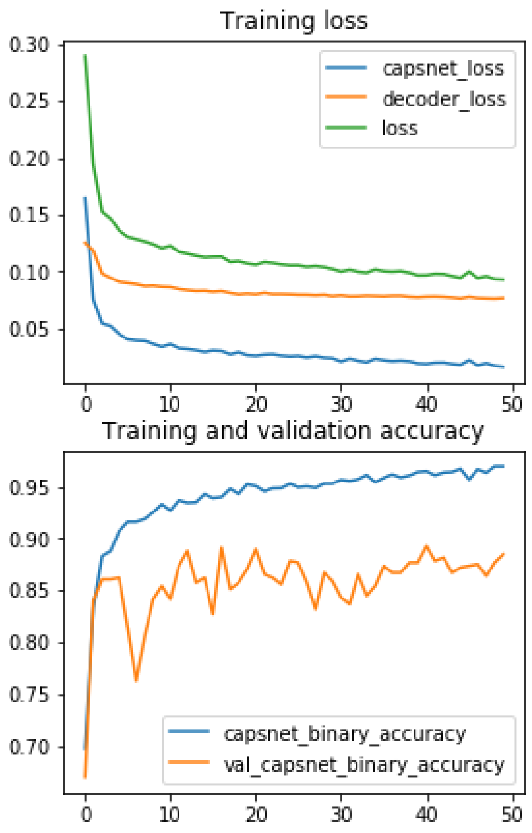


Figure 4.14: Pneumonia Learning Curve

Chapter 5

Multi-Pathology Detection

Another case we tackle in this thesis is using capsule networks with a bigger dataset and a more complex classification problem. We use the largest publicly available X-ray dataset, the national institutes of health (NIH) chest X-ray dataset [3], and the problem we tackle is a multi-label classification problem.

5.1 Dataset

The dataset [3] consists of 112,120 X-ray images belonging to 30,805 unique patients, and each image can be labeled with zero or more of the below 14 classes:

1. Atelectasis
2. Cardiomegaly
3. Effusion
4. Infiltration
5. Mass
6. Nodule
7. Pneumonia
8. Pneumothorax
9. Consolidation
10. Edema
11. Emphysema
12. Fibrosis

13. Pleural Thickening

14. Hernia

The dataset was constructed using text mining techniques and natural language processing from radiological reports. Its suitable for weakly-supervised learning, and the accuracy of the associated labels is expected to be greater than 90%.

Figure 5.1 and table 5.1 shows the total images count per pathology type, clearly we have a class imbalance issue that we need to tackle:

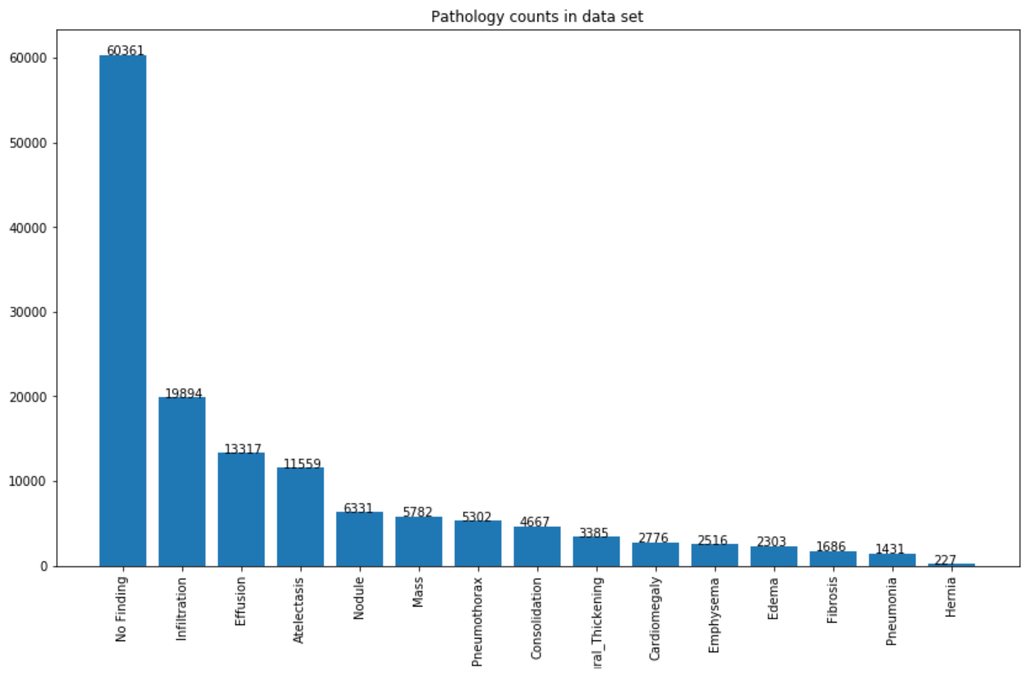


Figure 5.1: Pathology Counts

Pathology	Count
No Finding	60,361
Infiltration	19,894
Effusion	13,317
Atelectasis	11,559
Nodule	6,331
Mass	5,782
Pneumothorax	5,302
Consolidation	4,667
Pleural Thickening	3,385
Cardiomegaly	2,776
Emphysema	2,516
Edema	2,303
Fibrosis	1,686
Pneumonia	1,431
Hernia	227

Table 5.1: Pathology Counts

Figure 5.2 and table 5.2 shows the percentages of the positive cases out of the total number of images. As can be seen less than 2% of the total images are labeled with pneumonia, a dummy model that would always predict no pneumonia will achieve more than 98% accuracy.

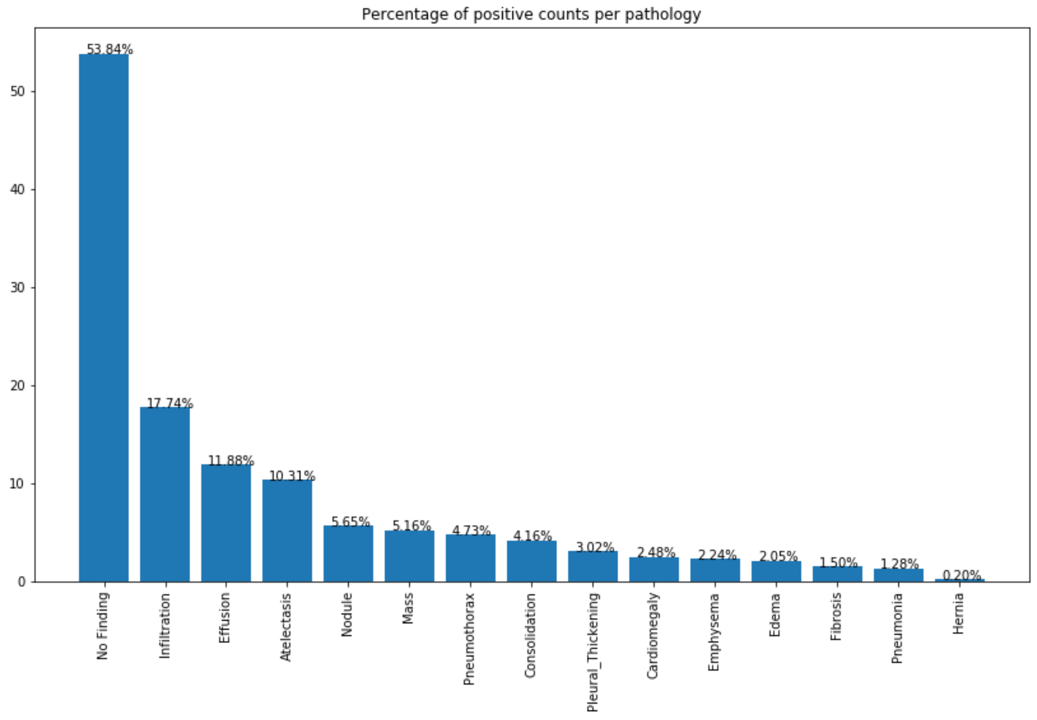


Figure 5.2: Pathology Percentage

Pathology	Percentage
No Finding	53.84%
Infiltration	17.74%
Effusion	11.88%
Atelectasis	10.31%
Nodule	5.65%
Mass	5.16%
Pneumothorax	4.73%
Consolidation	4.16%
Pleural Thickening	3.02%
Cardiomegaly	2.48%
Emphysema	2.24%
Edema	2.05%
Fibrosis	1.50%
Pneumonia	1.28%
Hernia	0.20%

Table 5.2: Pathology Percentage

Figure 5.3 shows a sample of the X-rays in the dataset:

