Analyzing Chest X-rays using Convolutional Neural Networks

ACIT4530-1 22V Data Mining at Scale: Algorithms and Systems

Torbir Dhaliwal	S366263
Fernando Gonzalez	S366264
Maria E. Pedersen	S360509

Contents

1 Introduction

Convolutional Neural Networks (CNNs) are a type of artificial neural network able to detect and identify objects in computer images. Due to recent advancements in deep learning, computer power, and the emergence of new, highly infectious diseases such as COVID-19, these networks have received major attention in the realm of computer-aided diagnosis of diseases based on medical imagery. In this paper we dissect the architectures of three prominent CNN models (AlexNet, Inception / GoogleNet, and DenseNet) as well as compare their performance when applied to chest x-rays. Chest x-rays are one of the most common medical procedures today and are frequently used to diagnose a range of illnesses such as fibrosis, pneumonia, and hernias.

We begin with a brief overview of CNNs and available datasets relevant to computer-aided diagnosis before diving deep into the structural details of AlexNet, GoogleNet, and DenseNet. After dissecting each model, we analyze their performance metrics and provide a general discussion on advantages and drawbacks of each model. We will also include our implementation code in a separate file.

1.1 What are Convolutional Neural Networks

CNN is a type of deep learning algorithm that can take an input image, assign importance to various objects in an image and learn to differentiate between the images. Structurally speaking, CNNs are similar to Artificial Neural Networks (ANNs) in that they are both made up of nodes that self-optimise through learning. These nodes are usually arranged in different kinds of layers such as: an input layer, one or more hidden layers, and an output layer. The nodes are connected to another and are assigned weights and thresholds (biases) [1]. However, traditional ANNs have issues with the computational complexity that is required when dealing with image data. The CNNs architecture is built to handle image data, hence their popularity in the field of pattern recognition within images.

A typical CNN consists of three types of layers: convolutional layers, pooling layers and fully connected layers. CNNs always start with a convolutional layer, and while more convolutional layers and pooling layers can be stacked in the middle, the networks always end with the fully connected layer. Through the layers, the CNN increases in its complexity, by first focusing on more simple features, such as colors, and later starts to perceive larger elements or shapes. In the end it will identify the object [2].

The convolutional layer applies a convolution operation to the input image. The main objective of this operation is to detect what type of features are present in an image, such as edges. It does so by applying different filters over the image that serve to enhance the most prominent features of the image. The pooling layer is responsible for dimensionality reduction. This layer aims to reduce complexity, improve efficiency and limit risk of overfitting [2]. It does this by reducing the size filtered image and eliminating all the values that don't contribute to the feature extraction. Finally, the fully connected layer is responsible to perform the classification based on the features that have been extracted through the previous layers.

1.2 Dataset Description

According to the Pan American Health Organization, between 70 to 80 percent of all diagnostic problems can be addressed using medical imaging techniques such as x-rays and ultrasounds, and an estimated 3.6 billion x-rays are captured every year. The problem is that most of this data is inconsistent, not centralized, and rarely labeled or annotated. However, in recent years researchers have used new data retrieval and text mining/NLP methods to curate medical imaging datasets compatible with modern AI frameworks.

One such dataset is "ChestX-ray8", made available in 2017 by NIH [3]. This dataset consists of 112,120 images of frontal X-rays from the thorax (chest region) of 30,805 patients. It is an improvement to the one used by Wang [4], adding six more disease categories to make 14 in total. Additionally, over 3,000 extra images were added to the dataset to make it more comprehensive. All X-rays are standardized as 1024 x 1024 greyscale images.

Conditions labeled in the dataset are Cardiomegaly, Hernia, Mass, Infiltration, Effusion, Nodule, Emphysema, Atelectasis, Pneumothorax, Pleural Thickening, Fibrosis, and Consolidation. It is important to note that there can be several images of a single patient and that a patient may have one or more of these conditions. There are cases as well are where the patient doesn't have a condition; these cases are labeled as "No Finding", making 15 labels in total.

2 Methodology

Barring few images, the X-rays are standardized as 1024 x 1024 greyscale images named in the following format: ########_###. The first eight digits correspond to a patient ID and the following three digits are a correlative indicating different pictures from the same patient. In addition to the images, the accompanying database containing 10 different fields for each image. The field we are interested in is called "Finding Labels", which as the name indicates contains the different labels for the diseases associated.

The first challenge we faced was the variable number of labels for each image. To solve this we restructured the base, ending with a database where each image has 15 binary values associated. Each of the binary values correspond to each of the possible diseases, adding one extra value to signify that No Finding was found on that particular X-ray. I.E., an X-ray tagged with Edema and Effusion, will have a 1 on those "columns" and a 0 in all the remaining. Thus, turning this project into a Multi-Variate classification problem. An excerpt of the CSV file is shown in Figure 1.

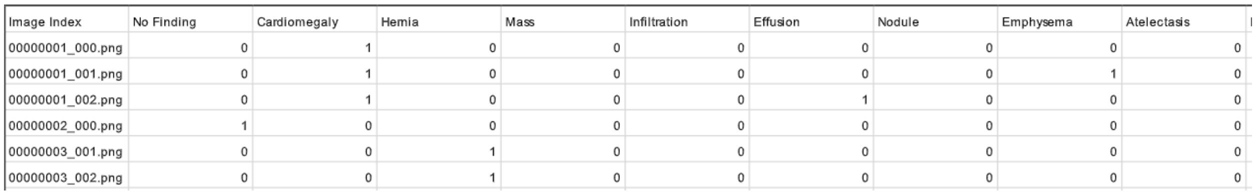


Image Index	No Finding	Cardiomegaly	Hemia	Mass	Infiltration	Effusion	Nodule	Emphysema	Atelectasis
00000001_000.png	0	1	0	0	0	0	0	0	0
00000001_001.png	0	1	0	0	0	0	0	1	0
00000001_002.png	0	1	0	0	0	1	0	0	0
00000002_000.png	1	0	0	0	0	0	0	0	0
00000003_001.png	0	0	1	0	0	0	0	0	0
00000003_002.png	0	0	1	0	0	0	0	0	0

Figure 1 CSV file of labels

To enable the completion of this project with our available computing power, we analyzed a sample corresponding to one percent of the images in the dataset (1,120 images). The selection of these images was done at random, checking to avoid duplicates and non-standard images. Later we checked the labels associated to the images selected to ensure that all labels had representation in the sample. We didn’t concern ourselves into making a sample that had the same proportion of labels of the original base since we believe the models we selected are able to generate accurate predictions based on their architecture, and not on necessarily long trainings.

