Predicting diseases in Chest X-Rays using Convolutional Neural Networks

Abstract

In this paper, we experiment with different architectures of deep neural networks to classify images in the ChestXray-14 dataset. After studying the results of these experiments, we finalized 2 architectures that worked best. We also experimented with data augmentation, different number and sizes of kernels, different number of convolutional, pooling and fully connected layers. Finally, we used two pre-trained CNN models to classify images. In addition, we visualize the filters obtained from training our models.

1 Introduction

The ChestXray-14 dataset [3] has 112,120 images of Chest X-rays and 14 diseases that are represented in these images. The original size of these images is 1024×1024 pixels. After loading the ChestXray-14 dataset, we prepare the dataset to separate the training and holdout data points. After shuffling the training points, we use 10% of them as our validation set, to determine which models give us the best performance. We play around with the number of convolutional, max-pool and fully connected layers. We also vary the number of filters and their kernel sizes. We experiment with Adam optimization, Xavier initialization and Batch Normalization, to estimate which combinations give us the best performance.

2 Related work

We implemented our two architectures, inspired by the existing Alexnet[1] and VGGnet[2] architectures. We built our first architecture based off of the simple Alexnet architecture. Alexnet architecture has 5 convolutional layers, 3 fully connected layers with 2 max pool layers, one at the end of the first two convolutional layers and the other at the end of the remaining two convolutional layers. Alexnet has stacked convolutional layers with different filter kernel size at each of the network such as 11 x 11, 8x8 and 5x5. Alexnet learns features of different sizes from an image at each level and thus we got inspired to try this model to see if we can get better performance with our chest X-ray image dataset.

For our second architecture, we got inspired from the VGGnet as it consistently learns a huge number of filters of same size. This means, the network can learn more specific and characteristic features for a particular image which would aid in better classification. We implemented a 11 layer network with 8 convolutional layers and 3 fully connected layers with 5 max pool layers. The convolutional layers in the middle of the network is stacked and pooled. We implemented these networks and made changes to this network to accommodate our computational constrains and our dataset.

We used VGGnet_19 with batch normalisation for our transfer learning model implementation with both features frozen and unfrozen. All our approaches and methods are discussed in the Section 3.

Preprint. Work in progress.

3 Methods

We implemented the baseline model, two of our own architectures and our transfer learning models with the suggested experimentation techniques. All the models implemented are described in the following subsections.

Labels	Diseases
0	No diseases
1	Atelectasis
2	Cardiomegaly
3	Effusion
4	Infiltration
5	Mass
6	Nodule
7	Pneumonia
8	Pneumothotax
9	Consolidation
10	Edema
11	Emphysema
12	Fibrosis
13	Pleural thickening
14	Hernia

Table 1: Labels and the diseases labelled by them

3.1 Baseline

The baseline model takes in a 512 $\times$ 512 pixel greyscale image and passes through a stack of three convolutional layers with 12, 10 and 8 output channels respectively. There are two fully connected layers stacked after the maxpooled convolutional output and acts as the classifier that uses the computed features and weights for classification. The filter kernel size used are 8 $\times$ 8 , 8 $\times$ 8 and 6 $\times$ 6 respectively for the three convolutional layers. The output features from the third convolutional layer is maxpooled with a kernel size of 2 $\times$ 2 and a stride of 3 which reduces the dimension by 3. The maxpooled output is flattened and as a result the input of the first fully connected layer was 164*164*8 in total from the maxpooled convolutional output with 128 output features. The second fc layer takes in 128 features and gives out 14 output features (no of classes to be classified) which are further activated using a sigmoid activation function. We are incorporating this activation function by using the multi label soft margin loss to get the multi-label probability to tag for all our 14 output classes. [1]

Layer	**In Channels**	**Out Channels**	**Kernel Size**	**Padding/Stride**	**Activation**
conv1	1	12	8×8	0/1	RRelu
conv2	12	10	8×8	0/1	RRelu
maxpool1			3×3	0/1	
conv3	10	12	8×8	0/1	RRelu
conv4	12	10	8×8	0/1	RRelu
conv5	10	8	8×8	0/1	RRelu
maxpool2			3×3	0/3	
fc1	0.0	512			RRelu
fc2	512	128			RRelu
fc3	128	14			

Table 2: Architecture 1 specifications

[1] Since `torch.nn.MultiLabelSoftMarginLoss` computes the sigmoid, we do not really implement a sigmoid in our final layer.