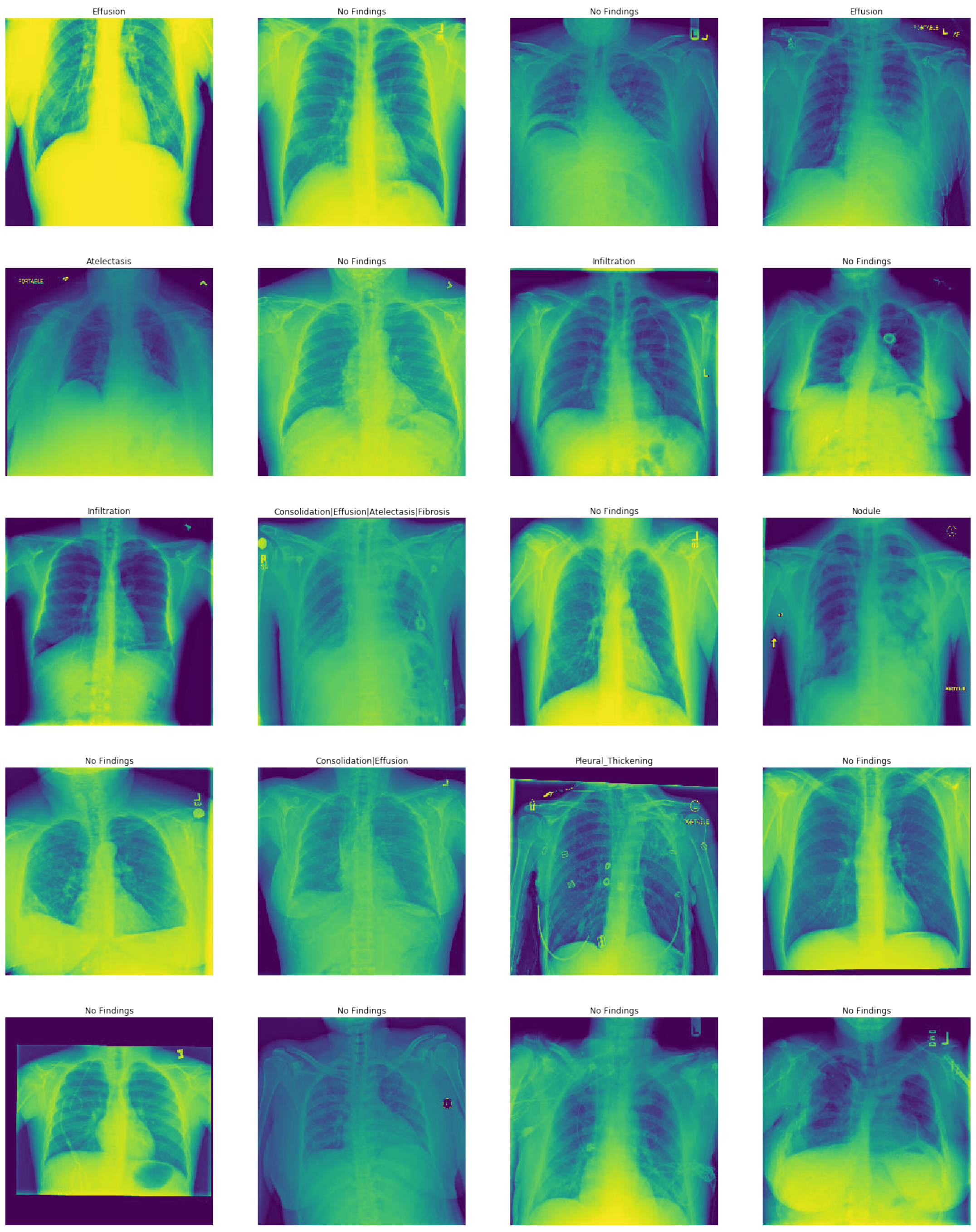


Figure 5.3: Sample X-rays

The chord diagram in figure 5.4 shows the interrelationships between the different pathology types, for instance we can see that there is a high correlation between Infiltration and Effusion.

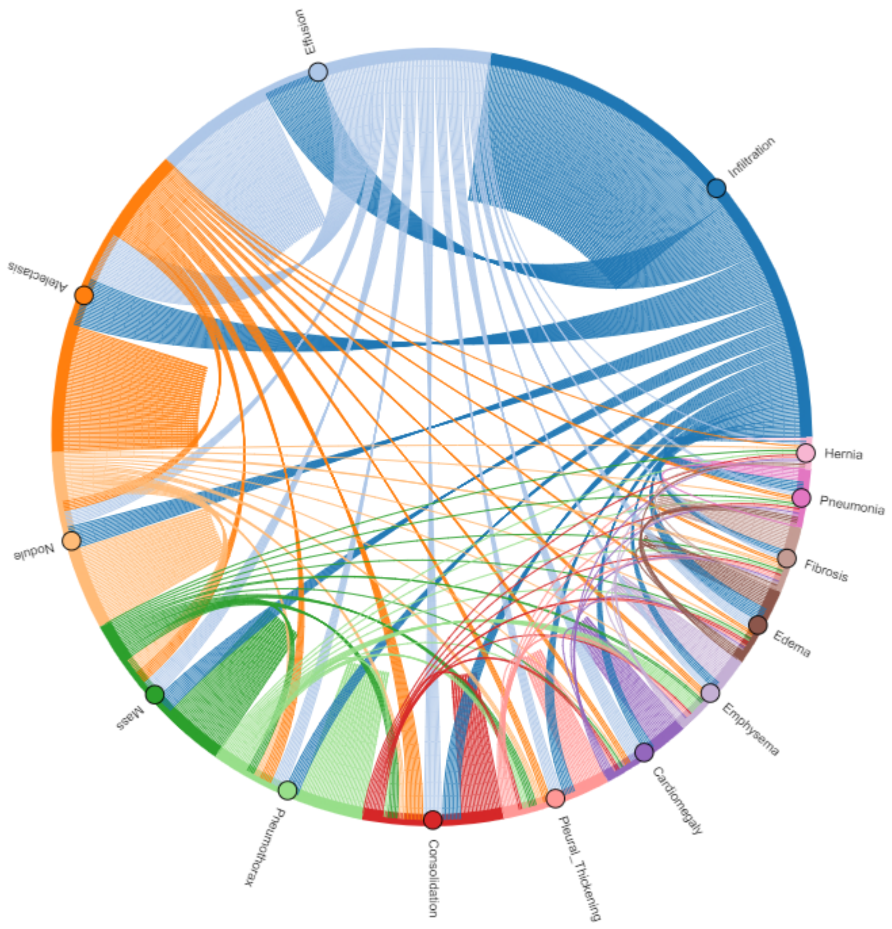


Figure 5.4: Pathology Inter-dependencies

Figure 5.5 shows the counts of single-label instances and instances with multiple pathologies, per pathology. There is a lot of images with multi-pathology labels, which indicates that we should tackle the problem as a multi-label and do not reduce it to a multi-class.

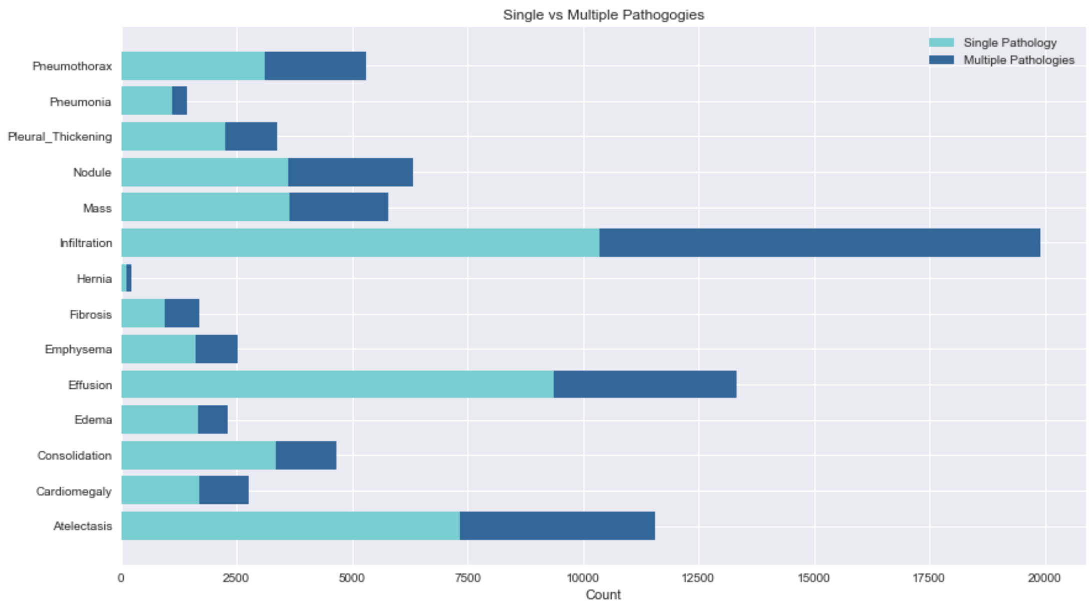


Figure 5.5: Labels Composition

Figure 5.6 shows the composition of instances containing pneumonia. As can be seen most of the cases that were diagnosed with pneumonia were also infected with other disease types, only 22.5% have pneumonia only (labels with less than 12 cases are omitted from the chart and they are all combined under "Other").

