Learnings from Classification of X-Ray Images Using LeNet, DenseNet, AlexNet, and Custom CNN

Sang Ik Han, Byung Hui Yoon, Philipp Gaissert, Amar Mohanty

December 10th, 2021

Abstract

Pneumonia is a deadly disease where the lungs fill with fluid making breathing difficult. Traditionally, identification of pneumonia requires radiologists to look at chest x-ray scans of patients. We aim to automate this process using machine learning. In this study, we evaluated DenseNet, LeNet-5, AlexNet, and custom CNN to classify chest x-ray images from Kaggle. We found that a pre-trained DenseNet model with transfer learning performed the best with 90.06% accuracy and 92.43% F1 score. Out of the four models we explored, we found that more complex models tend to perform better if some level of regularization was applied.

1 Motivation

Pneumonia is a lung infection where the lungs fill with fluid and make breathing difficult. This deadly sickness is unfortunately the leading cause of death among children under 5 years of age and is the most common cause of hospital admissions other than childbirth in the US [1]. To diagnose if a patient has Pneumonia, they are given a chest x-ray. Soon after, a radiologist would look at the scan to identify white spots in the lungs called infiltrates, which is an indicator of Pneumonia. We aim to automate the identification process with machine learning.

Pneumonia detection is suited for Machine Learning because we have access to a large database of high resolution x-ray scans of patients with and without Pneumonia. We framed this problem as a binary classification problem of pneumonia vs normal patients which can be modeled using LeNet, AlexNet, CNN, and DenseNet.

2 Related Work

Recently, there has been similar work with Pneumonia Identification using Chest X-Rays for patients with and without COVID-19 [2]. They were able to build an AlexNet model using chest x-rays to identify non-COVID-19 viral pneumonia and normal patients with a 94.43% accuracy, 98.19% sensitivity, and 95.78% specificity [2]. We aim to build upon this research by creating a general purpose model that classifies bacterial and viral pneumonia vs normal patients. Additionally, it is difficult to compare their performance apple-to-apple with other available models since we do not have access to their data set.

For the baseline performance of our models, we used LeNet-5, which was the basis for CNN[3]. It was first developed to recognize hand-written letters (32×32 pixel images) and got renowned for replacing "hand-crafted feature extraction" with "carefully designed learning machines" [3]. LeCun came up with an idea to feed the model with minimally processed inputs, which were the raw pixel images.

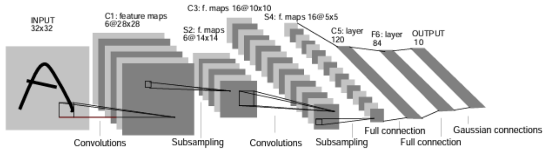


Figure 1: Model schematic of LeNet [3]

There has also been work with more complex models for Pneumonia Lung identification. For example, Mittal et al. found that using DenseNet-169 as a feature extractor and SVM as a classifier had an AUC score of .8002 for identifying pneumonia and non-pneumonia patients using chest x-rays [4]. The biggest limitation of this approach is that it requires large computational resources to train the model.

3 Dataset

For our dataset we used chest x-ray images of patients with and without pneumonia from Kaggle. The training and test sets each have images labeled as `PNEUMONIA` or `NORMAL`.

3.1 Dataset Dimensions

We split 10% of the provided training set to use as our validation set by implementing a stratified random sampling provided by `scikit-learn` to ensure that the sets shared the same class distribution. Below is the dataset sample sizes:

	Pneumonia	Normal
Training	3487	1207
Validation	388	134
Testing	390	234

Figure 2: Sample Size by Category

3.2 Class Balancing

Our training set has 74% pneumonia cases which leads to class imbalance. This is detrimental to our models because it is likely to increase recall while driving down specificity since the over-represented pneumonia cases can skew the model towards predicting pneumonia.

To account for the class imbalance in our dataset, we implemented weighted random sampling to upsample `NORMAL` images compared to `PNEUMONIA` images. Weights were computed for each image, depending on its class, and fed to a `WeightedRandomSampler` from `PyTorch`. This ensured that images sampled by the `DataLoader` in each mini-batch had a balanced distribution of the two classes during training.

3.3 Image Dimensions

The images did not have uniform dimension, and were not square. Thus, to prepare the images to be suitable inputs for our models, we used transformers in the `torchvision` package provided by `PyTorch`. In both sets, we used a rescaling transformer to scale the images down to 230 px (on their shortest side) and a center-cropping transformer to make all of the images square (224×224 px). To increase the amount of diversity in our training set, we used additional transformers that would apply random augmentations in each training epoch, including rotation ($\pm 10°$) and equalization.