Layer	In Channels	Out Channels	Kernel Size	Padding/Stride	Activation
conv1	1	4	3×3	0/1	RRelu
maxpool1			3×3	0/1	
conv2	4	8	3×3	0/1	RRelu
maxpool2			3×3	0/1	
conv3	8	16	3×3	0/1	RRelu
conv4	16	16	8×8	0/1	RRelu
maxpool3			3×3	0/1	
conv5	16	8	3×3	0/1	RRelu
conv6	8	8	3×3	0/1	RRelu
maxpool4			3×3	0/3	
conv7	8	8	3×3	0/1	RRelu
conv8	8	8	3×3	0/1	RRelu
maxpool5			4×4	0/4	
fc1	122*122*8	512			RRelu
fc2	512	128			RRelu
fc3	128	14			

Table 3: Architecture 2 specifications

We expect our model to behave badly as we have not incorporated techniques to deal with class imbalance, variance in the input data and normalization which are much needed for the model to generalise over the dataset and perform better with the test data.

3.2 Experimentation

In order to build a better model than the baseline, we incorporated various techniques to help our model to generalise on the dataset. The first technique used is z scoring where we normalised each image by subtracting the mean of that image and scaling the image by its standard deviation. This was done so that the inputs to the neural networks are on the same range and the network need not waste time in learning very large or very small weights. The second technique used was data augmentation, in order to accommodate our computational constraints, we resized our images to 256 $\times$ 256 and also did a random horizontal flip with a probability of 0.5. This is done to generalize over the positional variance. The third technique we incorporated is to fix the class imbalance problem which would drive the network to just generalise on one highly represented class than the other. We used a weighted loss function to prevent the network from over-fitting to one particular class. The weighted loss penalizes misclassifications involving a less frequent class more than one involving a more frequent class. Further, we also tried using Dropout to the fully connected layers of our best performing model (inspired by Alexnet - arch1) with a probability of 0.5. However, we witnessed a decrease in the performance of the model itself showing that it is underfitting to the dataset. Thus we removed the dropout and reported the results just for our vanilla arch1.

The specifications for the two architectures built by us inspired by Alexnet and Vggnet as discussed in the related works section are summarised and shown in the Table 2 and Table 3 respectively. The results are shown and discussed in the later sections of this report. We implemented all the experimentation discussed earlier to get a better performing model.

Furthermore, we implemented ensembling using the two architectures that we built. We implemented a (voting/averaging outputs) ensembling to use the capabilities of both the models to better perform with our dataset. The results are reported and discussed in the later sections.

3.3 Transfer learning

We implemented a model using the transfer learning technique which is essentially reusing the features a model learnt by training on the Imagenet dataset leveraging the fact that the network is trained enough to recognize a set of common features across images such as edges and other patterns. In this case, We assume that Our chest X-Ray 14 images come from the same distribution as the Imagenet dataset. We did the two types of implementation we were asked to do. Firstly, we used a Vgg19 network with batch normalisation pre-trained on Imagenet. We froze parameters of all

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	98.5	93.2	86.5	83.3	90.4	95.8	98.6	97.3	91.6	94.6	91.3	96.0	99.4	97.1	97.4
1	0.1	4.5	0.0	0.2	0.2	0.1	0.0	0.0	0.0	0.1	0.0	0.8	0.3	0.3	0.0
2	0.0	0.0	11.0	0.0	0.0	0.0	0.0	0.4	0.2	0.1	0.0	0.0	0.0	0.0	0.0
3	0.5	1.3	0.4	14.6	0.6	0.7	0.8	0.4	1.1	0.6	0.2	0.2	0.3	1.7	2.6
4	0.7	0.8	1.8	1.7	8.8	1.3	0.3	1.9	0.5	2.7	5.2	0.2	0.0	0.5	0.0
5	0.0	0.0	0.0	0.0	0.0	1.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.1	0.0	0.0	0.0	0.2	0.1	0.0	6.6	0.4	0.0	0.0	0.0	0.3	0.0
9	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	1.5	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.2	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.9	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 4: Confusion matrix for the baseline model - The columns denote true labels and the rows denote the predicted labels (Labels to diseases table can be found in Table 1).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	98.8	97.7	98.2	90.5	97.7	97.6	98.4	96.3	98.6	96.5	96.4	98.8	99.0	97.5	90.0
1	0.1	0.2	0.0	0.2	0.0	0.2	0.1	0.0	0.1	0.2	0.2	0.0	0.0	0.0	0.0
2	0.0	0.0	0.8	0.0	0.1	0.0	0.0	0.4	0.1	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	2.1	1.0	9.2	1.7	2.2	1.5	2.9	1.1	3.0	2.7	1.2	1.0	2.5	5.0
4	0.1	0.0	0.0	0.1	0.5	0.0	0.0	0.4	0.1	0.2	0.7	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0

Table 5: Confusion Matrix for Architecture 1 - The columns denote true labels and the rows denote the predicted labels (Labels to diseases table can be found in Table 1).

the layers except that of the last custom fully connected layer with 14 outputs. The network was trained and optimised only on the last layer with all the other parameters and the features learnt intact. Basically, it means that we are updating only the features in the last fully connected layer.