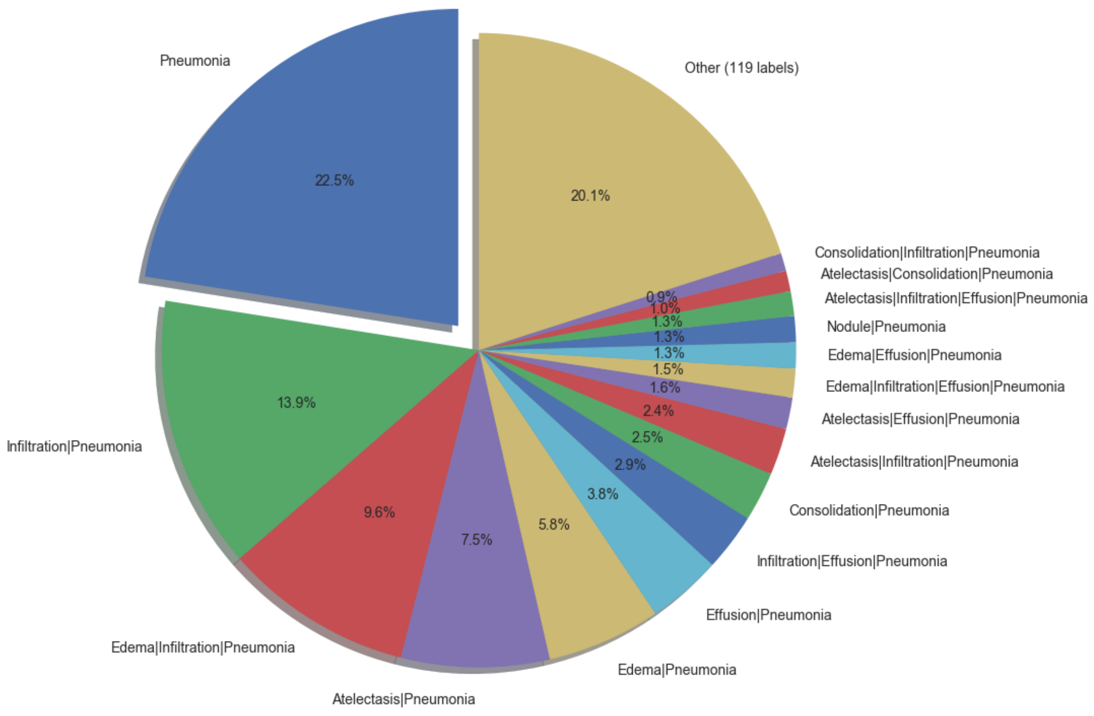


Figure 5.6: Pneumonia Labels

5.2 Images Preprocessing

We applied some preprocessing to the images before running the model:

1. Resized images to 128x128
2. We applied Contrast limited adaptive histogram equalization (CLAHE) to the images
3. The data was split as shown in table 5.3 and figure 5.7

Training	78,484(70%)
Validation	22,424 (20%)
Testing	11,212 (10%)

Table 5.3: NIH Dataset Split

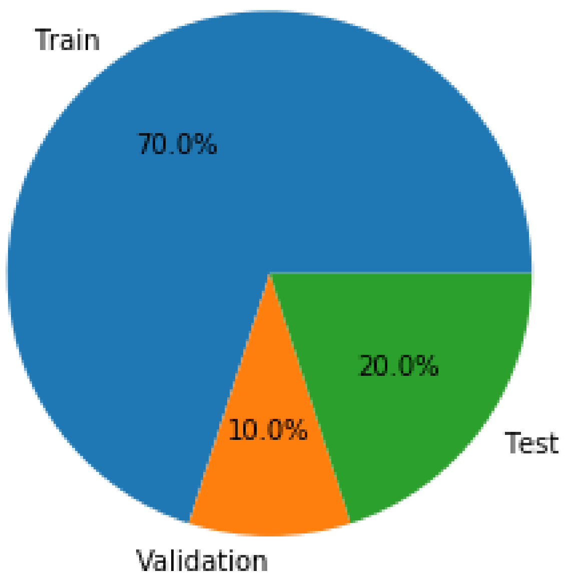


Figure 5.7: NIH Dataset Split

Figure 5.8 shows some randomly selected images without CLAHE:

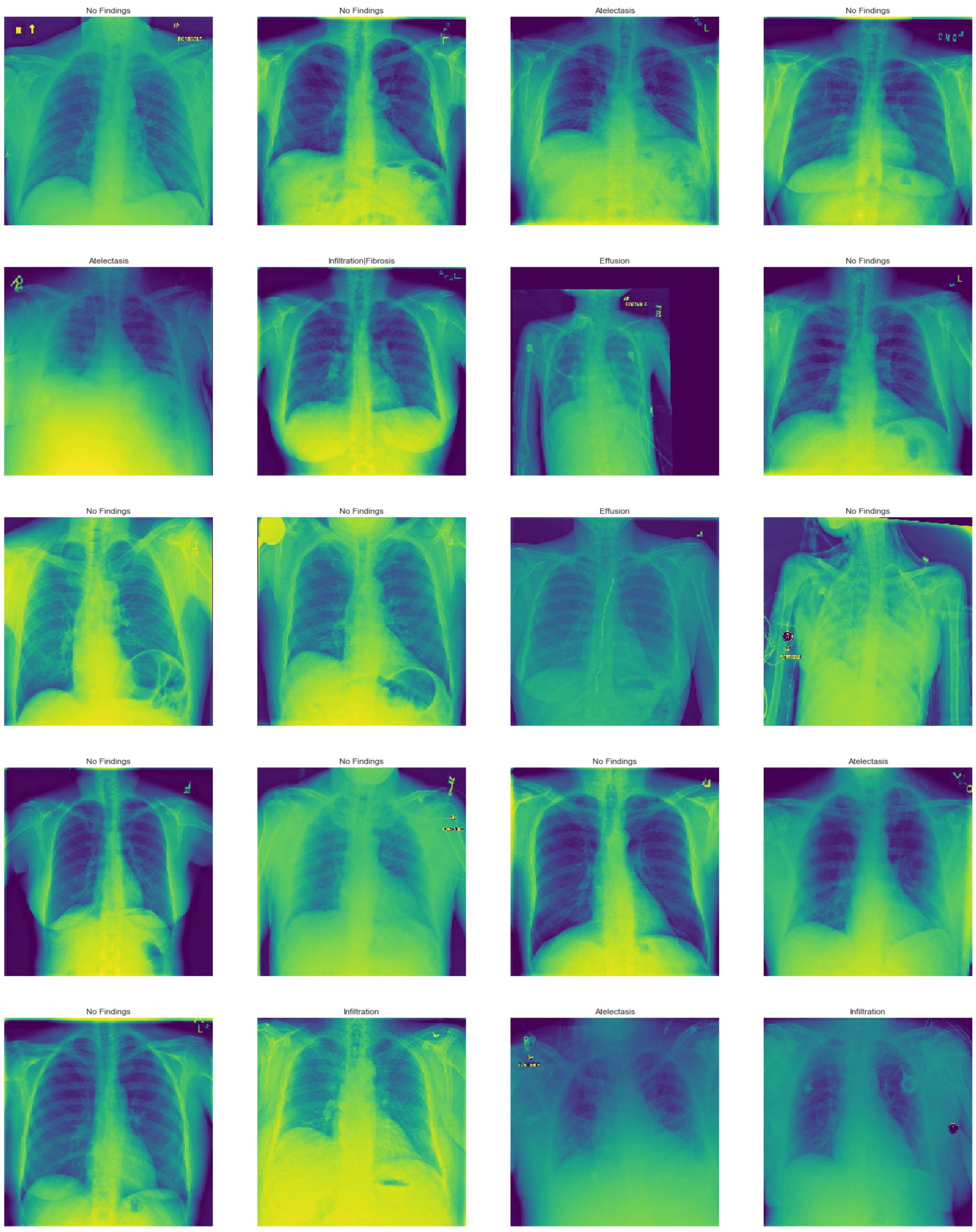


Figure 5.8: NIH Sample Without CLAHE

Figure 5.9 shows some randomly selected images with CLAHE:

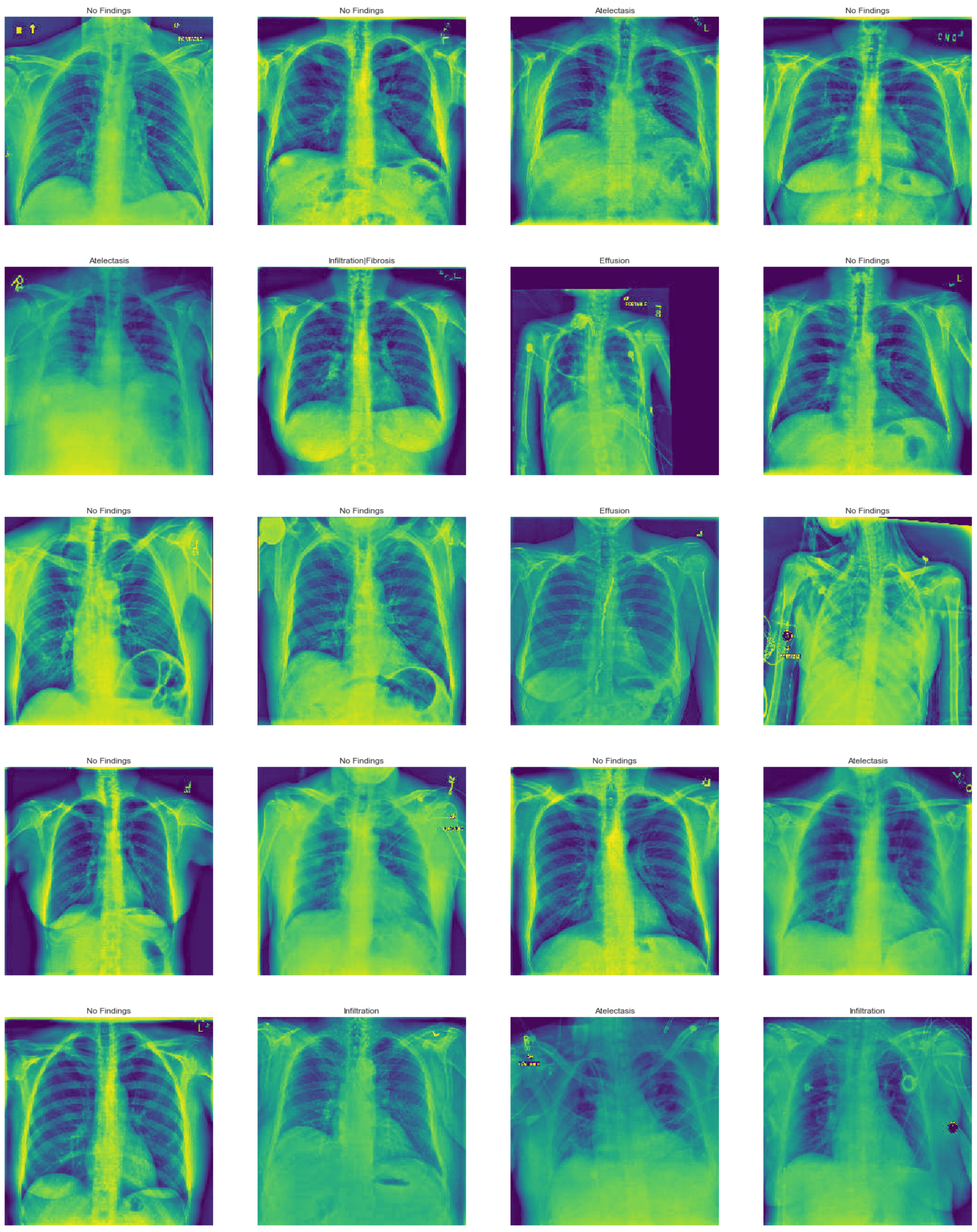


Figure 5.9: NIH Sample With CLAHE

5.3 The Capsule Network

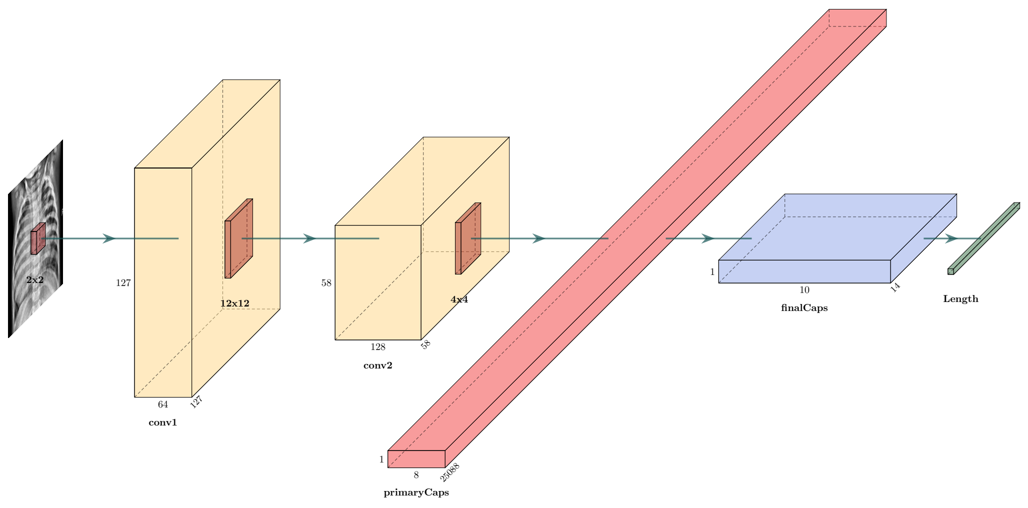


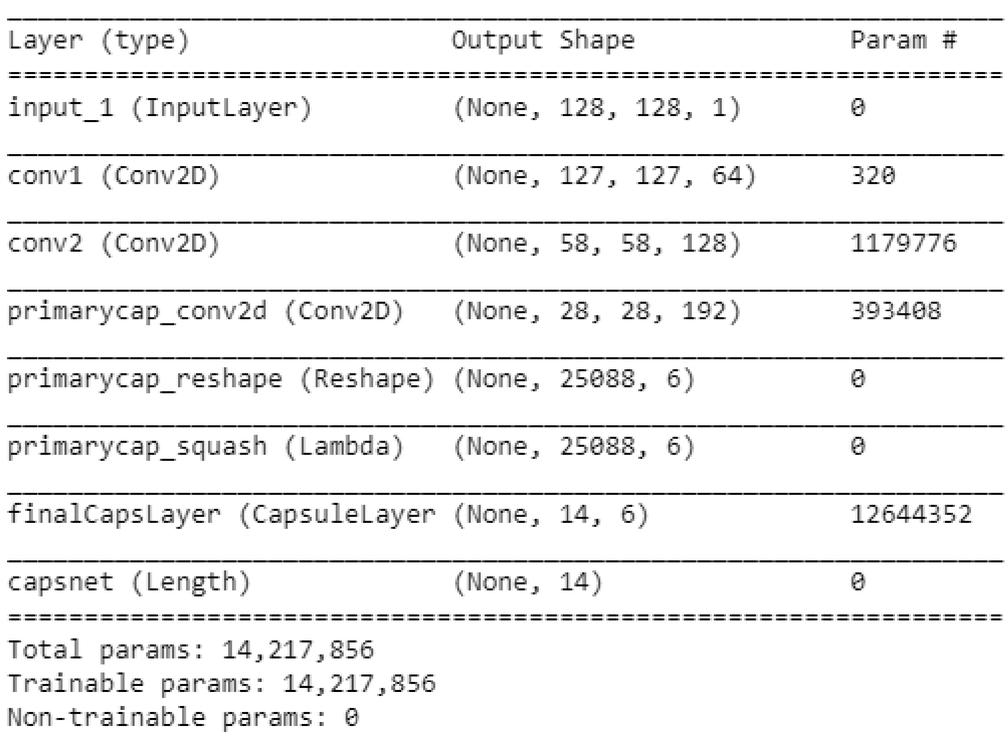
Figure 5.10: Multi-Pathology Capsules Network

The capsule network we used can be described as follows:

- Starts with a convolutional layer using kernel size 2x2, stride 1, 64 filters, and rectified linear unit activation function
- Followed by another convolutional layer with kernel size 12x12, stride 2, 128 filters, and rectified linear unit activation function
- Followed by a primary capsules layer, using kernel size 4x4, stride 2, 32 channels, and 6 dimensions. This layer takes the output from the last convolutional layer and produces vectors (squashed with 6 dimensions).
- The second capsule layer consists of 14 capsules corresponding to the pathology types. The dimension of each vector is 6 and the length of the vector is the probability of detecting the corresponding pathology. The values of the vector represent different instantiation parameters to be learned by the model.
- The last layer of the network is an activation layer that outputs the length of each vector.

- We used Adam optimizer, and a custom loss function.