Please note that the raw images also didn't have uniform orientation of the body or proportion as we can see from Figure 3. After applying transformation to feed into the model, the images still have some level of inconsistency as in Figure 4. While most images contain lung areas, some of them consist mostly of spine areas, which may adversely impact the model performance.

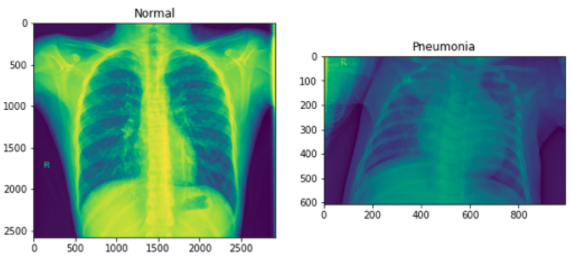


Figure 3: Raw Images Before Transformation

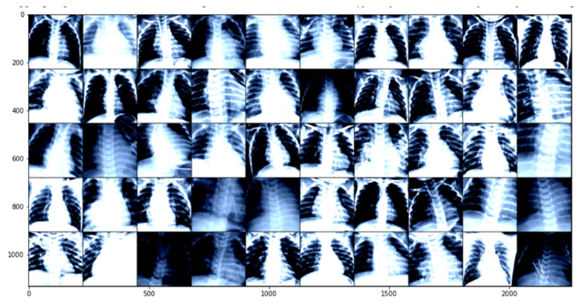


Figure 4: Transformed Image for DenseNet

4 Problem Formulation

Our goal was to develop a deep learning image classifier to diagnose pneumonia from a chest x-ray. In order to determine a suitable candidate for our classifier's architecture, we trained and compared the performance of four different convolutional neural networks: LeNet-5, AlexNet, DenseNet, and a custom CNN.

We formulated this as a binary classification problem, where the label $y = 1$ denotes `PNEUMONIA` and the label $y = 0$ denotes `NORMAL`. Our models were trained to minimize the cross-entropy loss with softmax, the most widely used loss function for deep learning classifiers.

5 Methods

5.1 LeNet-5 (Baseline Model)

LeNet-5, one of the earliest CNN architectures, was originally designed for classifying 32×32 grayscale images (and 28×28 with padding), e.g. the MNIST dataset. Given the relative simplicity of the problem it was designed to solve, its small number of convolutional layers, and its sensitivity to the "vanishing gradient problem" due to its use of a hyperbolic tangent activation function, it made sense to set LeNet-5 as our baseline model.

We implemented the LeNet-5 in `PyTorch`. In order for LeNet-5 to take our images (224×224 after pre-processing) as inputs, three subsampling layers (kernel width: 2, stride: 2) were added before the first convolutional layer to max-pool the images down to $28 \times 28 \times 3$.

Modern regularization techniques like dropout were considered for the purpose of controlling overfitting, but were ultimately not used in order to maintain an honest implementation of the LeNet-5 architecture.

5.2 Custom CNN

The custom CNN was modeled with `PyTorch`'s nn.sequential function which enables us to conveniently build our own CNN model. We implemented the custom CNN model to speculate how model complexity changes and what roles each layer play. CNN model 1 consists of convolutional layer that has 5x5 filter with stride 2 followed by ReLU activation and a max-pooling layer that has 2×2 filter size. Another identical set of layers were inserted to the model followed by a fully connected layer.

We built two CNN models with and dropout without dropout layers to see the effects of overfitting. In the model with dropout layers, we included the dropout layers after each max-pooling layer as well as after the fully-connected layer.