As per the implementation for the models, we utilized Python for all of them. We used the Pytorch and Tensorflow/Keras frameworks to replicate the models’ architecture based on the original publications, trying to preserve them as much as possible but modifying them to better suit our challenge. The code was written and executed using the remote services offered by Google Collaboratory.

3 CNN Models

While researching the subject we encountered plenty different models, each with its share of history and good reasons to be the subject of this project. We decided to settle for the following models due to their reliability, relevance, legacy, and impact to the field of machine vision: AlexNet, InceptionV2, and DenseNet. We feel that these models all approach the challenge of machine vision with different perspectives and ideas, and each had different contributions to the field. Overall, we felt the architectures are varied enough that replicating each of them can be both challenging and provide valuable insight and knowledge.

3.1 AlexNet

AlexNet was designed by Alex Krizheysky in collaboration with Ilya Sutskever and Geoffrey Hinton. AlexNet is a type of CNN that won the ImageNet Large Scale Visual Recognition Challenge (LSVRC) in 2012. LSVRC is a competition where research teams can apply their algorithms on the huge dataset of labelled images called ImageNet. Here the team that that achieve the highest accuracy on several recognition tasks wins.

The architecture of AlexNet consists of eight layers with weights; five convolutional layers and one fully connected layer. Each layer ends with a ReLU activation except for the last one, which outputs using a Softmax with distribution over the class labels. A visualization of the AlexNet architecture is illustrated in Figure 2.

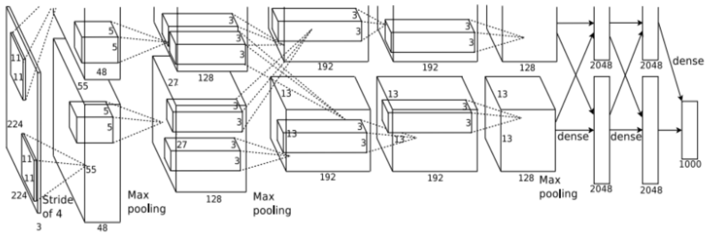


Figure 2: AlexNet Architecture

Dropout is applied after the first two fully connected layers and Max-pooling is applied after the first, second and fifth convolutional layer. The kernels belonging to the second, fourth and fifth convolutional layers are only connected to the kernel maps which occupy on the same GPU in the previous layers. However, the kernels of the third convolutional layer are connected to all the kernel maps of the previous layer. Lastly, the fully connected layer's neurons are connected to all the neurons in the previous layer [5].

One of the most important features of AlexNet is its use of ReLU (Rectified Linear Unit) nonlinearity. In fact, AlexNet showed that ReLU nonlinearity was a much faster approach to deep CNNs than activation functions like Tanh or Sigmoid [5]. Under normal circumstances, architectures like AlexNet that uses a high number of many parameters tend to overfit easily. AlexNet however, utilizes Dropout layers to reduce the number of parameters carried over the net to disincentivize overfitting.

AlexNet is considered to be a milestone in the realm of CNN image classification since many methods first utilized/popularized in this architecture such as: Conv+Pooling design, Dropout, GPU utilization, Parallel Computing and relying mainly on ReLU activations over the other options, are still the industrial standard to this day for computer vision. Nowadays this is thought as an older model that underperforms compared to more complex and deep architectures. On comparative tests, complex models such as ResNet and GoogleNet have shown to surpass AlexNet's performance.

3.1.1 Implementation (adaptation and improvements)

When implementing AlexNet for this project, there was not a lot of adjustments made to the architecture. However, the final layer was replaced from a Softmax to a Sigmoid. This change was done since our desired output is the individual probability of each of the possible labels, and not the combined probability of all the labels to appear. In other words, we want the net to give us for each of the 15 labels a probability from zero to one of the images having that label. This was due to our problem being a Multi-Variate classification problem.

Since the Sigmoid layer was used in the final layer of the model, we used Binary Cross Entropy criterion as the loss function. Adam was used as the optimizer and tried with different learning rates.

3.2 Inception/GoogLeNet

Inception's origins can be traced back to the publication "Going deeper with convolutions", and it was developed as a response to the "ImageNet Large-Scale Visual Recognition Challenge 2014"[6]. The results of the challenge proved that this architecture significantly outperformed what was considered State of the Art back then, winning the challenge. For this challenge, the network was submitted under the name of GoogLeNet, which has since become the way to refer to InceptionV1.

Even though the name inception comes from a meme[7], this architecture is nothing to joke around with. The main ideology behind this architecture is based on the authors' belief that the greatest progress in the fields has come from taking advantage of the synergy between bigger/deeper networks and classical computer vision algorithms. The goal was finding a way to approximate a local dense sparse structure in a convolutional network using readily available dense components. They achieved this by finding the optimal local construction and repeating it spatially [8].

The network itself is constructed at its core from convolutional building blocks. These convolution blocks are responsible for the feature extraction along the model and are composed of a ReLU layer, a Convolution layer, and a Batch Normalization layer which was first introduced in Inception V2.

The convolutional blocks are later combined to form Inception Modules as pictured in Figure 3. Citing from the paper “In general, an Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid. For technical reasons (memory efficiency during training), it seemed beneficial to start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion.”[8]. The original complete architecture can be observed in Table 1.