Figure 5.11 shows a summary of the model.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 128, 128, 1)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 127, 127, 64)      320
_________________________________________________________________
conv2 (Conv2D)               (None, 58, 58, 128)       1179776
_________________________________________________________________
primarycap_conv2d (Conv2D)   (None, 28, 28, 192)       393408
_________________________________________________________________
primarycap_reshape (Reshape) (None, 25088, 6)          0
_________________________________________________________________
primarycap_squash (Lambda)   (None, 25088, 6)          0
_________________________________________________________________
finalCapsLayer (CapsuleLayer (None, 14, 6)             12644352
_________________________________________________________________
capsnet (Length)             (None, 14)                0
=================================================================
Total params: 14,217,856
Trainable params: 14,217,856
Non-trainable params: 0
```

Figure 5.11: Multi-Pathology Capsules Network Summary

5.4 Handling the Class Imbalance

As can be seen in table 5.2, the NIH dataset is highly imbalanced, some classes have less than 1% of the data. To tackle this issue we tried three techniques:

- Undersampling
- Oversampling
- Custom loss function

The results we got for using a custom loss function were the best.

5.4.1 Undersampling

We reduced the maximum number of images per category to 3000 images and we used binary cross entropy loss function:

We achieved 0.61 average AUROC and 0.55 AUROC for pneumonia. Figure 5.12 shows the associated ROC curves.

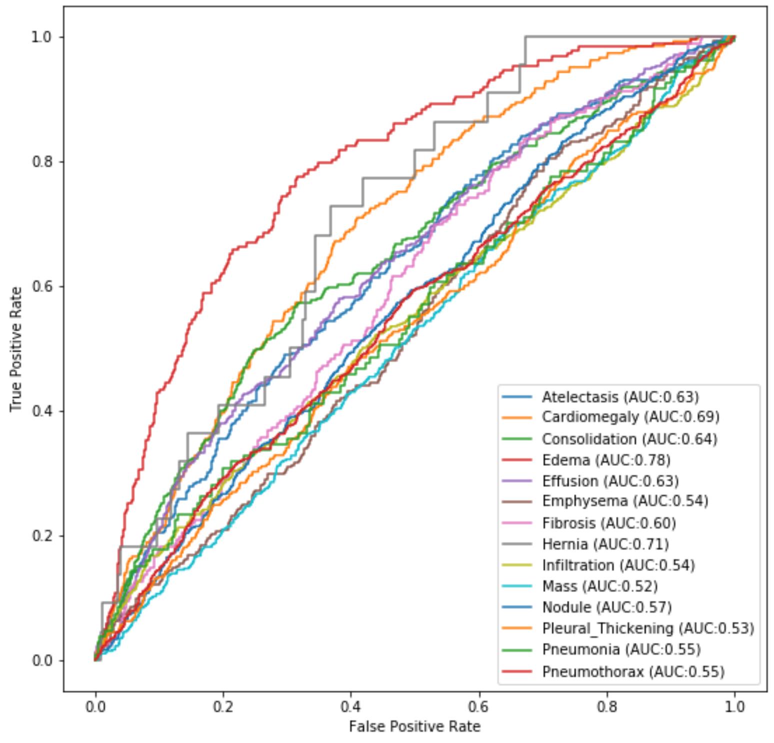


Figure 5.12: NIH Undersampling ROC

5.4.2 Oversampling

For oversampling we used SMOTE (Synthetic Minority Over-sampling TEchnique), we reduced the maximum number of images per category to 5000, and then used SMOTE to oversample the classes with images less than 5000.

We achieved 0.61 average AUROC and 0.55 AUROC for pneumonia, Figure 5.13 shows the associated ROC curves.

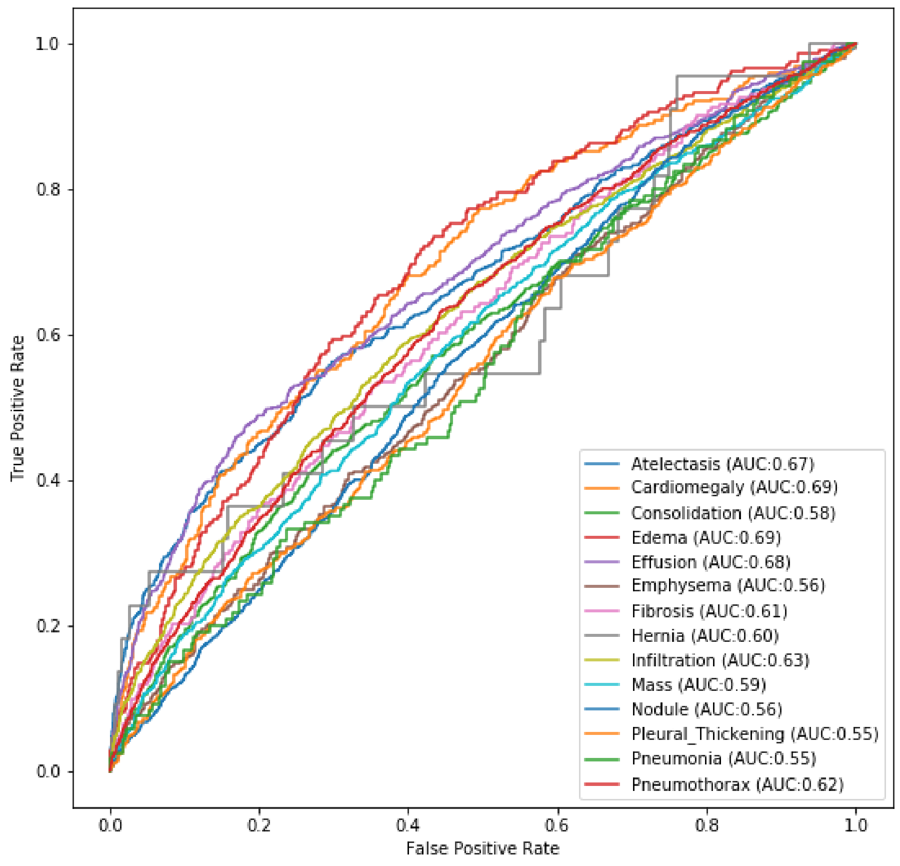


Figure 5.13: NIH Oversampling ROC

5.4.3 Custom Loss Function

Binary cross entropy loss function is commonly used for multi-label problems, its defined as below:

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{i=N}[T_i log(P_i) + (1 - T_i) log(1 - P_i)] \tag{5.1}$$

N is the total number of pathologies (in our case its 14)
T_i is 1 if the input image is labeled with pathology i else its 0
P_i is the predicted probability associated with pathology i

To tackle the class imbalance we used a custom loss function based on the binary cross entropy loss function but gives higher importance to the classes with very little occurrences. To do that, we split the binary cross entropy loss function to the summation of two losses, the first is the one associated with true labels, and the other is associated with the false labels. We refer to the first as the true loss and the second as the false loss, and they are defined as below:

$$tLoss = -\frac{1}{nbTrue}\sum log(TP_i) \tag{5.2}$$

Where TP are the probabilities associated with true labels and $nbTrue$ is the number of true labels in the batch.

$$fLoss = -\frac{1}{nbFalse}\sum log(1 - FP_i) \tag{5.3}$$

Where FP_i are the probabilities associated with a false labels and $nbFalse$ is the number of false labels in the batch.

The loss is defined as a weighted mean of the two means:

$$Loss = (tLoss + 0.9 * fLoss)/2 \tag{5.4}$$

5.5 Results

Using the custom loss function described in section 5.4.3 we were able to achieve our best results with 0.7536 average AUROC and 0.6788 AUROC for pneumonia as can be seen in Figure 5.14.

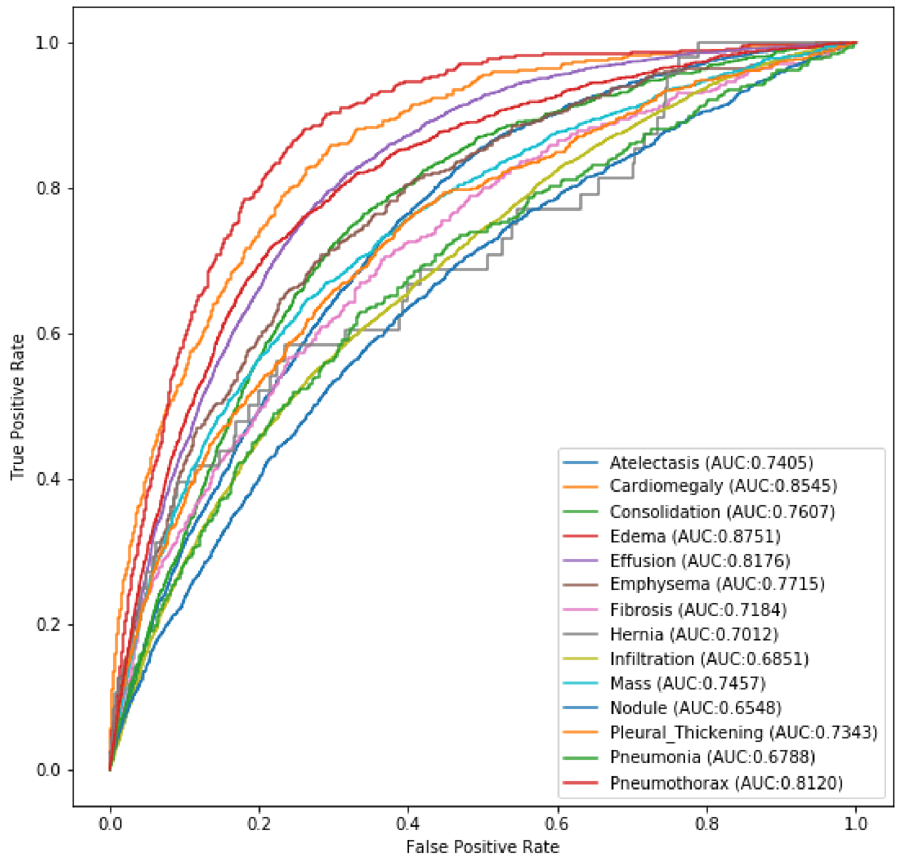


Figure 5.14: Multi-Pathology ROC

Table 5.4 shows our results in comparison with the top 3 research papers tackling the same problem (AUROC):

	Wang et al. (2017)	Yao et al. (2017)	CheXNet	Ours
Atelectasis	0.7003	0.772	0.8094	0.7405
Cardiomegaly	0.8100	0.904	0.9248	0.8545
Effusion	0.7585	0.859	0.8638	0.8176
Infiltration	0.6614	0.695	0.7345	0.6851
Mass	0.6933	0.792	0.8676	0.7457
Nodule	0.6687	0.717	0.7802	0.6548
Pneumonia	0.6580	0.713	0.7680	0.6788
Pneumothorax	0.7993	0.841	0.8887	0.8120