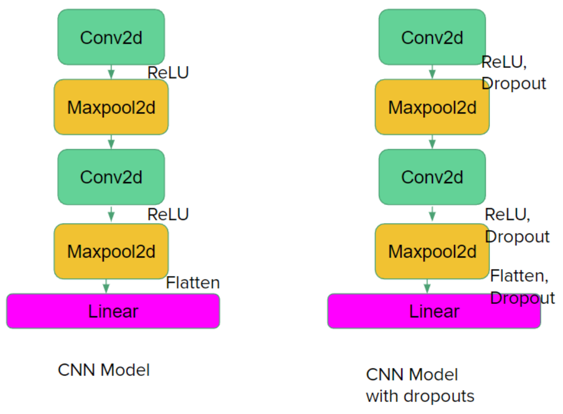


Figure 5: Custom CNN model

5.3 AlexNet

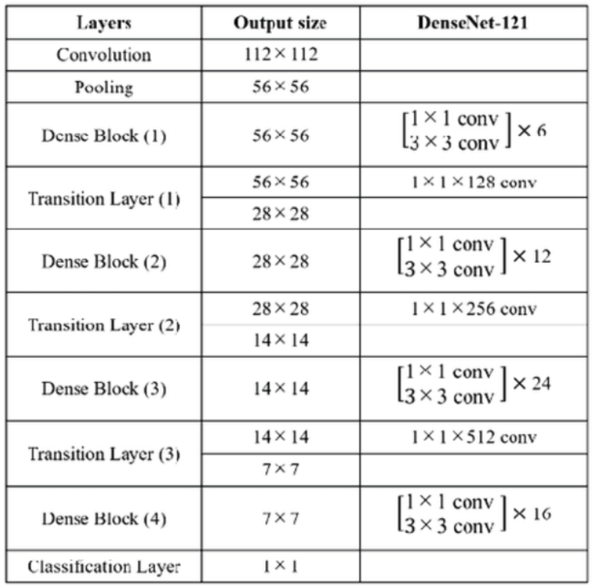
We chose AlexNet because it is a classic CNN architecture that is used in image classification. This older model gained its fame in 2012 when it won the ImageNet visual recognition challenge. Additionally, AlexNet was the basis of DenseNet so we wanted to see the performance differences between the two.

To do so, we used a pre-trained AlexNet model from `PyTorch`. This model has 5 convolution layers, 3 max pooling layers, 2 dropout layers, and 3 fully connected layers with ReLu activation functions. The model is slightly different from the classic AlexNet architecture because it includes dropout layers. However, those layers would be useful to ensure that we are not overfitting to the data.

5.4 DenseNet with Transfer Learning

We implemented DenseNet with transfer learning. DenseNet is by far the most complex out of all the models we investigated. Since each layer connects to every other layer in a feed-forward fashion, DenseNet has L(L+1)/2 direct connections, where L is the number of layers. We initially started with DenseNet161, which has the lowest Top-1 error when tested on other data sets. But it also has higher risk of overfitting since it creates input of size 2208 for the classification layer. So we chose DenseNet121, which reduces the input size by nearly half (=1024) for the classifier.

We replaced the last classification layer with a new sequential layer. In order to avoid overfitting, we increased the drop out and added extra bottleneck, but increased the number of epochs from 12 to 30.



Layers	Output size	DenseNet-121
Convolution	112×112	
Pooling	56×56	
Dense Block (1)	56×56	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times 6$
Transition Layer (1)	56×56	$1\times1\times128$ conv
	28×28	
Dense Block (2)	28×28	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times 12$
Transition Layer (2)	28×28	$1\times1\times256$ conv
	14×14	
Dense Block (3)	14×14	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times 24$
Transition Layer (3)	14×14	$1\times1\times512$ conv
	7×7	
Dense Block (4)	7×7	$\begin{bmatrix}1\times1\text{ conv}\\3\times3\text{ conv}\end{bmatrix}\times 16$
Classification Layer	1×1	

Figure 6: DenseNet121 Architecture[5]

We imported a pretrained DenseNet121 model with the architecture shown above from `PyTorch`, and replaced the last classifier layer with the custom layers below. Since our primary concern was overfitting, we experimented with dropout and bottleneck to improve our model.

Layer1	Fully Connected	(in:1024, out:256)
	Dropout	30%
	ReLu	
Layer2	Fully Connected	(in:256, out:32)
	Dropout	30%
	ReLu	
Layer3	Fully Connected	(in:32, out:2)
	Dropout	30%
	ReLu	
Out	Softmax	

Figure 7: Classifier Architecture

6 Experiments and Results

6.1 Evaluation Metrics

We used the following metrics to evaluate the performance of our models: classification accuracy, precision, recall, specificity, F1 score, and the area under the curve (AUC) of the receiver operating characteristic (ROC) curve.

Accuracy served as a general performance metric for binary classification and allowed us to visualize how our models improved over training epochs. However, in a real world setting this is insufficient as a performance metric for medical diagnosis. In the context of our classification problem, a false negative means that a patient may not receive the medical treatment they require to overcome their illness, while a false positive may lead to the patient receiving unnecessary treatment, which may carry both greater financial costs and a greater risk of harmful side-effects from treatment. Therefore it was particularly important for our models to correctly diagnose pneumonia when present and to avoid incorrectly diagnosing pneumonia when absent. Hence, we placed greater emphasis on recall and specificity when evaluating and comparing our models, as well as the F1 score, which takes both into account:

$$\text{F1} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

6.2 Evaluation Results

Model	Acc.	Prec.	Rec.	Spec.	F1
DenseNet	90.06%	88.13%	97.17%	78.20%	92.43%
AlexNet	83.97%	81.39%	96.41%	63.25%	88.27%
Custom CNN	78.85%	75.39%	98.21%	46.58%	85.30%
LeNet-5	65.87%	59.40%	96.74%	35.96%	73.61%

Figure 8: Performance on the Testing Set

6.3 Data Contamination

All of the experiment results below in this section show highly synchronized training accuracy and validation accuracy curves. This tendency made it hard for us to perform any meaningful hyperparameter turning. We tried to desynchronize train and validation sets by shuffling the data set before the random split but our efforts were not successful. During our discussion with the professor, he mentioned the possibility that multiple images are taken from the same set of patients.