Secondly, we fine tune the pre-trained model features to our dataset to see if the model adapted to classifying our dataset better. We experimented with Resnet34 model pre-trained on Imagenet for this task. We unfroze parameters of all the layers and trained our network. We optimized on parameters from all the layers and recorded our results.

4 Results

The per-class accuracy, precision and recall values are shown for all models in Tables10,11, 12 and 13 respectively, while 14 shows the weighted metrics for the five models.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	99.7	99.7	99.8	99.5	99.6	99.6	99.6	99.2	99.9	100.0	99.8	99.8	100.0	99.3	100.0
1	0.1	0.1	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.1	0.1	0.2	0.3	0.1	0.2	0.3	0.4	0.1	0.0	0.0	0.0	0.0	0.4	0.0
4	0.1	0.0	0.0	0.0	0.2	0.0	0.0	0.4	0.0	0.0	0.2	0.2	0.0	0.2	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.1	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6: Confusion matrix for Architecture 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	98.6	97.4	98.0	92.0	96.1	97.3	98.8	98.8	96.7	96.2	98.0	97.3	98.7	98.6	100.0
1	0.2	1.1	0.2	0.3	0.2	0.2	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.7	1.0	1.0	6.7	0.8	1.6	0.9	0.8	1.7	1.2	0.5	1.0	0.6	1.1	0.0
4	0.5	0.5	0.8	1.0	2.9	0.9	0.3	0.4	1.1	2.4	1.5	1.5	0.6	0.3	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.2	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 7: Confusion matrix for the pre-trained and all layers frozen VGG-19 batch normalisation model - The columns denote true labels and the rows denote the predicted labels (Labels to diseases table can be found in Table 1).

4.1 Baseline model

The training and validation losses for the baseline model are shown in Figure 1. The model stopped training after 4 epochs due to increasing validation costs. In addition, we see that the training costs do not reduce a lot after the first epoch. In spite of having very high per-class accuracy, the baseline still does not take into consideration of various factors such as class imbalance and thus could effectively perform bad on more unseen data as the underlying characteristics and the distribution of the dataset is not taken into consideration. However, among all the implemented models, we found that baseline performed better on seeing its high BCR. The confusion matrix for this model is shown in Table 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	95.6	90.2	89.2	70.9	95.2	70.6	90.6	96.1	91.6	91.1	95.5	88.4	95.5	89.9	42.1
1	0.4	3.8	0.2	0.2	0.2	0.1	0.3	0.0	0.1	0.5	0.0	0.0	0.6	0.2	0.0
2	0.2	0.0	8.8	0.1	0.2	0.1	0.1	0.0	0.1	0.0	0.0	0.2	0.0	0.0	2.6
3	1.7	3.7	1.2	26.7	1.7	1.4	1.3	1.9	2.7	4.9	1.2	0.2	2.5	6.2	2.6
4	0.2	0.0	0.0	0.2	1.2	0.1	0.3	0.0	0.2	1.0	2.5	0.2	0.6	0.0	0.0
5	1.6	2.1	0.4	1.8	1.4	26.6	5.7	1.9	2.3	2.4	0.7	1.0	0.3	3.8	2.6
6	0.1	0.0	0.0	0.0	0.0	0.8	1.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.0	2.2	0.0	0.0	1.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.6	0.0	0.0	9.1	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.3	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.2	0.0	0.1	0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	50.0

Table 8: Confusion matrix for the pre-trained and all layers unfrozen Resnet34 model - The columns denote true labels and the rows denote the predicted labels (Labels to diseases table can be found in Table 1).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	99.7	99.7	100.0	99.8	99.8	99.8	99.8	99.6	99.5	99.9	99.5	99.2	99.7	99.4	100.0
1	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.3	0.2	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.1	0.1	0.0	0.1	0.1	0.1	0.2	0.4	0.2	0.0	0.5	0.2	0.0	0.2	0.0
4	0.1	0.0	0.0	0.1	0.1	0.1	0.0	0.0	0.3	0.0	0.0	0.4	0.0	0.2	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.2	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 9: Confusion matrix for the ensembling of Arch1 and Arch2

