Chest Xray Disease Classification

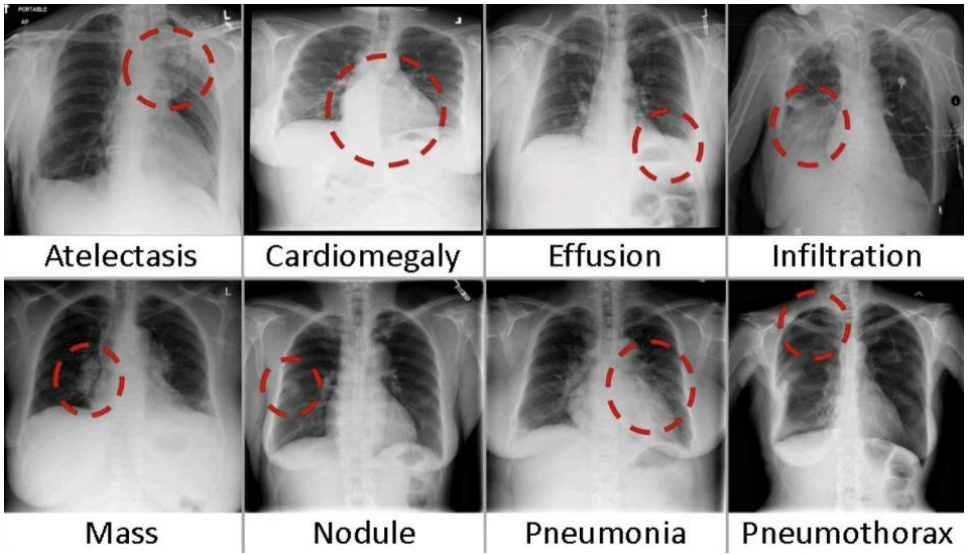
By: Vernon, Chia-Yu & Alif

Problem Statement

- Multi-Label Classification on the ChestX-ray14 Dataset.
- Given a Xray image, predict if the Xray image has any diseases.

Dataset (ChestX-ray14)

- Comprises of 112,120 frontal-view X-ray images of 30,805 unique patients.
- Total of 14 disease labels.



Existing Works: CheXNet

- Achieved state of the art results on all 14 diseases. (ChestX-ray14)
- Used existing 121-layer Dense Convolution Network (DenseNet) pretrained on ImageNet.
- Also trained on multi-label classification.

Pathology	Wang et al. (2017)	Yao et al. (2017)	CheXNet (ours)
Atelectasis	0.716	0.772	**0.8094**
Cardiomegaly	0.807	0.904	**0.9248**
Effusion	0.784	0.859	**0.8638**
Infiltration	0.609	0.695	**0.7345**
Mass	0.706	0.792	**0.8676**
Nodule	0.671	0.717	**0.7802**
Pneumonia	0.633	0.713	**0.7680**
Pneumothorax	0.806	0.841	**0.8887**
Consolidation	0.708	0.788	**0.7901**
Edema	0.835	0.882	**0.8878**
Emphysema	0.815	0.829	**0.9371**
Fibrosis	0.769	0.767	**0.8047**
Pleural Thickening	0.708	0.765	**0.8062**
Hernia	0.767	0.914	**0.9164**

Figure 2: Taken from CheXNet paper

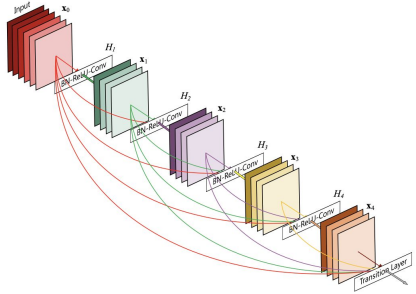


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Models

- Naive CNN
- ResNet
- DenseNet
- Unet Classifier
- Vision transformer

Naive Convolutional Network

6 consecutive Convolution block.