After running the test set along with train and validation sets, we realized that the the hypothesis of the professor is likely to be true. It is otherwise hard to explain why validation accuracy and train accuracy are almost synchronized, but test accuracy show drastically different trend shown in Figure 8. The validation set was drawn from the same pool of data as train set using random split. We couldn't identify the root cause, but we think it might be because multiple images are taken from the same patients, hospital, or machines.

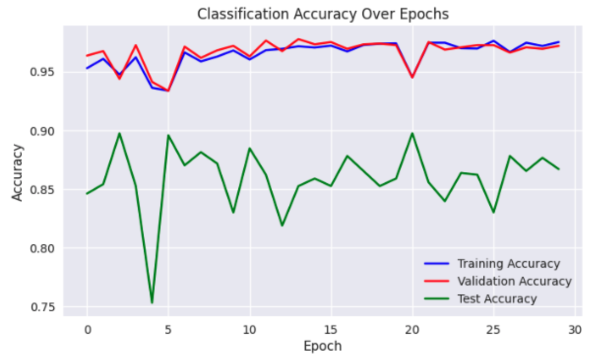


Figure 9: Accuracy Curves over Train, Validation, and Test Sets (DenseNet)

6.4 LeNet-5 (Baseline Model)

Our LeNet-5 model showed the poorest performance of the models on the testing set, with an overall classification accuracy of 65.87% and F1 score of 73.61%. This outcome was expected due to the reasons stated in Section 5.1. It's architecture was the simplest, with only ten layers in total, including the three extra max-pooling layers. Though the extra max-pooling layers allowed the LeNet-5 model to take the images as inputs, it is possible that important features were lost or obscured before the first convolutional layer.

To maintain an honest implementation of the original LeNet-5 architecture, no dropout or other modern regularization techniques for CNN were implemented. This explains the low testing accuracy, given the high training and validation accuracy (Fig. 11).

The most notable result given by the LeNet-5 model was its low specificity, 35.96%. This means that, given a `NORMAL` image, the probability of the model correctly classifying the image is worse than random chance. This was surprising, given that we had handled class imbalance in the dataset with weighted random sampling.

We used the following hyperparameters for our LeNet-5 model:

```
{
    torchvision_transforms: [
        RandomRotation(10),
        RandomEqualize()
    ],
    learning_rate: 0.0001,
    batch_size: 50,
    training_epochs: 12
}
```

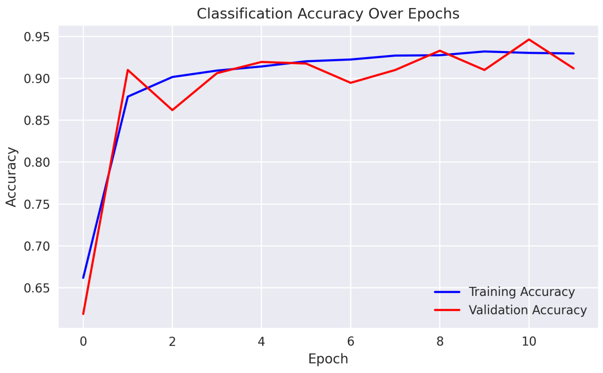


Figure 10: Accuracy over epochs (LeNet-5)

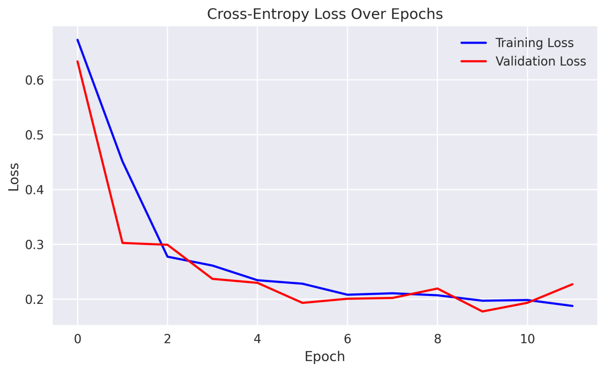


Figure 11: Cross entropy loss over epochs (LeNet-5)

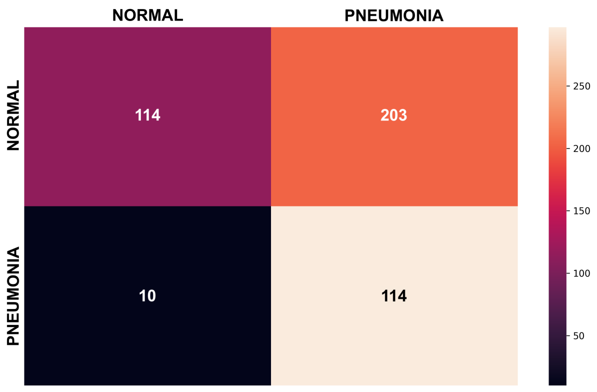


Figure 12: Confusion Matrix (LeNet-5)