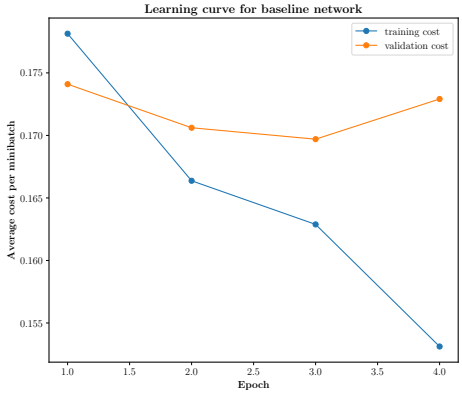


Figure 1: Training and Validation losses for the baseline model

4.2 Architecture 1 - Alexnet inspired

Contrary to our expectations, the Alexnet inspired model (Arch1) does a poorer job than the baseline, in spite of incorporating features such as Z-scoring and class imbalance. The training and validation costs for this model are plotted in Figure 2. The model did not train beyond three epochs due to increasing validation cost, and the training loss does not significantly reduce after the first epoch. Though this model performed worse than the baseline, it performed better than the second architecture. The confusion matrix for this architecture is shown in Table 5

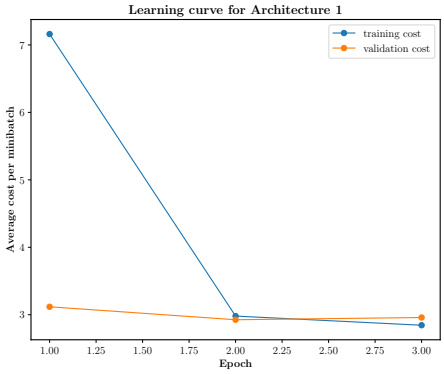


Figure 2: Training and Validation losses for Architecture 1

4.3 Architecture 2 - Vggnet inspired

This model does worse than the other two models, in spite of the increase in the number of convolutional layers. We see that the training stops after 4 epochs in this case too. The costs are plotted in Figure 3, and the confusion matrix is shown in Table 3

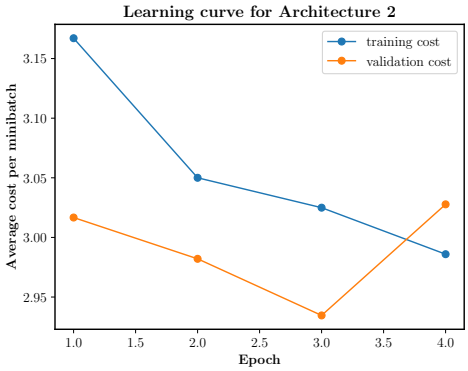


Figure 3: Training and Validation losses for the Architecture 2

4.4 Transfer Learning 1- Vgg19 with Batch normalisation

The training occurred only for two epochs. The confusion matrix is shown in Table 7 and the costs are shown in Figure 4

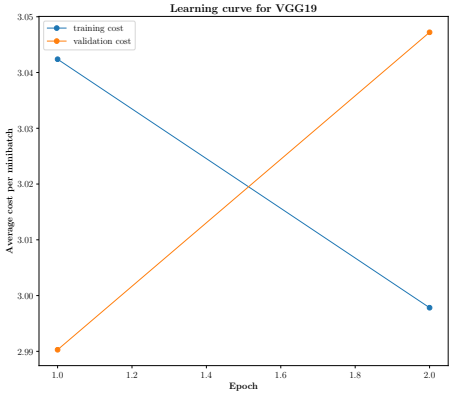


Figure 4: Training and Validation losses for Transfer Learning 1

4.5 Transfer Learning 2 - resnet34 with Batch normalisation

The training occurred only for two epochs. The confusion matrix is shown in Table 8, and the costs are shown in Figure 5. We see that this networks performs really well for the number of epochs it trained, which could be because in addition to pre-training, we are also optimizing the weights of the network further.