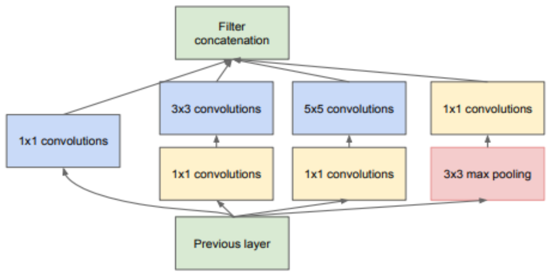
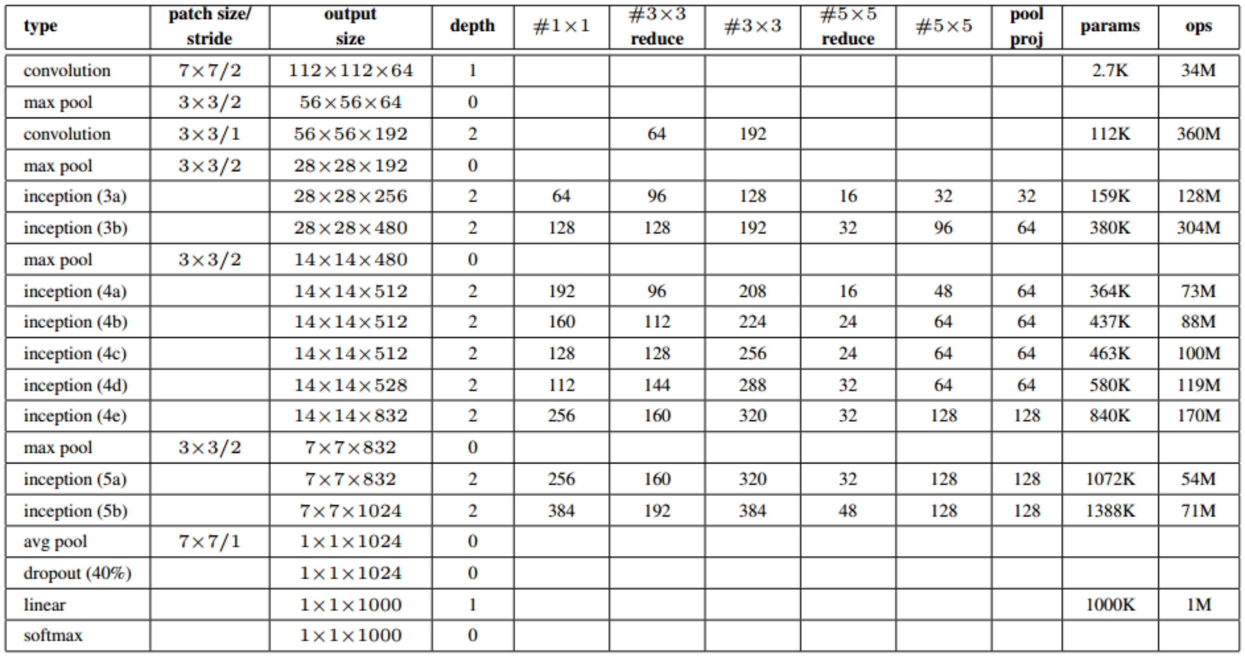


Figure 3: Inception Module Structure



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogleNet architecture

The benefit of this architecture is its ability to increase depth without necessarily increasing the computational cost. This is achieved through the inception module structure, by combining the different kernel sized convolutions it makes the network more complicated but less expensive to compute. Basically, computing each different convolution to keep the original dimensions (“same” convolution) and the Pooling with padding (unusual, but necessary to keep the dimensions), concatenating them together one can expand the total amount of channels while keeping the original dimensions. In order to avoid the cost of the higher magnitude convolutions, there is an intermediate 1x1 convolution with a fraction of the channels which is then iterated over with the intended convolution size and the corresponding channels. This 1x1 convolution is the key to reduce computational cost since computing it and the following bigger convolution requires less operations than just computing the larger convolution for the total amount of channels [9]. This convolution is often referred to as a “bottleneck” layer and is present in other models as well. As with most deep models, the drawbacks are the amount of information needed and the extensive training for them to be effective.

The original architecture has been worked upon and improved since its publication. Since 2014 we have since the introduction of InceptionV2, InceptionV3, InceptionV4. Every new iteration builds upon or modifies the original network in order to improve its performance on different aspects:

- V2 adds batch normalization to speed up training and add regularization.
- V3 modifies the inception blocks, introducing asymmetrical convolutions (1x7 and 7x1) as well as replacing the original 5x5 block with multiple 3x3. Overall, the blocks also got deeper.
- V4 incorporates the principles of ResNet [10], albeit with not as good results.

3.2.1 Implementation (adaptation and improvements)

For this model we decided to replicate its structure from scratch. This decision was driven by the desire of understanding each of its components better and the abundance of information for each parameter (such as Table 1). Initially we had settled upon implementing V1, however researching the later iterations we noticed that implementing V2 narrowed down to adding just a couple lines of code (the ones pertaining batch normalization). We decided to opt for implementing V2 because the batch normalization layers allow for uses of higher learning rates and set less emphasis in Dropouts and weight decay [11] .

The original net was designed to take inputs of size 224x224 and 3 channels. Since our pictures didn't meet these specifications, and in our desire to preserve the original network's architecture without losing information by starting with compressing, we included an Adaptative Max Pooling layer after the last Inception block in order to fix the input size to the last layers regardless of the image size. Additionally, leaving the number of channels as a variable in the net allowed us an extra degree of freedom. This had the negative side effect of increasing computational power needed and thus training time.

Finally, as we did with AlexNet, we replaced the network's last layer from a Softmax to a Sigmoid. Reiterating, this change was implemented to get the individual probabilities of each label instead of the normal "most probable" label outcome one gets from a Softmax.