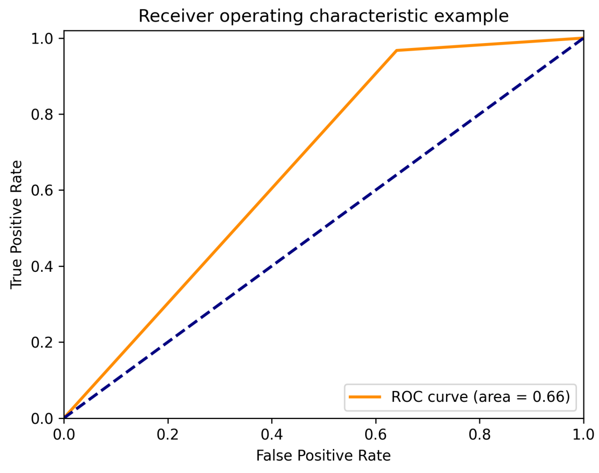


Figure 13: ROC curve (LeNet-5)

6.5 Custom CNN

We experimented with a custom CNN architecture with and without dropout layers. We found that adding dropout layers to our CNN architecture increased the F1-score (86.64% vs 82.52%). The main improvement was with specificity because it went from 31.19% to 46.58%. We expected this to happen because the model would be less prone to overfitting. Additionally, when we added the dropout layer, we increased number of epochs from 12 to 20 since there was less concern about over-fitting. While CNN model with dropout layers showed slightly higher testing accuracy, there was no noticeable difference in training accuracy over epochs. This probably means that over-fitting is the primary bottleneck for improving testing accuracy, and regularization can help improve out models. We also noted that the loss and accuracy converged much slower than AlexNet and DenseNet did. This is in line with our expectation since AlexNet and DenseNet used pre-trained models that can converge faster.

Here is the list of hyperparameters we used for our CNN models:

```
{
    torchvision_transforms: [
        RandomHorizontalFlip(),
        RandomAffine(10),
        RandomAugmentation()
    ],
    learning_rate: 0.0001,
    batch_size: 32,
    training_epochs:
        12 - CNN without Dropout Layers
        20 - CNN with Dropout Layers
}
```

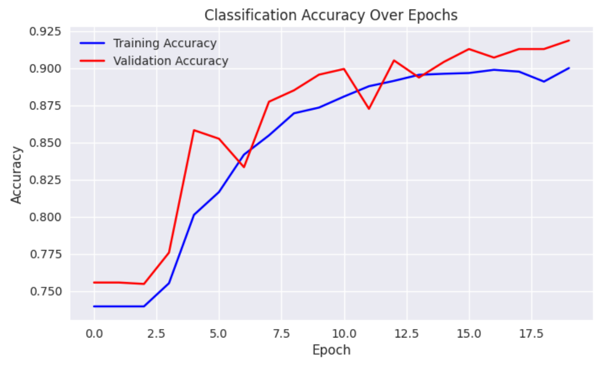


Figure 14: Accuracy over epochs (Custom CNN with Dropout Layers)

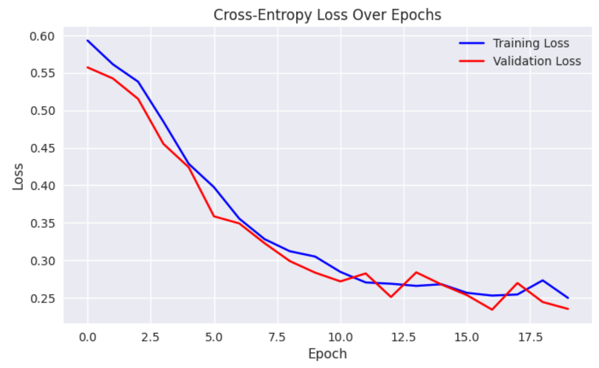


Figure 15: Cross entropy loss over epochs (Custom CNN with Dropout Layers)

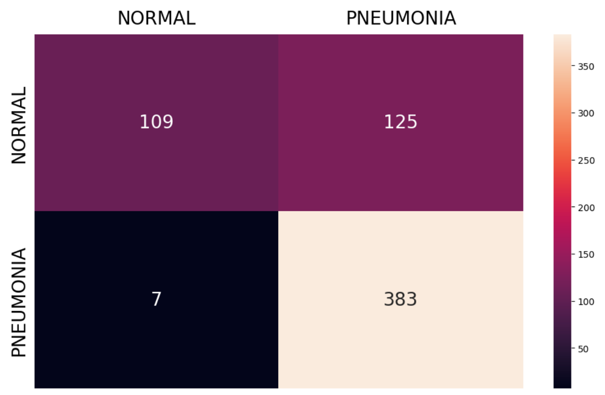


Figure 16: Confusion Matrix (Custom CNN with Dropout Layers)