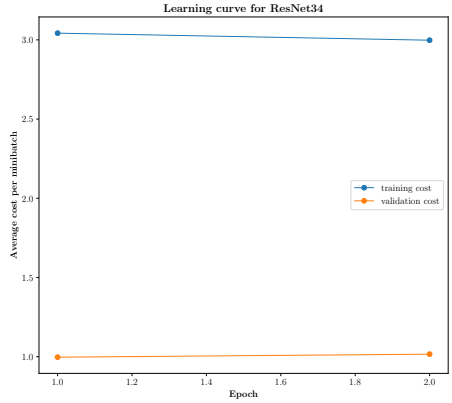


Figure 5: Training and Validation losses for Transfer Learning 2

Disease	Baseline	Arch1	Arch2	Ensemble	Transfer1	Transfer2
0	0.471	0.442	0.427	0.424	0.442	0.488
1	0.922	0.919	0.919	0.919	0.919	0.921
2	0.982	0.979	0.980	0.981	0.981	0.982
3	0.913	0.903	0.907	0.903	0.903	0.912
4	0.865	0.857	0.856	0.860	0.859	0.862
5	0.962	0.961	0.961	0.961	0.961	0.954
6	0.955	0.954	0.954	0.955	0.955	0.955
7	0.990	0.990	0.990	0.990	0.990	0.990
8	0.963	0.962	0.961	0.961	0.960	0.961
9	0.968	0.966	0.966	0.968	0.968	0.968
10	0.985	0.982	0.982	0.984	0.984	0.984
11	0.980	0.980	0.981	0.980	0.980	0.981
12	0.988	0.988	0.988	0.988	0.988	0.988
13	0.975	0.977	0.978	0.975	0.975	0.976
14	0.999	0.998	0.998	0.998	0.998	0.999

Table 10: Accuracy results for the five architectures

Disease	Baseline	Arch1	Arch2	Ensemble	Transfer1	Transfer2
0	0.449	0.434	0.426	0.424	0.432	0.457
1	0.732	0.178	0.2	0.25	0.379	0.538
2	0.843	0.4	0	0	0	0.551
3	0.693	0.392	0.226	0.1	0.450	0.571
4	0.588	0.528	0.214	0.08	0.395	0.443
5	0.791	0	0	0	0	0.362
6	0.333	0	0	0	0	0.419
7	0	0	0	0	0	0
8	0.795	0	0	0	0.6	0.595
9	0.8	0	0	0	0	0
10	0.867	0	0	0	0	0
11	0.833	0	0	0	0.25	0.723
12	0	0	0	0	0	0.167
13	0	0	0	0	0	0
14	0	1	0	0	0	0.613

Table 11: Precision results for the five architectures

Disease	Baseline	Arch1	Arch2	Ensemble	Transfer1	Transfer2
0	0.984	0.988	0.997	0.997	0.986	0.956
1	0.045	0.002	0	0.001	0.011	0.038
2	0.101	0.007	0	0	0	0.088
3	0.146	0.092	0.002	0.001	0.067	0.267
4	0.088	0.005	0.001	0	0.029	0.012
5	0.019	0	0	0	0	0.266
6	0	0	0	0	0	0.015
7	0	0	0	0	0	0
8	0.066	0	0	0	0.003	0.022
9	0.014	0	0	0	0	0
10	0.032	0	0	0	0	0
11	0.029	0	0	0	0.002	0.091
12	0	0	0	0	0	0.003
13	0	0	0	0	0	0
14	0	0.05	0	0	0	0.5

Table 12: Recall results for the five architectures and the ensemble

Metric	Baseline	Arch1	Arch2	Ensemble	Transfer1	Transfer2
Precision	0.549	0.320	0.249	0.221	0.340	0.438
Recall	0.457	0.431	0.426	0.425	0.432	0.453
BCR	0.503	0.375	0.338	0.323	0.386	0.445

Table 14: Weighted metrics for the five networks and the ensemble

Disease	Baseline	Arch1	Arch2	Ensemble	Transfer1	Transfer2
0	0.716	0.711	0.712	0.711	0.709	0.707
1	0.389	0.091	0.100	0.126	0.195	0.288
2	0.477	0.204	0	0	0	0.320
3	0.419	0.242	0.108	0.051	0.258	0.419
4	0.338	0.266	0	0.040	0.212	0.228
5	0.405	0	0	0	0	0.314
6	0.167	0	0	0	0	0.217
7	0	0	0	0	0	0
8	0.430	0	0	0	0.301	0.328
9	0.407	0	0	0	0	0
10	0.449	0	0	0	0	0
11	0.431	0	0	0	0.126	0.407
12	0.029	0	0	0	0	0.085
13	0	0	0	0	0	0
14	0	0.525	0	0	0	0.557