Consolidation	0.7032	0.788	0.7901	0.7607
Edema	0.8052	0.882	0.8878	0.8751
Emphysema	0.8330	0.829	0.9371	0.7715
Fibrosis	0.7859	0.767	0.8047	0.7184
Pleural Thickening	0.6835	0.765	0.8062	0.7343
Hernia	0.8717	0.914	0.9164	0.7012
Average	0.7451	0.8027	0.8414	0.7536

Table 5.4: Multi-Pathology Results (AUROC)

As can be seen in table 5.4, our model was able to outperform Wang et al. [3] on most disease types including pneumonia and on average AUROC.

Pathology	AUROC	Accuracy	Specificty	Sensitivity
Atelectasis	0.7405	0.3895	0.3281	0.9344
Cardiomegaly	0.8545	0.8736	0.8806	0.5869
Consolidation	0.7607	0.6879	0.6859	0.7361
Edema	0.8751	0.8762	0.8811	0.6484
Effusion	0.8176	0.5512	0.5003	0.9248
Emphysema	0.7715	0.8932	0.9043	0.4076
Fibrosis	0.7184	0.9682	0.9816	0.0944
Hernia	0.7012	0.9979	1.0000	0.0000
Infiltration	0.6851	0.2243	0.0595	0.9879
Mass	0.7457	0.6716	0.6702	0.6969
Nodule	0.6548	0.3789	0.3524	0.8158
Pleural Thickening	0.7343	0.8583	0.8721	0.4282
Pneumonia	0.6788	0.9717	0.9843	0.0682
Pneumothorax	0.8120	0.6719	0.6645	0.8207
Average	0.7536	0.7153	0.6975	0.5822

Table 5.5: Multi-Pathology Results

Figure 5.15 shows a randomly selected sample of X-ray images and their predictions (top 4 predictions shown in the figure):

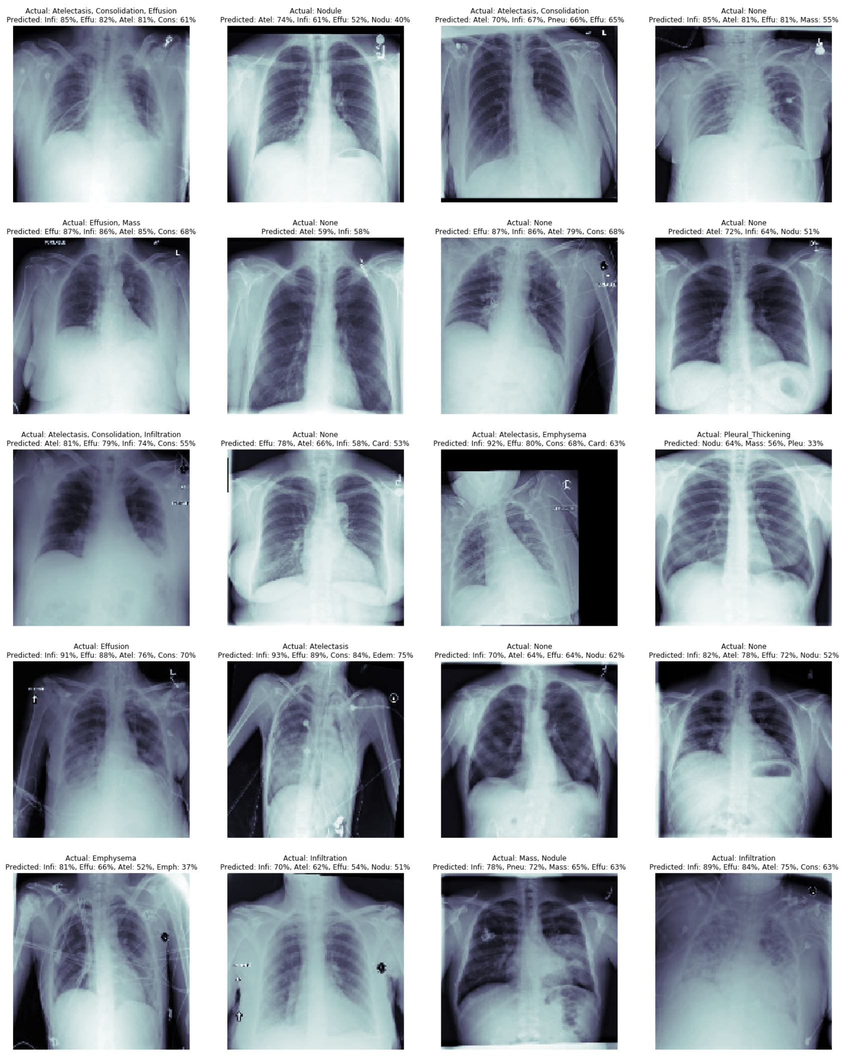

Actual: Atelectasis, Consolidation, Effusion
Predicted: Infi: 85%, Effu: 82%, Atel: 81%, Cons: 61%
Actual: Nodule
Predicted: Atel: 74%, Infi: 61%, Effu: 52%, Nodu: 40%
Actual: Atelectasis, Consolidation
Predicted: Atel: 70%, Infi: 67%, Pneu: 66%, Effu: 65%
Actual: None
Predicted: Infi: 85%, Atel: 81%, Effu: 81%, Mass: 55%
Actual: Effusion, Mass
Predicted: Effu: 87%, Infi: 86%, Atel: 85%, Cons: 68%
Actual: None
Predicted: Atel: 59%, Infi: 58%
Actual: None
Predicted: Effu: 87%, Infi: 86%, Atel: 79%, Cons: 68%
Actual: None
Predicted: Atel: 72%, Infi: 64%, Nodu: 51%
Actual: Atelectasis, Consolidation, Infiltration
Predicted: Atel: 81%, Effu: 79%, Infi: 74%, Cons: 55%
Actual: None
Predicted: Effu: 78%, Atel: 66%, Infi: 58%, Card: 53%
Actual: Atelectasis, Emphysema
Predicted: Infi: 92%, Effu: 80%, Cons: 68%, Card: 63%
Actual: Pleural_Thickening
Predicted: Nodu: 64%, Mass: 56%, Pleu: 33%
Actual: Effusion
Predicted: Infi: 91%, Effu: 88%, Atel: 76%, Cons: 70%
Actual: Atelectasis
Predicted: Infi: 93%, Effu: 89%, Cons: 84%, Edem: 75%
Actual: None
Predicted: Infi: 70%, Atel: 64%, Effu: 64%, Nodu: 62%
Actual: None
Predicted: Infi: 82%, Atel: 78%, Effu: 72%, Nodu: 52%
Actual: Emphysema
Predicted: Infi: 81%, Effu: 66%, Atel: 52%, Emph: 37%
Actual: Infiltration
Predicted: Infi: 70%, Atel: 62%, Effu: 54%, Nodu: 51%
Actual: Mass, Nodule
Predicted: Infi: 78%, Pneu: 72%, Mass: 65%, Effu: 63%
Actual: Infiltration
Predicted: Infi: 89%, Effu: 84%, Atel: 75%, Cons: 63%

Figure 5.15: Multi-Pathology Sample Predictions

Chapter 6

Additional Experiments

6.1 Face Recognition using CNN

Even though we know that CNNs by design does not have the capability of detecting the spatial relationship between the different entities of an image, we took an already existing CNN [25] that was trained on face recognition and then we tested it on an altered face image.

The CNN was trained on Olivetti dataset [26] which consists of 400 images belonging to 40 people (10 each):

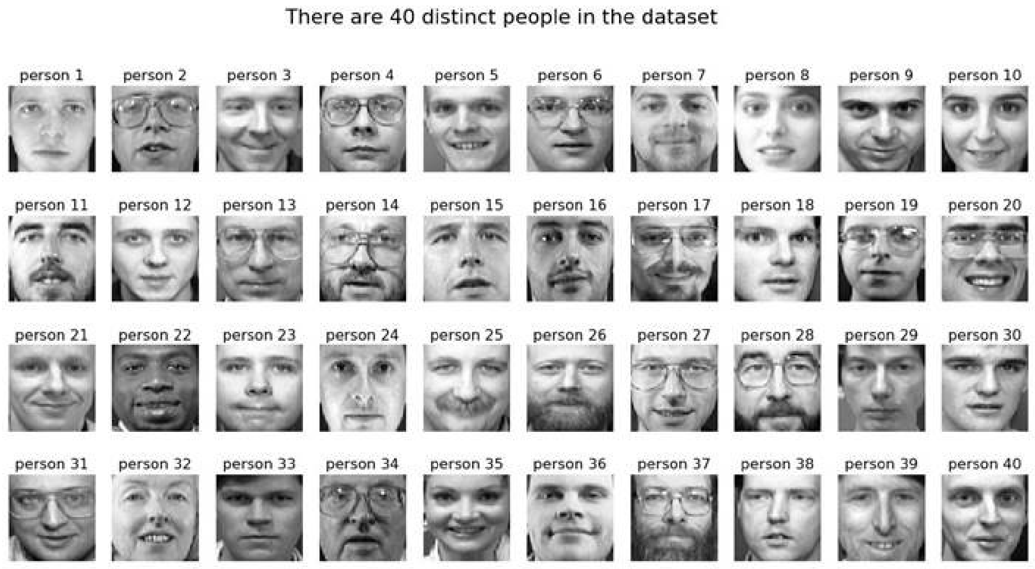


Figure 6.1: Olivetti Dataset

When we tested the CNN we got the same prediction for the two faces shown in figure 6.2.

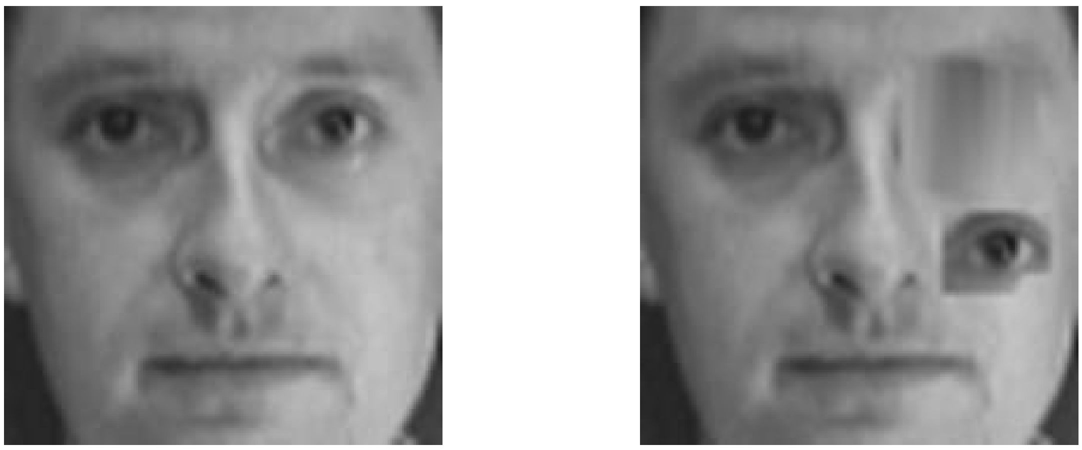

(a) First Input Image (b) Second Input Image

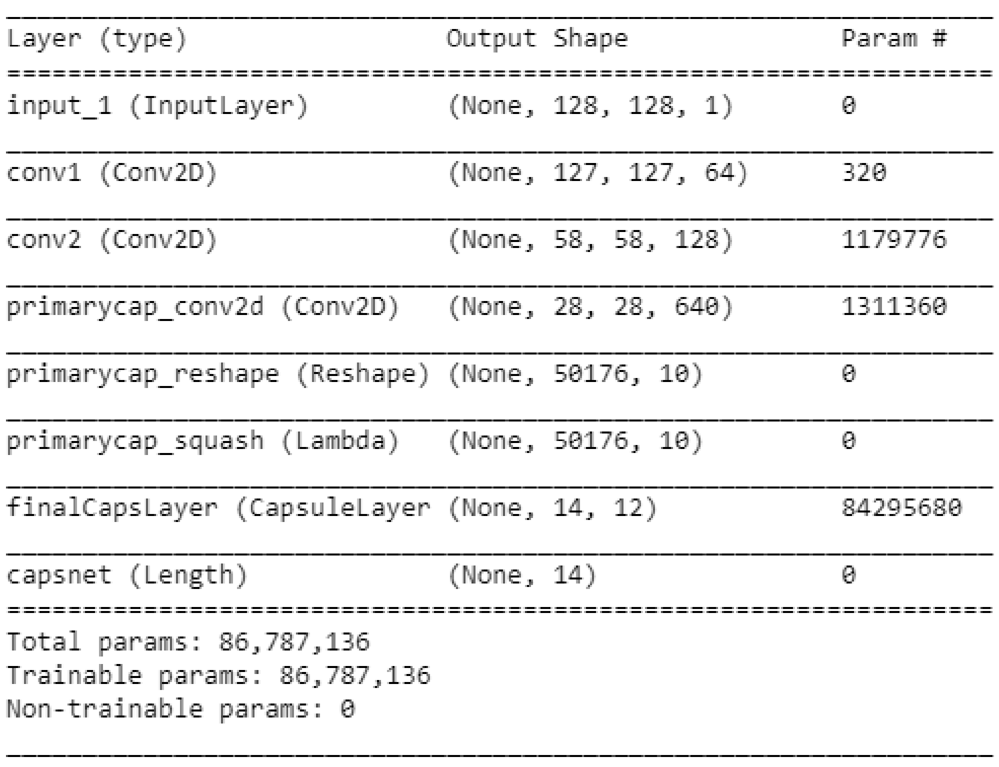
Figure 6.2: Face CNN Test

6.2 Multi-Pathology detection Without CLAHE

We tested a model on the multi-pathology dataset without applying CLAHE, the model can be described as follows:

- Two convolutional layers followed by two capsule layers
- The first convolutional layer consists of 64 filters, kernel size 2x2, and stride =1
- The second convolutional layer consists of 128 filters, kernel size 12x12, and stride =2
- The first capsule layer uses 10 dimensions, with 64 channels, kernel size 4x4 and stride =2
- The second capsule layer consists of 14 capsules with 12 dimensions each
- We used a weighted binary cross entropy loss function where we multiplied the true loss by (nbTrue+nbFalse)/nbTrue, and we multiplied the false loss by (nbTrue+nbFalse)/nbFalse, where nbTrue is the number of ones in the true label and nbFalse is the number of zeros.