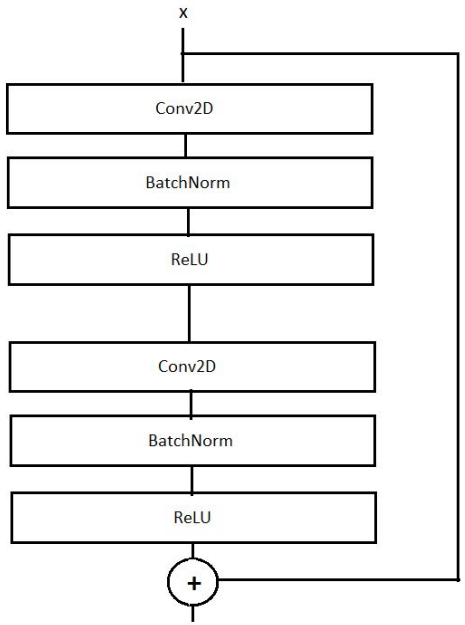
Each block consist of a 3x3 kernel size of 16 filters with same padding, followed by BatchNorm, ReLU activation and 2x2 max pooling. After the 6 blocks, it is flattened and connected to a fully connected layer.

```python
class ConvolutionBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(16,16,kernel_size=(3,3),padding='same')
        self.batchnorm = nn.BatchNorm2d(16)
        self.activation = nn.ReLU()
        self.pooling = nn.MaxPool2d(kernel_size=(2,2))

    def forward(self,inp):
        x = self.conv(inp)
        x = self.batchnorm(x)
        x = self.activation(x)
        x = x + inp ## Skip connection for each block
        x = self.pooling(x)
        return x


class NaiveConvolutionNetwork(nn.Module):
    def __init__(self,n_blocks=6):
        super().__init__()
        self.blocks = [nn.Conv2d(3,16,kernel_size=(3,3))]+[ConvolutionBlock() for _ in range(n_blocks)]
        self.layers = nn.Sequential(*self.blocks)
        self.pooling = nn.MaxPool2d(kernel_size=(2,2))
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(144,28)
        self.fc2 = nn.Linear(28,14)

    def forward(self,x):
        x = self.layers(x)
        x = self.flatten(x)
        x = nn.ReLU()(self.fc(x))
        x = self.fc2(x)
        return x
```

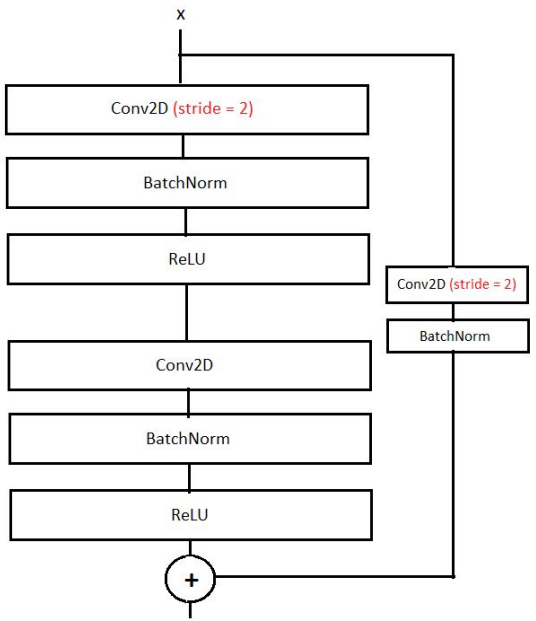
ResNet (Residual Network)

ResNet: A model architecture that leverages “skip-connections” to mitigate vanishing gradients & allow deeper models.

Our ResNet uses Residual Blocks which allow for the option of downsampling inputs, so that our final few layers (especially the linear layer) will not have to deal with too many features.



Residual Block
(no downsample)



Residual Block
(with downsample)

ResNet (Residual Network)

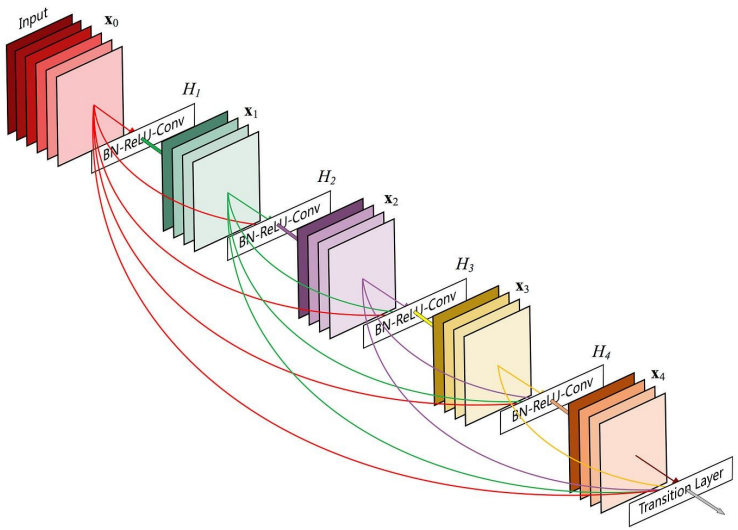
ResNet: A model architecture that leverages “skip-connections” to mitigate vanishing gradients & allow deeper models.