As per the training, we fed the model small batches that fitted our computational power. Due to the addition of the Sigmoid layer in the model, we chose the Binary Cross Entropy (BCELoss) criterion as our loss function. Alternatively, we could have used the BCEwithLogits since it combines a Sigmoid and a BCELoss in a single class. However, we felt more comfortable with both elements separated due to our familiarity with them. A visual representation of our implementation can be examined in Figure 4.

Figure 4: Visualization of Inception Implementation

3.3 DenseNet

Another CNN architecture frequently discussed in academic literature pertaining to machine vision and object detection is the DenseNet architecture. The paper "Densely Connected Convolutional Networks" [12] published by Liu, Guang & Weinberger 2016 outlines the motivations and advantages of the DenseNet architecture. The architecture uses blocks of densely connected layers to address the vanishing gradient problem that often impedes Deep CNNs.

Similar to how residual blocks are the foundation of ResNet and Inception modules the foundation of Inception/GoogleNet; Dense blocks are the foundation of DenseNet. In dense blocks, each layer is directly connected to all proceeding layers in a feed forward fashion. This in turn reduces the complexity of the network. While a standard convolutional network with L layers will have L connections, DenseNet have L(L+1)/2 connections. Each dense block contains a prespecified number of layers inside of it, and among them feature maps are shared. Convolutions within dense blocks are composite functions that consists of three steps: batch normalization, ReLU, and 3x3 convolutions [12].

The output of a dense block is then fed into a transition layer which, again similar to ResNet and Inception/GoogleNet, uses the bottleneck concept (1x1 convolution layer) to allow for max pooling to reduce the size of feature maps. These transition layers, which are strategically positioned between adjacent dense blocks, reduce the feature map using 1x1 convolutions and 2x2 average pooling. This parallels the “bottleneck” concept presented in previous sections (3.2Inception/GoogLeNet). A simple visualization for this net can be observed in Figure 5.

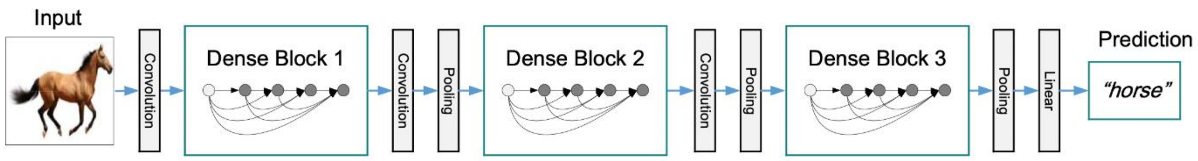
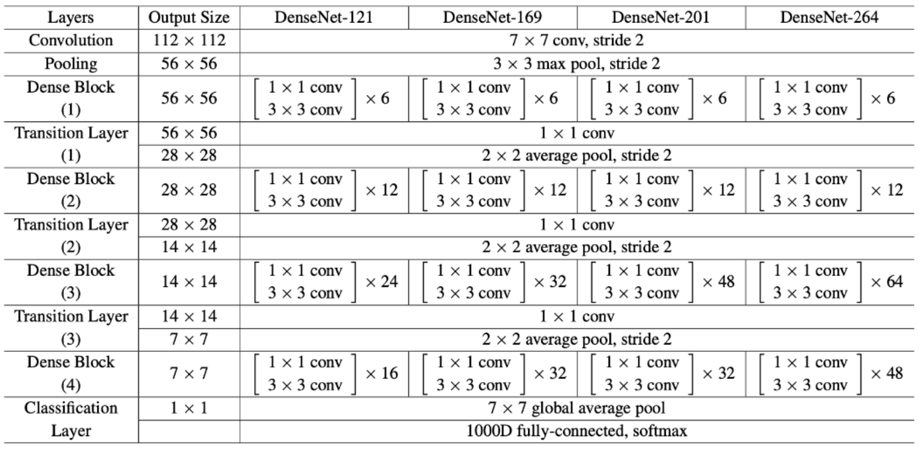


Figure 5: DenseNet conceptualization

In their paper, Huang present four versions of DenseNet: DenseNet-121, DenseNet-169, DenseNet-201, and DenseNet-264 with different preset number of convolution layers or feature maps within each dense block. The three-digit suffix indicates the total number of layers in each version. Note that the initial convolution and pooling layer, transition layers, and classification layer remain consistent in all versions. Figure 6 details the architectural comparison of each DenseNet model.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure 6: DenseNet architecture

The primary concern addressed by DenseNet is the vanishing gradient problem in convolutional networks as the depth of a network increases. DenseNet improves gradient propagation and classification by creating shorter paths from the original input to subsequent layers. Unlike ResNet, which uses summation to combine features, DenseNet leverages concatenation as summation can slow the flow gradient in the network during training. Dense blocks also have direct access to the gradient and loss of the original input, making it a deeply supervised network. Another advantage of DenseNet is the reduced number of parameters. As other models require additional parameters that tells what information to preserve, DenseNet retains the feature maps from previous layers with concatenation. DenseNet also has a fixed number of output feature maps per layer, so each layer adds only a limited number of parameters.

3.3.1 Implementation of DenseNet

We used DenseNet-121 as described in [12] for our implementation of DenseNet. An Adam optimizer was used as the classification layer with fixed learning rates of 0.01, 0.001, and 0.0001, and binary cross entropy used as our accuracy metric. As with the preceding models, we used the sigmoid as opposed to Softmax to obtain individual probabilities of each label.

4 Results

As we aren't able to replicate the training process for the original architectures we decided to make some concessions, mainly on the length of the training and the parameters we used. To be able to try different parameters and have comparable results we decided to set the number of epochs and try different learning rates to see its impact on the outcome. As per the optimizer used during the training process, we settled for Adam as in our understanding it's versatile enough to suit our problem. All the models were split 80/20 respectively in a train and test set. The models were only trained on 30 epochs due to computational limitations, and so that the results could be justifiably comparable. The batch sizes varied from model to model, again due to computational power limitations.