Figure 6.3 shows the summary of the model used.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 128, 128, 1)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 127, 127, 64)      320
_________________________________________________________________
conv2 (Conv2D)               (None, 58, 58, 128)       1179776
_________________________________________________________________
primarycap_conv2d (Conv2D)   (None, 28, 28, 640)       1311360
_________________________________________________________________
primarycap_reshape (Reshape) (None, 50176, 10)         0
_________________________________________________________________
primarycap_squash (Lambda)   (None, 50176, 10)         0
_________________________________________________________________
finalCapsLayer (CapsuleLayer (None, 14, 12)            84295680
_________________________________________________________________
capsnet (Length)             (None, 14)                0
=================================================================
Total params: 86,787,136
Trainable params: 86,787,136
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.3: Without CLAHE Model Summary

Using this setup we achieved an average AUROC of 0.70 and 0.66 for pneumonia.

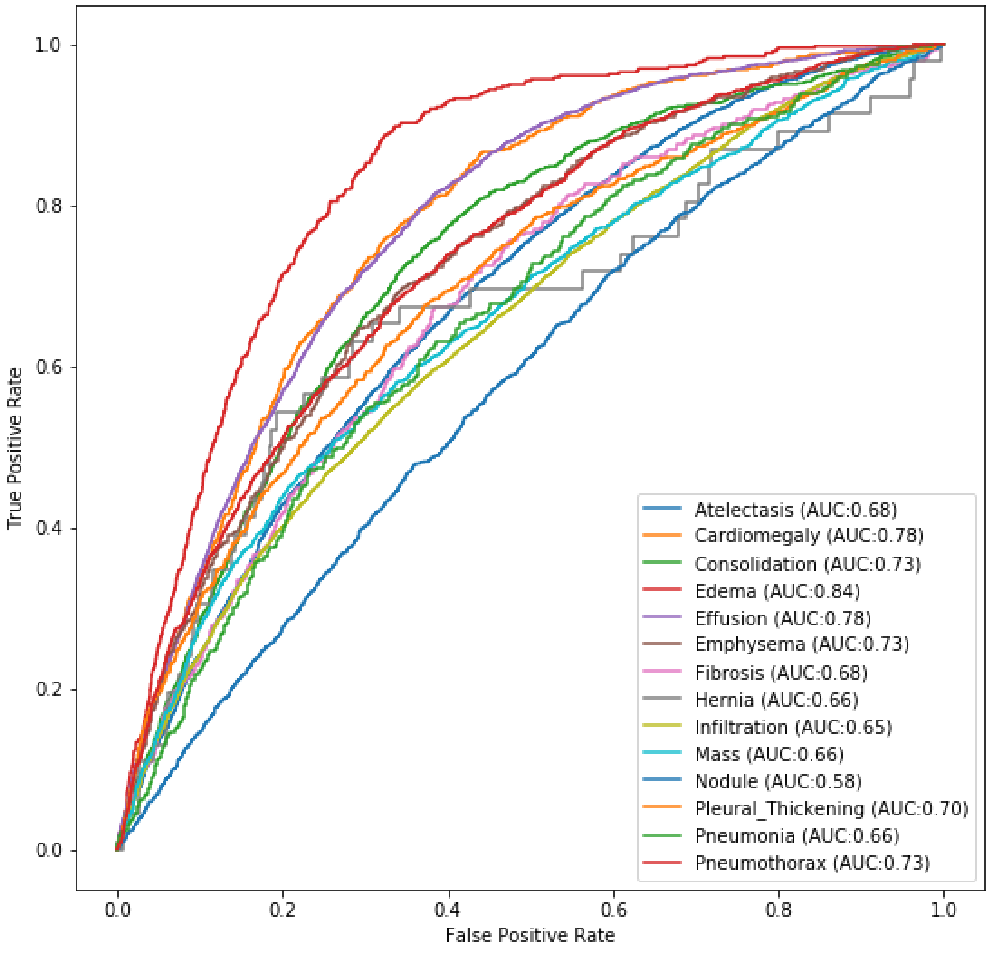


Figure 6.4: Without CLAHE Model ROC

Chapter 7

Conclusion and Future Work

Using our capsule network on the pneumonia dataset [2] yielded better results than the most voted kernel on Kaggle [9] as can be seen in table 7.1.

	CNN	CapsNet (Ours)
Accuracy	0.9054	0.9475
Recall	0.8756	0.9385
Precision	0.9971	0.9905
F1-Score	0.9324	0.9638

Table 7.1: Pneumonia Detection Results

A recall of 0.9385 is very important especially we are dealing with a medical problem and we need to minimize the false negatives.

On the other side, capsule networks performed well on the multi-pathology problem and outperformed convolutional neural networks on many disease types, however it could not be as good as the top CNNs handling the same problem as shown in table 7.2.

	Wang et al. (2017)	Yao et al. (2017)	CheXNet	Ours
Atelectasis	0.7003	0.772	0.8094	0.7405
Cardiomegaly	0.8100	0.904	0.9248	0.8545
Effusion	0.7585	0.859	0.8638	0.8176
Infiltration	0.6614	0.695	0.7345	0.6851
Mass	0.6933	0.792	0.8676	0.7457
Nodule	0.6687	0.717	0.7802	0.6548
Pneumonia	0.6580	0.713	0.7680	0.6788
Pneumothorax	0.7993	0.841	0.8887	0.8120
Consolidation	0.7032	0.788	0.7901	0.7607
Edema	0.8052	0.882	0.8878	0.8751
Emphysema	0.8330	0.829	0.9371	0.7715
Fibrosis	0.7859	0.767	0.8047	0.7184
Pleural Thickening	0.6835	0.765	0.8062	0.7343
Hernia	0.8717	0.914	0.9164	0.7012
Average	0.7451	0.8027	0.8414	0.7536

Table 7.2: Multi-Pathology Results (AUROC)

Capsule networks are kind of new to the machine learning community, the first paper published by Geoffrey Hinton and his team was in late 2017. They do introduce very important concepts and try to mimic the inverse graphics process done by the human brain to tackle many limitations in regular convolutional networks. We need a little more time to see how the machine learning community can build on these new concepts. Convolutional neural networks concepts dates back to 1959 (the work done by David Hubel and Torsten Wiesel [12]), later in 1980, the first CNN model [14] was proposed by Kunihiko Fukushima, however the first CNN as we know it today saw the light in 1998 through the work of Yann LeCun on LeNet-5 [15].

The results we achieved in this thesis are promising and performing better than the best CNN models in some cases, and comparable results in other cases. They are definitely a step in the right direction, but there is a lot of space for improvements.

In the future, i will be working to enhance the model of the multi-pathology case through different hyper parameters, different loss functions, and architectures hoping to achieve better results than CheXNet [7] and we will see what the machine learning community brings into capsule networks. Also, i will try to leverage what we achieved in this thesis and train the model on the X-ray images that are at the medical center of the American university of Beirut, this involves creating a new dataset.

Appendix A

Abbreviations