Our ResNet implementation comprises of a collection of such ResidualBlocks as well as other layers, as seen in the code cell ->

```
class ResNet(nn.Module):
    def __init__(self, in_channels=3, outputs=14):
        super(ResNet, self).__init__()

        self.layer0 = nn.Sequential(
            nn.Conv2d(in_channels, 48, kernel_size=7, stride=2, padding=3),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(48),
            nn.ReLU()
        )

        self.layer1 = nn.Sequential(
            ResidualBlock(48, 48, downsample=False),
            ResidualBlock(48, 48, downsample=False)
        )

        self.layer2 = nn.Sequential(
            ResidualBlock(48, 96, downsample=True),
            ResidualBlock(96, 96, downsample=False)
        )

        self.layer3 = nn.Sequential(
            ResidualBlock(96, 192, downsample=True),
            ResidualBlock(192, 192, downsample=False)
        )

        self.layer4 = nn.Sequential(
            ResidualBlock(192, 384, downsample=True),
            ResidualBlock(384, 384, downsample=False)
        )

        self.gap = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(384, outputs)
```

DenseNet

DenseNet: A model architecture that leverages ‘dense’ connections between layers to support efficient gradient propagation and feature reuse.

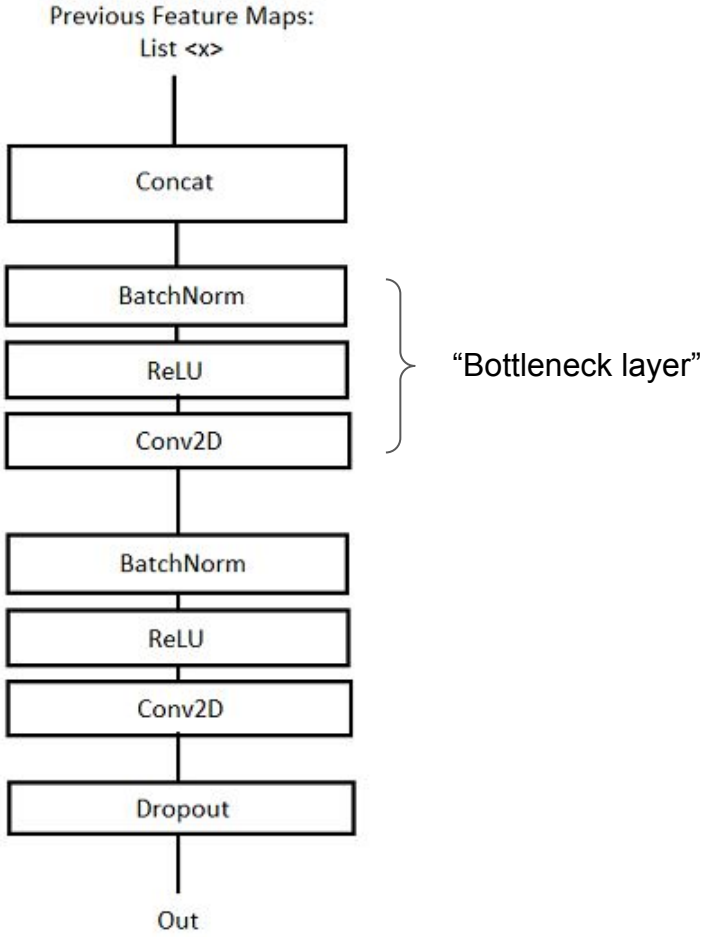


DenseNet

DenseLayer:

1) Concatenates the feature maps it receives from prev layers

2) Passes the feature maps through the depicted sequence of layers



DenseNet

DenseBlock:

Contains a number of *DenseLayers*, where each layer receives the output feature maps of all preceding layers (within the DenseBlock) as its input.