4.1 AlexNet

The network was trained on three different learning rates and a batch size of 32. The results from training can be observed in

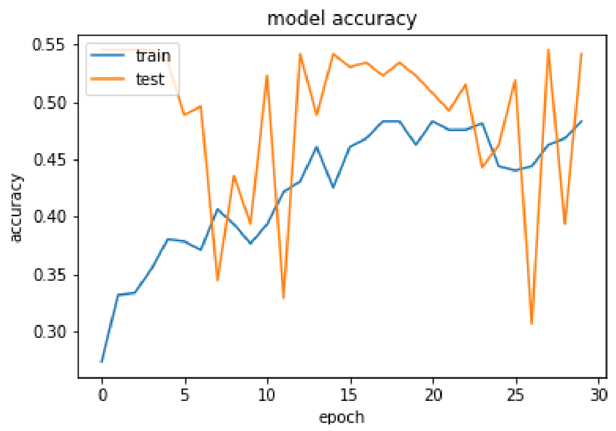
0.0001	**0.1364**	**0.6418**	**0.2732**	**0.4242**

Table 2.

LEARNING RATE	TRAINING LOSS	TRAINING ACCURACY	VALIDATION LOSS	VALIDATION ACCURACY
0.01	0.2480	0.4795	0.3063	0.5417
0.001	0.2419	0.4832	0.2738	0.5418
0.0001	0.1364	0.6418	0.2732	0.4242

Table 2 AlexNet training and validation results

There is no significant difference in performance between the Learning Rates. Looking at the results one can observe that the model with a learning rate of 0.001 had the highest performance as it had the highest validation accuracy of 54% and lowest validation loss of 0.27. A visualization of the accuracy and loss during training is shown in Figure 7.



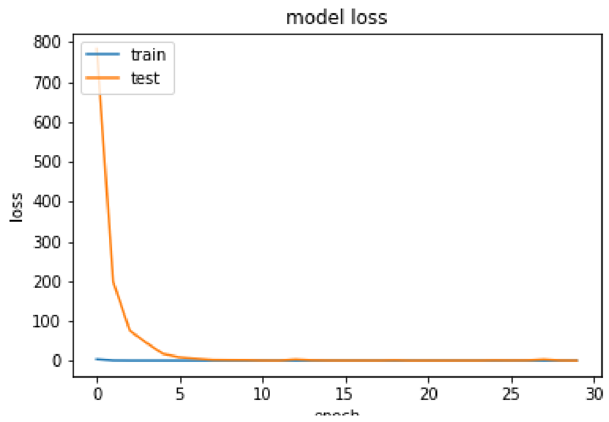


Figure 7: Learning Curves for AlexNet

Looking at the loss plot, there is no obvious sign of overfitting. However, the accuracy plot indicates a trend of increase in accuracy during training, while the validation accuracy has a high variability.

To get a better understanding of how well the model predicts on unseen data, ROC curves were plotted. This gave an indication of the tradeoff between the true positive rate and false positive rate of the predictive model. Figure 8 on the left shows and ROC curve that show a more generic understanding of the tradeoff between the classes, while Figure 8 on the right illustrates how well all the different classes were predicted individually.

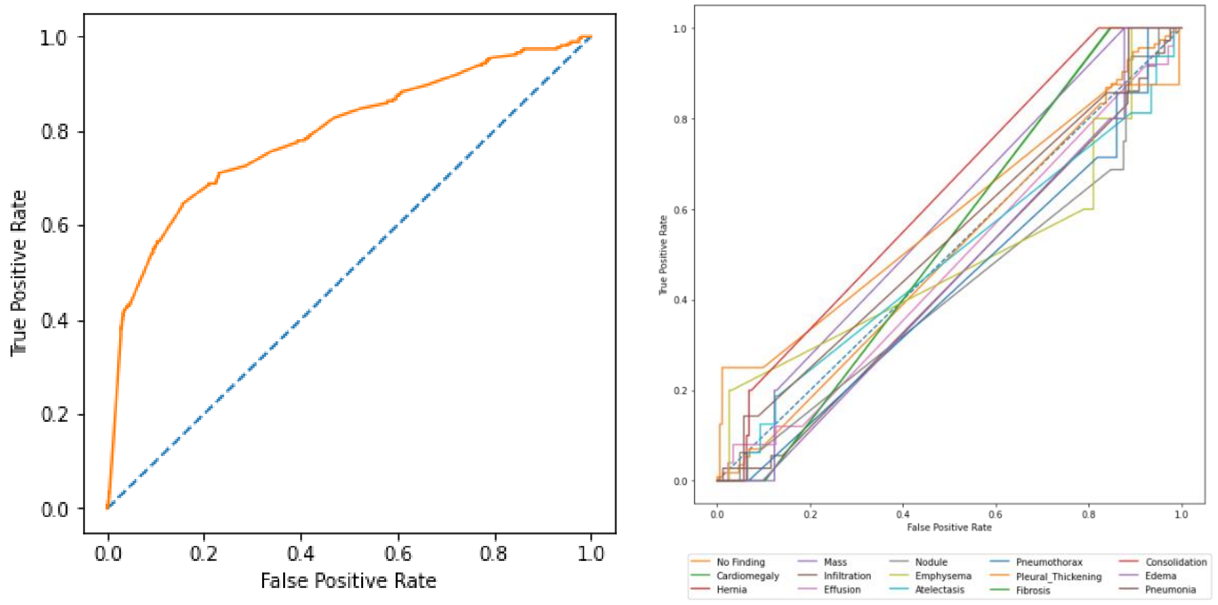


Figure 8: AlexNet ROC- curve visualization

Figure 8 on the left has an AUC of 0.79. Note that Figure 8 on the right shows all classes except for Hernias. There were as few as 227 images of Hernia in the entire X-Rays dataset, and therefore none of those ended up in the test dataset.