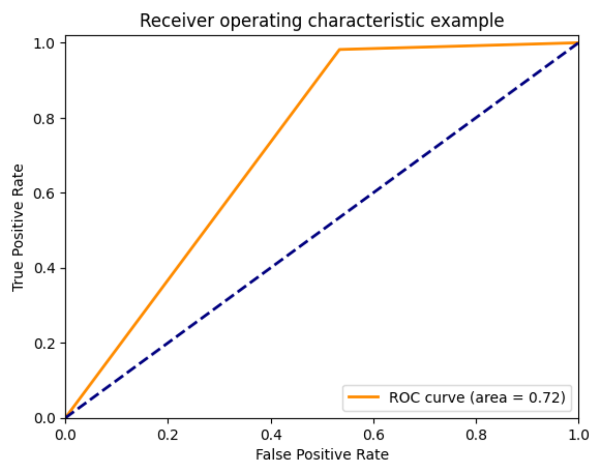


Figure 17: ROC Curve (Custom CNN with Dropout Layers)

6.6 AlexNet

As mentioned earlier, it was difficult to tune the parameters due to data contamination between the validation and training data. Thus, to tune our hyperparameters, I manually adjusted the values. For example, I initially ran my model for 50 epochs and found that 12 epochs gave me the best results. The following parameters gave the best testing results:

```
{
    torchvision_transforms: [
        RandomHorizontalFlip(),
        RandomAffine(10),
        RandomAugmentation()
    ],
    learning_rate: 0.001,
    batch_size: 32,
    training_epochs: 12
}
```

AlexNet had a relatively high F1 Score of 88.27%. We found that after data balancing, our model's specificity increased significantly because it was over-fitting to the pneumonia data. However, our model still had a significantly higher sensitivity than specificity. That being said, those results are acceptable because we want to make sure that we identify as many pneumonia patients as possible even if it results in some false positives.

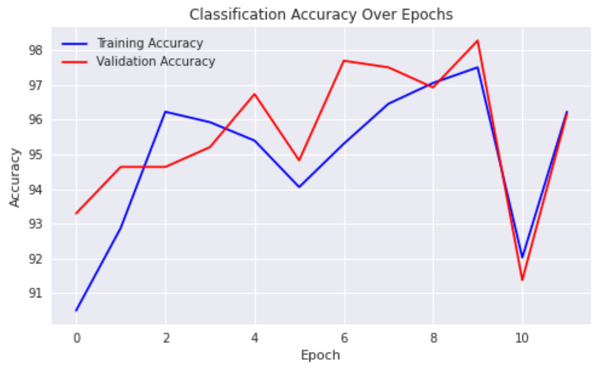


Figure 18: Accuracy over epochs (AlexNet)

Because we were using a pre-trained AlexNet model, it was able to converge to an optimal solution rapidly as shown in the cross-entropy loss figure. Additionally, this could also attribute to why the training and validation accuracy starts so high.

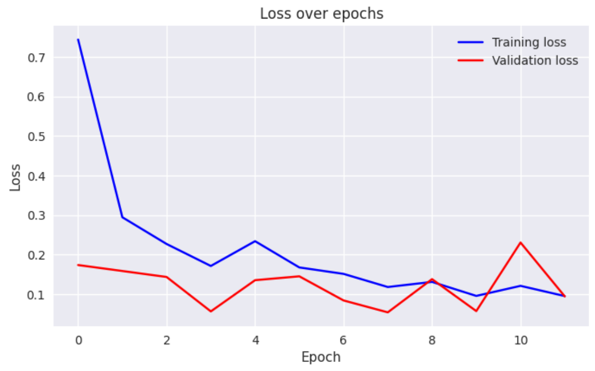


Figure 19: Cross entropy loss over epochs (AlexNet)

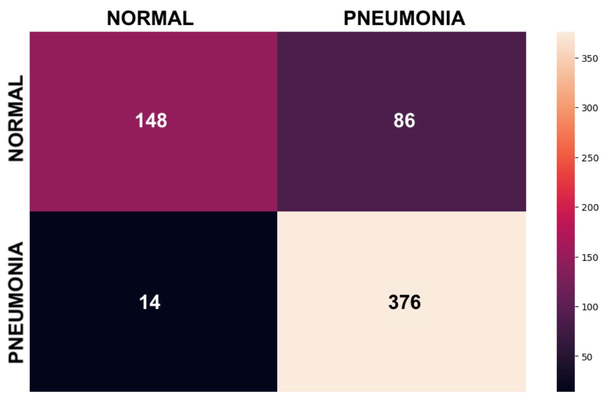


Figure 20: Confusion Matrix (AlexNet)