ML	Machine Learning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
SMOTE	Synthetic Minority Over-sampling TEchnique
KNN	K-Nearest Neighbors
CLAHE	Contrast Limited Adaptive Histogram Equalization

Appendix B

Software and Hardware Used

We used the below software/libraries:

1. Tensorflow 1.14.0
2. Keras 2.2.4
3. Python 3.7.4
4. Keras implementation of capsule networks [27]

For the hardware we used the HPC (high performance computing) infrastructure at the American University of Beirut. And we ran our models on a node with 8 cores, 16 GB RAM, and NVIDIA GRID vGPU grid_v100d-16q. For our reported best model each epoch was taking around an hour (we used 50 epochs with early stop).

Bibliography

[1] Wikipedia, *Pneumonia*, 2020 (accessed February 7, 2020). https://en.wikipedia.org/wiki/Pneumonia.

[2] Kaggle, *Pneumonia Dataset*, 2018 (accessed February 7, 2020). https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia.

[3] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2097–2106, 2017.

[4] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 3856–3866, Curran Associates, Inc., 2017.

[5] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with EM routing," in *International Conference on Learning Representations*, 2018.

[6] L. Yao, E. Poblenz, D. Dagunts, B. Covington, D. Bernard, and K. Lyman, "Learning to diagnose from scratch by exploiting dependencies among labels," *arXiv preprint arXiv:1710.10501*, 2017.

[7] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, *et al.*, "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning," *arXiv preprint arXiv:1711.05225*, 2017.

[8] WHO, *WHO Pneumonia*, 2019 (accessed February 7, 2020). https://www.who.int/news-room/fact-sheets/detail/pneumonia.

[9] A. Nain, *Beating everything with Depthwise Convolution*, 2018 (accessed January 3, 2020). https://www.kaggle.com/aakashnain/beating-everything-with-depthwise-convolution.

[10] D. Hebb, *Hebbian theory*, Online (accessed January 6, 2020). https://en.wikipedia.org/wiki/Hebbian_theory.

[11] Wikipedia, *Artificial Neuron*, Online (accessed January 6, 2020). https://en.wikipedia.org/wiki/File:Artificial_neuron.png.

[12] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[13] Online, *Hubel Experiment*, 2013 (accessed January 6, 2019). https://goodpsychology.wordpress.com/2013/03/13/235.

[14] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[15] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] Brilliant, *Convolutional Neural Network*, Online (accessed January 6, 2020). https://brilliant.org/wiki/convolutional-neural-network/.

[17] Wikipedia, *k-nearest neighbors algorithm*, Online (accessed January 6, 2020). https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[18] M. Pechyonkin, *Understanding Hinton's Capsule Networks. Part I: Intuition*, 2017 (accessed January 8, 2020). https://medium.com/ai\%C2\%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.

[19] J. Hui, *Understanding Dynamic Routing between Capsules (Capsule Networks)*, 2017 (accessed January 8, 2020). https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules.

[20] M. Pechyonkin, *Understanding Hinton's Capsule Networks. Part 2. How Capsules Work*, 2017 (accessed January 8, 2020). https://pechyonkin.me/capsules-2/.

[21] R. Kunert, *SMOTE explained for noobs - Synthetic Minority Over-sampling TEchnique line by line*, 2017 (accessed January 4, 2020). http://rikunert.com/SMOTE_explained.

[22] M. Sharma, *Histograms in Image Processing with skimage-Python*, 2019 (accessed January 8, 2020). https://towardsdatascience.com/histograms-in-image-processing-with-skimage-python-be5938962935.

[23] Zefram, *Wikipedia - Histogram Equalization*, 2006 (accessed January 8, 2020). https://commons.wikimedia.org/w/index.php?curid=668605.

[24] S. Sudhakar, *Histogram Equalization*, 2017 (accessed January 8, 2020). https://towardsdatascience.com/histogram-equalization-5d1013626e64.

[25] H. Ozen, *Face Recognition by LR, RForest, KNN and CNN*, 2018 (accessed January 4, 2019). https://www.kaggle.com/hakanozen/face-recognition-by-lr-rforest-knn-and-cnn.

[26] S. M. Imran, *Olivetti, Version 1*, 2017, December (accessed January 4, 2019). https://www.kaggle.com/imrandude/olivetti.

[27] X. Guo, *CapsNet-Keras*, 2017 (accessed February 7, 2020). https://github.com/XifengGuo/CapsNet-Keras.