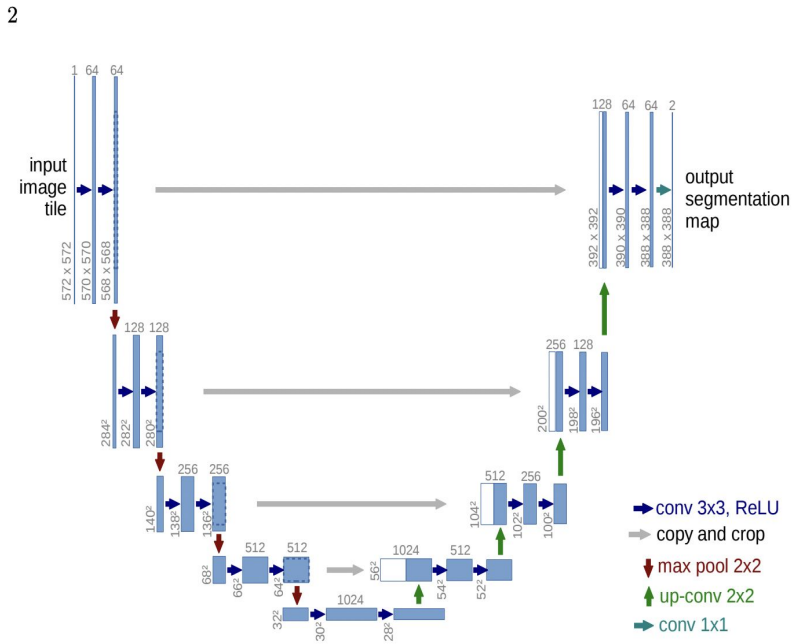
*Contains logic for directing feature maps from prev layers into subsequent ones.

DenseNet:

Our DenseNet model consists of 4 *DenseBlocks*, containing 4, 8, 16, and 12 *DenseLayers* each. All these DenseLayers share the growth rate of 24, and a batch norm size of 4.

UNet Classifier

Unet has an encoder-decoder architecture, originally meant for image segmentation. The encoder part is responsible for feature extraction and the decoder part is responsible for generation segmentation mask. We utilize the encoder part of the Unet and built a classifier by adding a classification head on the Unet encoder.



XvT (Xray Vision Transformer)

- Our convolution transformer encoder consists of 6 layers.
- For our experiments, we set number of attention heads = 6, patch size and stride = 14, embed dim = 192, and projection kernel size = 3.