4.2 Inception

As stated, we decided to implement Inception V2, and since the selling point of this version is the reduced dependency towards the learning rate, we decided on making this our independent variable. We settled on training for 30 epochs as a compromise due to the amount of time we disposed of, since each epoch takes in average 30 minutes to complete. The learning rates were selected after running a short training cycle with only 20 images. Whenever applicable the decay rate was set to 0.1 with a patience of 0 (due to the short training).

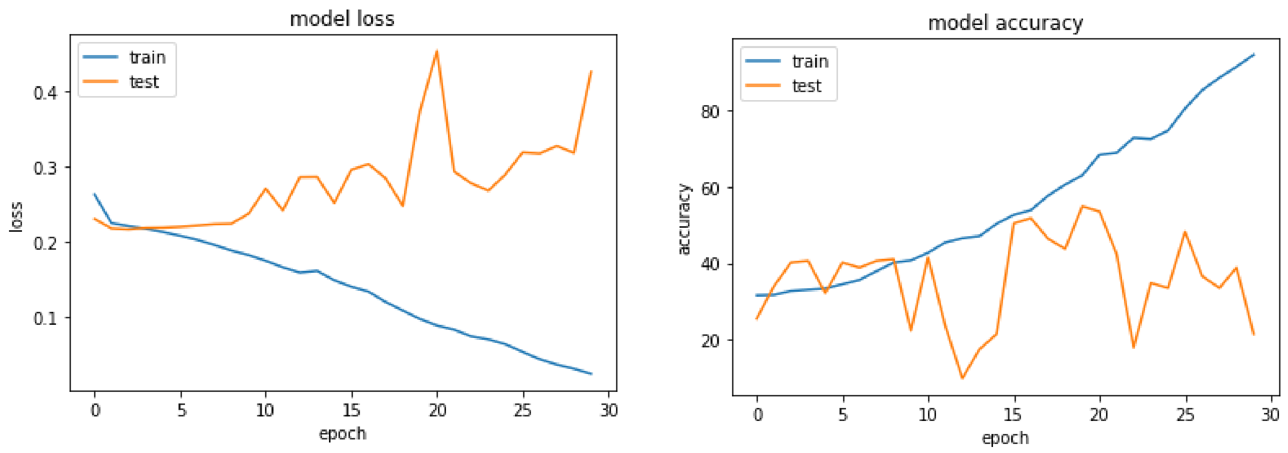


Figure 9: Learning curves for Inception

A summary of the results can be observed in Table 3. We decided to include both the final values and the best values since for most cases with static learning rate, the model started overfitting quite early, giving good results for the training data but decreasing results for the validation set. Figure 9 shows a visualization for the training process while with a static learning rate equal to 0.00005. Figure 10 depicts the ROC curve for the model at one of its best iterations, both as a binary for the whole prediction (evaluating each label as correct) as well as a detail for each of the labels.

DECAYING LEARNING RATE	LEARNING RATE	FINAL TRAINING LOSS	BEST TRAINING LOSS	FINAL TRAINING ACCURACY	BEST TRAINING ACCURACY	FINAL TEST LOSS	BEST TEST LOSS	FINAL TEST ACCURACY	BEST TEST ACCURACY
NO	0.001	0.157	0.157	0.467	0.473	0.399	0.244	0.366	0.406
YES	0.001	0.221	0.221	0.345	0.345	0.222	0.222	0.357	0.357
NO	0.0005	0.201	0.2	0.362	0.365	0.252	0.217	0.134	0.549
NO	0.0001	0.065	0.065	0.741	0.741	0.42	0.219	0.245	0.545
NO	0.00005	0.024	0.024	0.946	0.946	0.427	0.216	0.214	0.549
YES	0.00005	0.201	0.201	0.347	0.367	0.217	0.217	0.361	0.419

Table 3 Inception results

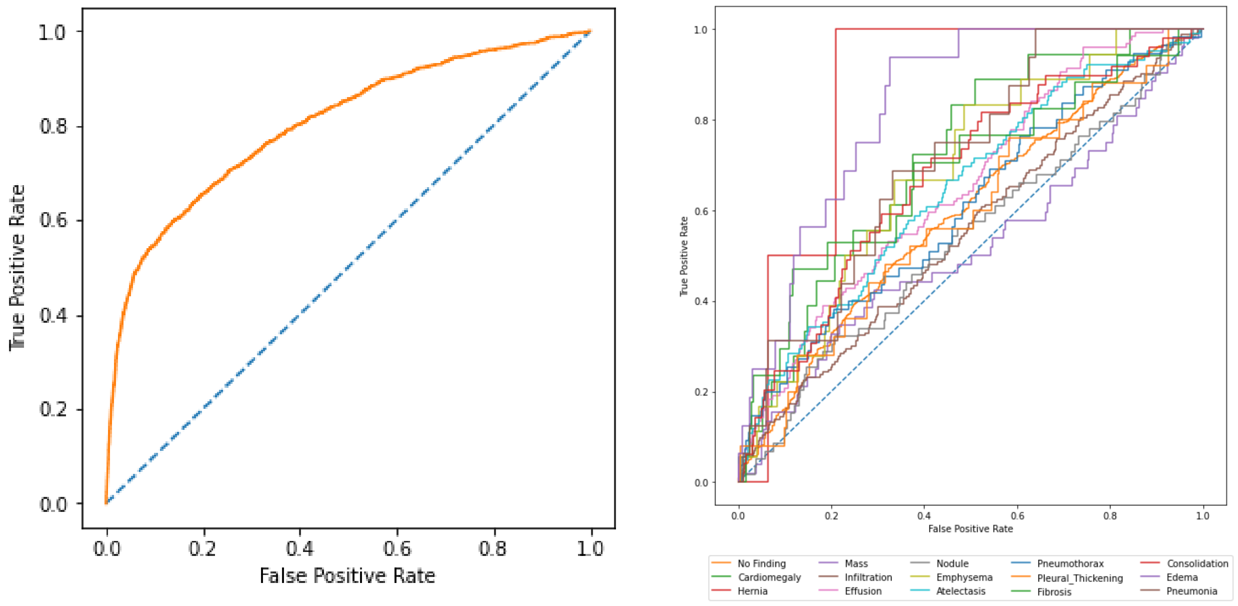


Figure 10: Inception's Roc Curves

4.3 DenseNet

In Table 4 shown below we see the DenseNet-121 training results with variable learning rates of 0.01, 0.001, and 0.0001.