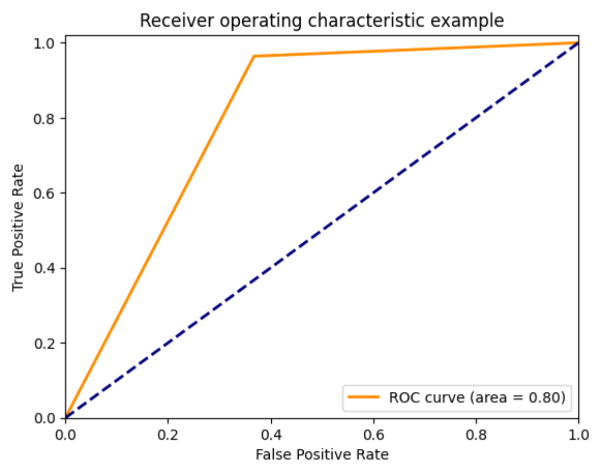


Figure 21: ROC Curve (AlexNet)

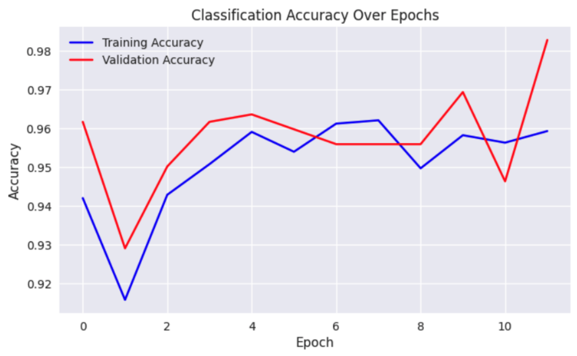


Figure 22: Accuracy over epochs (DenseNet)

6.7 DenseNet with Transfer Learning

DenseNet has the highest F1 score of 92.43% and performs better than other models on all other evaluation criteria. DenseNet reaches training accuracy above 90% only after the first epoch. DenseNet was also the least over-fitting model with the smallest gap between training accuracy and testing accuracy. This is probably because we imported a pre-trained model with highly complicated architecture for feature extraction, and focused our efforts mainly on regularization. Since we were primarily concerned about over-fitting, we added bottleneck for the classification layer, with two drop out layers each with 30% dropout rates. We tried adding more dropout layers or increasing dropout rates, but two dropout layers each with 30% dropout rate seemed to yield the optimal results. We used the following hyperparameters for DenseNet:

```
{
    Preprocessing:
    (RandomRotation(10), RandomEqualize()),
    learning_rate: 0.0001,
    batch_size: 50,
    # of dropout layers: 2
    dropout rates: 30%
    training_epochs: 30
}
```

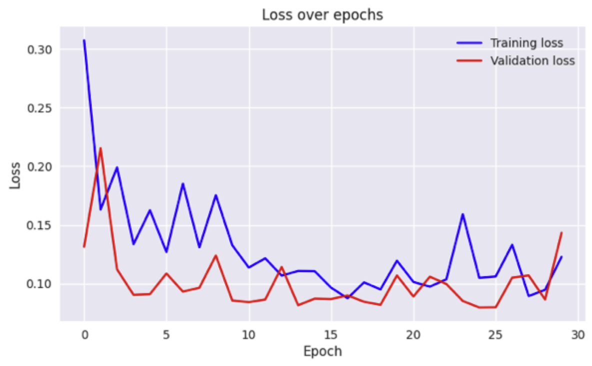


Figure 23: Cross entropy loss over epochs (DenseNet)

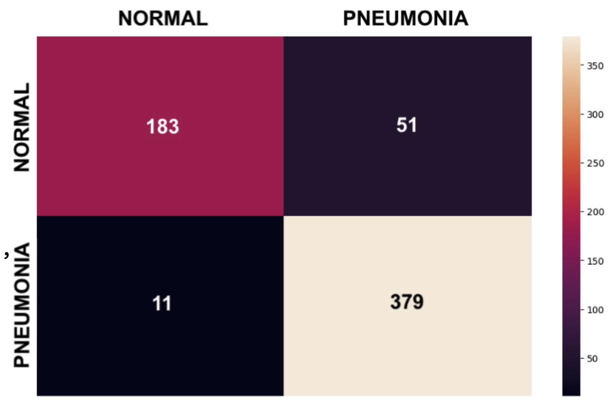


Figure 24: Confusion Matrix (DenseNet)