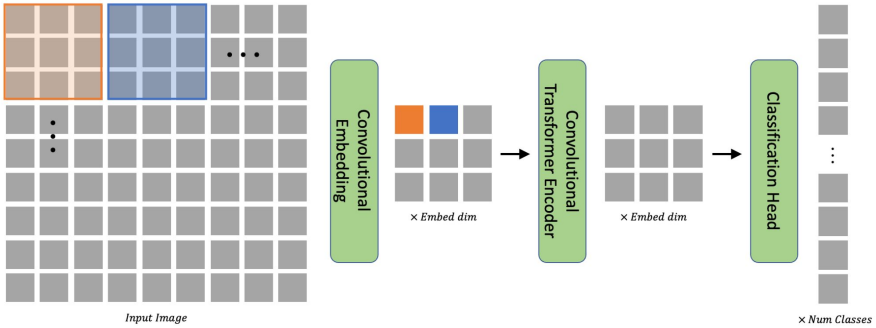


Figure 1: XvT Model Architecture

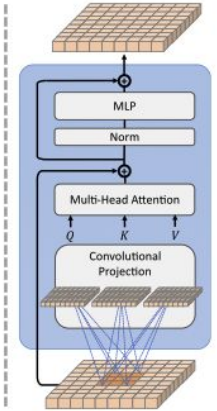


Figure 2: Transformer layer

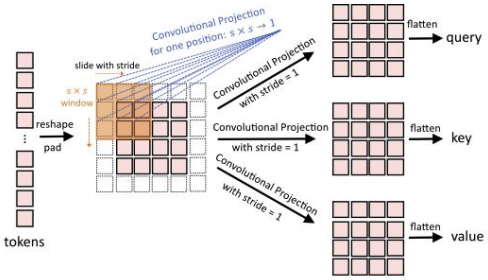
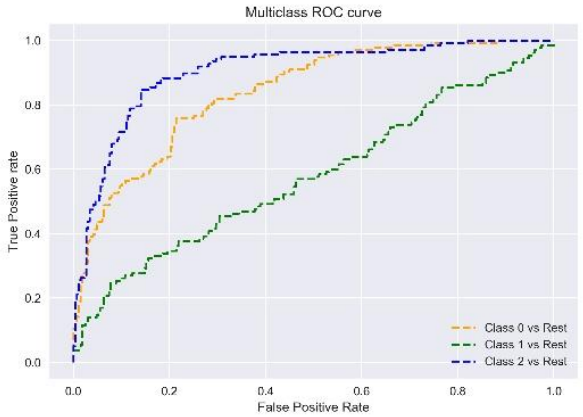


Figure 3: Convolutional Projection

Performance metrics

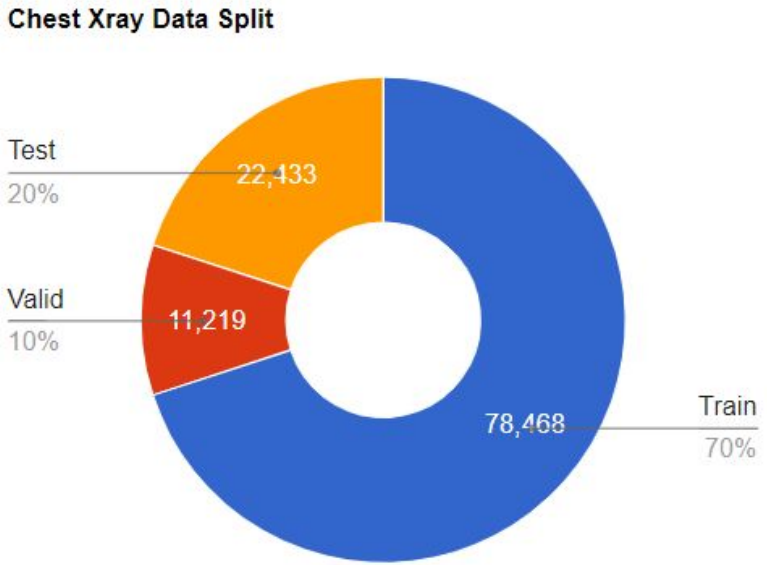
We compute the mean AUROC over 14 classes as our evaluation metric. This metric is also used by CheXNet. We used sklearn.metrics.auc_score to calculate the AUROC for each class.



Experimental Procedure

- Uses 70-10-20
  Train-Valid-Test split

- No patient overlap between sets to prevent data leakage



Pre-processing & Augmentation

Preprocessing:

- Resized images to 224x224, then normalized according to mean and variance of ImageNet dataset.

Augmentation:

- For augmentation, we did not use random flipping like the paper did, due to the inherent asymmetrical nature of human thoracic anatomy.
- Instead, we opted for *random cropping* and adding a slight *random rotation* of +- 15 degrees.
- However, the model converged to poorer levels of performance, therefore we omitted the data augmentation step altogether.

Training

Since our model aims to predict the independent probabilities of each disease being present in a patient's chest xray image, the appropriate loss function to optimize would have to be the **Binary Crossentropy Loss**.

However, due to a severe class imbalance, we experimented with the use of Weighted Binary Crossentropy Loss, where each class in the ground truth label is assigned a weight based on its relative number of occurrences, as described below. This technique significantly improved the AUROC scores of our trained models.

> For example, if a dataset contains 100 positive and 300 negative examples of a single class, then *pos_weight* for the class should be equal to $\frac{300}{100} = 3$. The loss would act as if the dataset contains $3 \times 100 = 300$ positive examples.

Source: BCEWithLogitsLoss — PyTorch 2.0 documentation

Training Hyperparameters

XvT: **AdamW** optimizer is used with the **weight decay of 0.05** for XvT. We train XvT with an initial **learning rate of 0.00025** and a batch size of 128 for 40 epochs, with a **cosine learning rate decay scheduler.**

Other models: For our other models, we found that the **Adam** optimizer with a standardized **learning rate of 0.0001** and **L2 regularization (weight decay) with factor of 1 * 10^(-5)** worked best. We also added a **scheduler** which reduces the learning rate to a factor of 20% whenever a plateau in validation loss is hit (when eval loss does not improve for 3 epochs in a row).