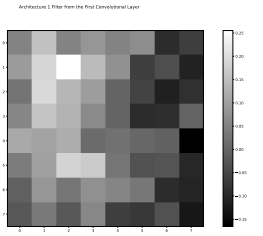
Table 13: Balanced Classification results for the five architectures and the ensemble

4.6 Ensembling

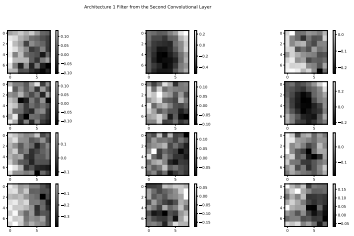
We tried to ensemble the outputs from the two architectures we had built (Arch1 and Arch2) to investigate whether ensembling improves our results. We find that our model does not improve it to our expectations, which could be because of the voting strategy we used (which essentially takes the OR of the two result classifier arrays). We feel this could have increased our chances of mis-classification. However, the ensembling performs as well as our two networks in spite of this weakness.

4.7 Filter visualization

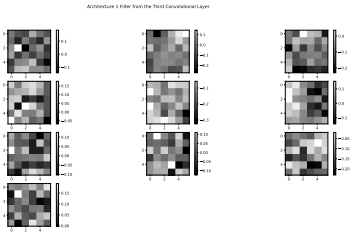
The filters for the baseline model as it was the best performing model are shown in Figures 6a, 6b, 6c and 6d for the first, second and for two weights third convolutional layers respectively. The filters represent the patterns that could give out the maximal activation. Since we are visualising the baseline filters, The filters in the first convolutional layer acts directly on the pixels and small 8×8 patch on the image which has bright pixel blobs which resembles an edge pattern. With increase in depth, the effective receptive field of the filters is increased and can detect much bigger patterns in an image which are more non linear.



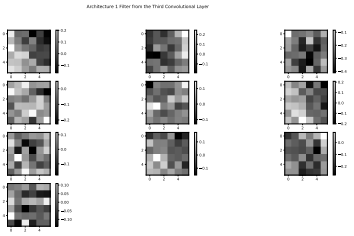
(a) A filter from the first layer



(b) A filter from the second layer



(c) A filter from the third layer



(d) A filter from the third layer

Figure 6: Visualization of the convolutional filters of the baseline model

5 Discussion

We trained five networks on the NIHCC Chest X-ray dataset. The first of the five was designated the "baseline" network which, as expected performed poorly in distinguishing the diseases, especially the rare ones, in spite of exceptionally high accuracy. We identified this to not only arise from the network's inherent inability due to its shallowness, but also from the fact that class imbalances have not been taken into account. To counter this, we created two networks (named Architecture 1 and Architecture 2) inspired by AlexNet and VGG-Net respectiely. We found that these networks performed worse than the baseline. In addition, we tried ensembling of the two networks and found that it does not perform better than either of the two networks. Finally we used networks pre-trained on ImageNet, replaced the last layer with a 14-category sigmoid, and trained the last layer with and without freezing other parameters. We still did not find any significant improvement to the baseline network.

We think that our performance between our two architectures differ due to the differences in size, which could result in under-fitting of the second network compared to the first, and the first compared to the baseline. Our model constantly predicted the no disease class for most of the diseases, and this did not vary significantly between the architectures. The performance metrics (precision, recall and accuracy) for the networks are more or less the same. We feel recall and precision have more ability to distinguish a network's performance, especially on the rare classes, recall being the ability of the network to identify the existence of the disease when provided an input of images containing the disease, and precision being the ability of the network to correctly classify the existence of a disease, given that it is reporting that the disease is present. A network that constantly predicts "no disease" will have nearly 100% accuracy to predict a rare disease, but very low values of precision and recall. Hence good values of precision and recall implies an increased ability of the network to correctly identify the rare diseases

We think the under-training of the networks could be attributed to the follwoing reasons

- A ceiling on the size of the network due to hardware constraints. The original VGGNet, which inspired us to do our second architecture, has at least 16 channels in all of its convolutional layers, going all the way up to 512 channels. In contrast, our network had a maximum of 8 channels. We were severely constrained by hardware to downsize our network to this level. In addition, we could not increase the depth of our network due to

similar issues. We feel this is a important contributor to our low metrics, in spite of having a lot of training images.

- Noisiness and lack of uniformity in the training set. We also think the inherent noise in the labelling and the lack of uniformity in the "format" (i.e., shape and size) of the chest X-rays, coupled with lack of images to train the network to learn these artifacts, could have also contributed to the effect

References

[1] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[3] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *2017 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, pages 3462–3471, 2017.