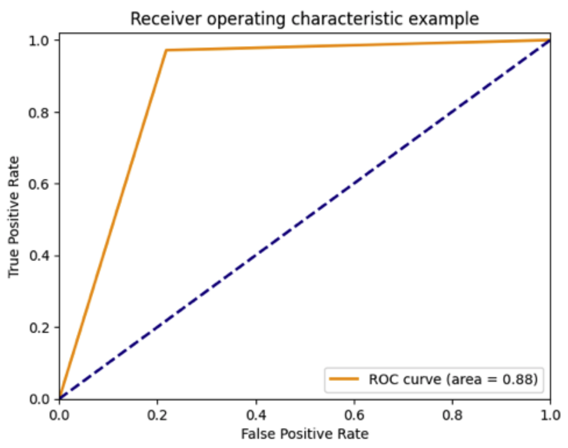


Figure 25: ROC Curve (DenseNet)

7 Conclusion and Discussion

7.1 Summary of Overall Performance

Out of the four models we explored, DenseNet performed the best in all of our evaluation metrics. We expected this because it is the newest and most complex model. Additionally, it could take advantage of the pre-trained layer that can extract features at very granular level and focus the training process mainly on regularization.

Because we found at the final stage of our research that our data was contaminated, we speculate that we could have performed more hyperparameter turning that led to better results. Also, the contaminated dataset itself might have led to lower testing accuracy, since the repeated instances of images sharing similar characteristics can drive our models to overfit to the training set.

7.2 LeNet-5 vs CNN Results

We found that the custom CNN model performed better than the LeNet-5 model despite having 1 less convolution layer. This may have occurred because LeNet-5 was originally designed to have an image of 28 x 28 pixels. So we had to downsample our image significantly to fit the model. Alternatively, our CNN model was designed to get a 224 x 224 px image. This results illustrates the negative impact downsampling an image can have on the performance of your model.

7.3 Lessons Learned from Data Quality and Imbalance

Through this project, we learned the importance of data quality in a machine learning project. Unfortunately, we found that our testing and validation sets were contaminated which led to our model's having similar testing and validation performance. This made parameter tuning difficult. Additionally, we found that in some of our models that the model's F1 score and accuracy reduced after data balancing. We believe this may have happened because the data contamination caused our model to overfit to the training data.

7.4 Visual Inspection of Results

We visually inspected the results to see if there are discernible trends in misclassified cases. While we found that some images had extra noise such as the patient wearing accessories in the X-Ray image, we couldn't identify a clear trend from the images that were misclassified.

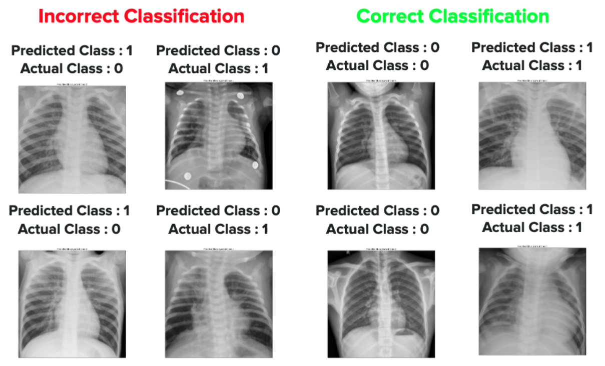


Figure 26: Classified Image Samples (AlexNet)
1 = Pneumonia / 0 = Normal

7.5 Future Work

We foresee some future research opportunities to improve upon the models that we built. The biggest improvement would be ensuring that the dataset has one x-ray per patient to ensure no data contamination. Additionally, we could add more regularization to our data, via techniques such as dropout, to ensure that there is less overfitting. Also, establishing a well-defined set of random augmentations that better reflect real-world noise could help increase the amount of useful data in our training set. Finally, we could add more classification to the data to not only distinguish between normal and pneumonia lungs but also what type of pneumonia, bacterial or viral.

References

[1] A. T. Society, “Top 20 pneumonia facts—2019,” 2019. Last accessed 11 December 2021.

[2] D. Varshni, K. Thakral, L. Agarwal, R. Nijhawan, and A. Mittal, “Pneumonia detection using cnn based feature extraction,” in *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–7, 2019.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] A. U. Ibrahim, M. Ozsoz, S. Serte, F. Al-Turjman, and P. S. Yakoi, “Pneumonia Classification Using Deep Learning from Chest X-ray Images During COVID-19,” *Cognit Comput*, pp. 1–13, Jan 2021.

[5] R. Sato, Y. Iwamoto, K. Cho, D.-Y. Kang, and Y.-W. Chen, “Accurate bapl score classification of brain pet images based on convolutional neural networks with a joint discriminative loss function †,” *Applied Sciences*, vol. 10, p. 965, 02 2020.