Results Comparison

Pathology	CheXNet	CNN	ResNet	DenseNet	UnetEncoder	XvT
Atelectasis	**0.8217**	0.6790	**0.7330**	0.7139	0.7193	0.6866
Cardiomegaly	**0.9054**	0.7221	**0.8450**	0.8412	0.8018	0.7727
Effusion	**0.8821**	0.7478	**0.8176**	0.7994	0.7758	0.7439
Infiltration	**0.7082**	0.6376	**0.6689**	0.6600	0.6407	0.6336
Mass	**0.8491**	0.5716	**0.6874**	0.6766	0.6240	0.6132
Nodule	**0.7740**	0.5431	**0.6407**	0.6392	0.5759	0.5517
Pneumonia	**0.7730**	0.6474	**0.7111**	0.6907	0.6281	0.6568
Pneumothorax	**0.8661**	0.6238	**0.7782**	0.7676	0.7110	0.6753
Consolidation	**0.8026**	0.7348	**0.7748**	0.7644	0.7318	0.7382
Edema	**0.8897**	0.8279	**0.8644**	0.8539	0.8398	0.8130
Emphysema	**0.9113**	0.6120	**0.7783**	0.7400	0.6929	0.6443
Fibrosis	**0.8220**	0.6831	**0.7502**	0.7263	0.6974	0.6506
Pleural Thickening	**0.7742**	0.6072	**0.6961**	0.6838	0.6412	0.6073
Hernia	**0.8867**	0.8372	**0.8761**	0.8547	0.7641	0.6970
Average AUROC	**0.833**	0.677	**0.759**	0.744	0.703	0.677
Trainable params	7,051,854	19,026	6,299,342	2,136,302	21,129,038	2,823,758

Table 1: AUROC scores. **Underline** represents best result across all models. **Bold** represents the best result in our experiments.

Discussion: Observation of results

Pathology	CheXNet	ResNet
Atelectasis	**0.8217**	**0.7330**
Cardiomegaly	**0.9054**	**0.8450**
Effusion	**0.8821**	**0.8176**
Infiltration	**0.7082**	**0.6689**
Mass	**0.8491**	**0.6874**
Nodule	**0.7740**	**0.6407**
Pneumonia	**0.7730**	**0.7111**
Pneumothorax	**0.8661**	**0.7782**
Consolidation	**0.8026**	**0.7748**
Edema	**0.8897**	**0.8644**
Emphysema	**0.9113**	**0.7783**
Fibrosis	**0.8220**	**0.7502**
Pleural Thickening	**0.7742**	**0.6961**
Hernia	**0.8867**	**0.8761**
Average AUROC	**0.833**	**0.759**
Trainable params	7,051,854	6,299,342

Observation 1: Our best performing model is ResNet

With a similar number of trainable parameters of our ResNet as compared to CheXNet, we managed to achieve a score which is 0.074 away from CheXNet