LEARNING RATE	TRAINING LOSS	TRAINING ACCURACY	VALIDATION LOSS	VALIDATION ACCURACY
0.01	0.2129	0.439	0.3670	0.510
0.001	0.2264	0.450	0.3784	0.531
0.0001	0.1819	0.449	0.3822	0.412

Table 4 DenseNet's results

While we observe similar accuracy and loss for the larger learning rates, performance degradation occurs learning rate is set to 0.0001. The best performance was with a learning rate of 0.001. In Figure 11 below we show the accuracy and loss plots. While we do not observe overfitting, we do observe some abnormalities in accuracy and loss, likely due to us being restricted to a batch size of 4 for our implementation of DenseNet due to memory constraints.

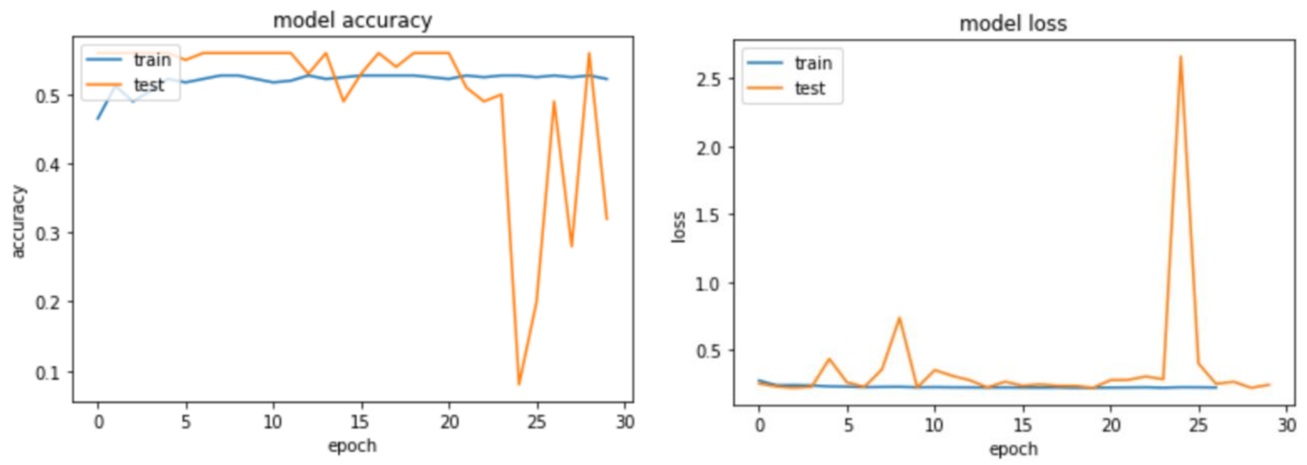


Figure 11: DenseNet's learning curves

Finally, we plot the ROC curves for the model overall, as well as by each individual class or disease, in Figure 12.

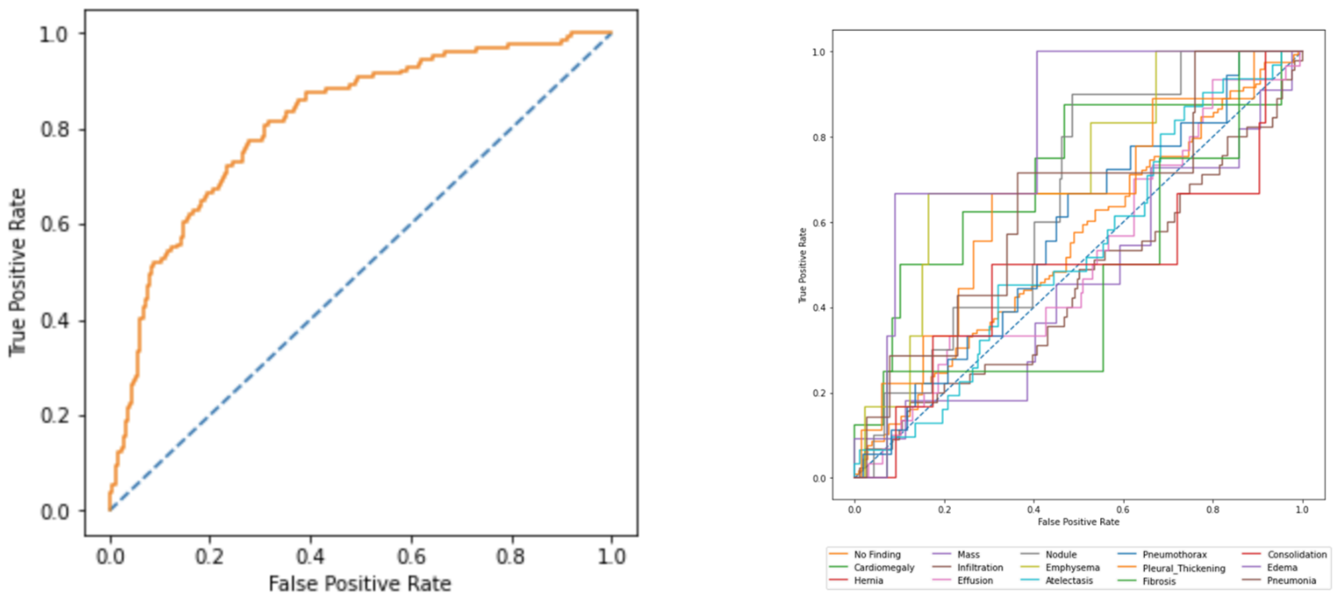


Figure 12 DenseNet's ROC Curves

5 Discussion

To give a bit of context and nuance to our results we feel the need to reiterate over a couple of issues that will be reflected throughout all of them. We tried replicating models that were developed by teams with more knowledge and better equipment, we had fun and learned a lot while doing it, but this definitely had an impact over the final results. The lack of computational power was probably the biggest issue we encountered though this challenge, making it so that we were not able to use the whole dataset and had long hours in front of our computer screens baby-sitting Google Colab. Not unexpected, but none of the models achieved an accuracy above 55%. However, all the models showed different trends when it came to training, which give us good material to talk about.