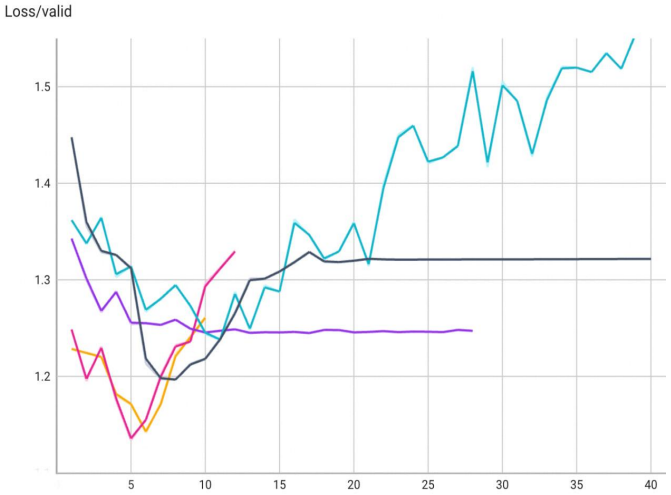
Discussion: Observation of results

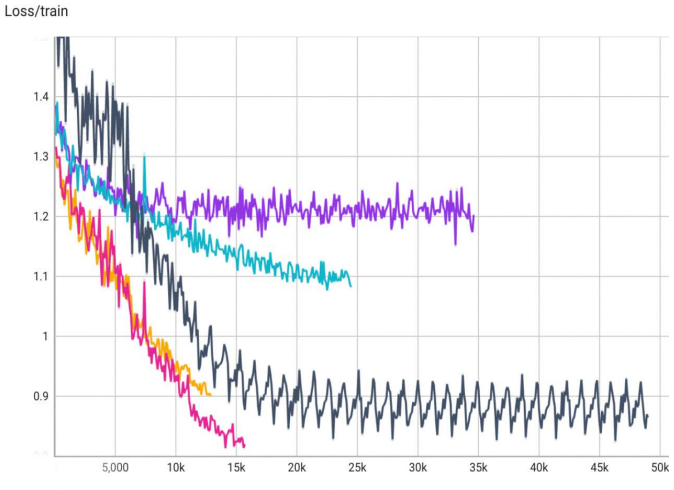
Pathology	CheXNet	XvT
Atelectasis	**0.8217**	0.6866
Cardiomegaly	**0.9054**	0.7727
Effusion	**0.8821**	0.7439
Infiltration	**0.7082**	0.6336
Mass	**0.8491**	0.6132
Nodule	**0.7740**	0.5517
Pneumonia	**0.7730**	0.6568
Pneumothorax	**0.8661**	0.6753
Consolidation	**0.8026**	0.7382
Edema	**0.8897**	0.8130
Emphysema	**0.9113**	0.6443
Fibrosis	**0.8220**	0.6506
Pleural Thickening	**0.7742**	0.6073
Hernia	**0.8867**	0.6970
Average AUROC	**0.833**	0.677
Trainable params	7,051,854	2,823,758

Observation 2: XvT did not perform as well.

One possible reason is that our transformer model might not be large enough to produce significant result

Discussion: Observation of results





Legend

- XvT
- UNet
- Naive Conv
- DenseNet
- ResNet

Observation 3: Our larger architectures (DenseNet, ResNet & UNet) generally performed better, but face the drawback of overfitting more quickly.

Deduction: Having more trainable parameters allow these larger architectures to capture more complex features, but due to the limited diversity in our training dataset, the models end up learning overly complex features to fit the training data too well, resulting in quick divergence of its validation loss. Our best step moving forward would be to increase the size of our models, while carefully adjusting factors such as data augmentation, learning rate, and regularization to mitigate the increasing tendency to overfit.

Discussion: Observation of results

Pathology	CheXNet
Atelectasis	**0.8217**
Cardiomegaly	**0.9054**
Effusion	**0.8821**
Infiltration	**0.7082**
Mass	**0.8491**
Nodule	**0.7740**
Pneumonia	**0.7730**
Pneumothorax	**0.8661**
Consolidation	**0.8026**
Edema	**0.8897**
Emphysema	**0.9113**
Fibrosis	**0.8220**
Pleural Thickening	**0.7742**
Hernia	**0.8867**
Average AUROC	**0.833**
Trainable params	7,051,854

Observation 4: Even though our models are smaller than the DenseNet architecture used by ChexNet, our models began to overfit within a few epochs, whereas the ChexNet was able to be trained for much longer and still achieved much better results.

Deduction: The ChexNet model is built from a DenseNet model which was pre-trained to extract useful features from a diverse dataset of images. Fine-tuning such a model **reduces the search space of trainable parameters**, and **requires updating only a small subset of trainable parameters**, thereby **reducing the likelihood of overfitting** on the training dataset.

Conclusion

In conclusion, our main contributions/findings are as follows:

- Using weighted binary crossentropy loss is effective in alleviating class imbalance.
- Explored the use of Convolutional Transformers in Multi-Label Image Classification
- Further improvements in results would require larger architectures and better mitigation of the increased overfitting tendencies (via data augmentation, regularization, tuning learning rate)
- The disparity in performance between our models and the ChexNet model can be largely attributed to the use of pre-training in the ChexNet model.