Looking at the results from AlexNet there is no sign of overfitting, but it does show that accuracy stops increasing and loss stops decreasing at around epoch 18/19. This could indicate that model stops training at some point due to there not being enough training samples to learn from. There is however, an upward trend in accuracy which signifies that the model learns progressively. The ROC-curve that shows label vs label had an AUC score of 0.79. This curve will automatically give a higher AUC than the one with all the labels and their individual ROC-curves since we are comparing each of the training labels against each of the testing labels, instead of assessing if the whole set of predicted labels was identical to the set of true labels. We decided to measure label vs label since we wanted to get an idea of the situation whether the model was over-labeling, under-labeling or just miss-labeling. From the ROC-curve, it shows that the model is best at predicting the classes Consolidation and Moss and worse at predicting the classes Nodule and Pneumothorax.

Under most circumstances the results from our implementation of Inception can be described as overfitting. This model was designed to handle very large amounts of images so exposing it to our small dataset resulted in heavy specific learning of the training dataset that didn't extrapolate to the whole set. This is the reason we decided to implement decaying learning rate only for this model (that and time constrains), since we believed that this could help with the issue. Overall, the results for both accuracy and loss are the lowest amongst all our models, but to our satisfaction using a low enough learning rate paired with an aggressive decay policy produced more stable results between training and validation sets. This improvement comes at the cost of reduced overall accuracy, which we believe could be solved with a larger dataset. As a curious result, it appears that when the model manages to have a good accuracy, it does best at predicting conditions that AlexNet is also good predicting. These results may come from the conditions being the ones that have the more easily identifiable feature, be a product of relying on parallel structures working simultaneously withing each model or just coincidental, thus we decided to classify it as curious.

Analyzing the results for DenseNet, we observe that it reaches one of the highest values of accuracy between all our models in very few Epochs. Simultaneously, its resulting loss sinks down to one of the lowest values equally as fast. However impressive these results might appear we are concerned about the stable tendency they present after these first training cycles. This tendency for the results to remain constant could be attributed to the way the Dense blocks work, and that by appending the resulting values from the top layers through the rest of the process our model is not able to learn over the initial iterations. One way these results could have been improved is to implement a decaying learning rate policy that is dependent on the validation loss. One positive aspect we can note from this model is its proclivity to not overfit, demonstrated by the values for both the training and validation set remaining similar. The DenseNet model performed better on certain classes, specifically Mass and Emphysema, while it struggled with Pleural Thickening.

Tying all the results together, it seems that AlexNet was the model that performed the best throughout this challenge. It was the only model that showed signs of gradual learning and achieved one of the highest accuracies of all the models, with no signs of overfitting. As mentioned in Section 3.1, AlexNet has been outperformed in tests and practical applications by other more complex networks, but due to the constrains we had during this project it appears that this model was the best choice.

Analyzing our results per label we see some similarities between the models regarding the labels they predict the best and worst. However, when we compare those results with the amount of labels the models were trained with, there is no clear correlation between amount and prediction accuracy, which leads us to believe that the conditions with the highest prediction accuracy along the models are the ones that have the most identifiable features, thus all our models are able to detect them correctly. Likewise, we can conclude the opposite happens with the labels with low predictive accuracy since there seems they seem to have a low correlation as well with the amount of labels. Finally, the most common in both the whole dataset and our sample is "No Finding", which all our models have a relative medium success at predicting. This result combined with the good AUC scores of all our model while comparing label vs label lead us to conclude that our models for the most part have a problem of miss-labeling (opposite to under or over-labeling), which we believe in most cases could be solved by an overall better training.

We would like to think that if we were to replicate this project under different conditions (better equipment) and with all the learning from this semester, we could achieve results more akin to the many comparative studies that have been published including these models; and we look forward to the opportunity to do so.

6 References

1. O'Shea, K. and R. Nash, *An introduction to convolutional neural networks.* arXiv preprint arXiv:1511.08458, 2015.
2. Education, I.C. *What are Convolutional Neural Networks*. 2020 [cited 2022 20.05]; Available from: https://www.ibm.com/cloud/learn/convolutional-neural-networks.
3. Center, N.I.o.H.-C. *CXR8*. 2017 [cited 20122; Available from: https://nihcc.app.box.com/v/ChestXray-NIHCC.
4. Wang, X., et al. *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*. 2017.
5. Pujara, A., *Concept of AlexNet:- Convolutional Neural Network*. 2020: medium.
6. ImageNet. *ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC2014)*. 2014 [cited 2022 05/05/2022]; Available from: https://www.image-net.org/challenges/LSVRC/2014/.
7. Meme, K.Y. *We Need To Go Deeper*. 2010 [cited 2022 05/05/2022]; Available from: https://knowyourmeme.com/memes/we-need-to-go-deeper.
8. Szegedy, C., et al. *Going deeper with convolutions*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
9. DeepLearningAI. *C4W2L06 Inception Network Motivation*. 2017 [cited 2022; Available from: https://www.youtube.com/watch?v=C86ZXvgpejM.
10. Szegedy, C., et al. *Inception-v4, inception-resnet and the impact of residual connections on learning*. in *Thirty-first AAAI conference on artificial intelligence*. 2017.
11. Ioffe, S. and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. in *International conference on machine learning*. 2015. PMLR.
12. Huang, G., et al. *Densely connected convolutional networks*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.