Comparing Image Classification Models on Medical Images

Raghav Juyal EP20BTECH11018

Abstract— **Machine learning on image data is very important, especially for medical science. However, training models on image data is computationally expensive. The aim of this project is to test various image classification models and compare their performance. By doing so we can better understand which models work better and which hyperparameters affect the performance more. Doing so is important since we will gain a good starting point to choose our model, thus saving time and computational resources.**

I. Introduction

Machine learning applied to image analysis holds paramount significance, particularly within medical science, where the ability to discern and classify diseases from scans can substantially enhance diagnostic accuracy and patient care. Images encapsulate an immense reservoir of data, rendering the training of models on such data an exceedingly resource-intensive process. The computational demands are staggering, compounded by the requisite need for copious amounts of meticulously labelled data.

Within the constraints of the available experimental evidence, my focus lies in an in-depth exploration of performance metrics across a range of models applied to the analysis of medical image datasets. Through meticulous comparison and evaluation of models such as logistic regression, convolutional neural networks (CNNs), vision transformers, and others, I aim to identify empirical trends behind superior model performance on specific image data types.

This empirical analysis seeks to derive patterns from the experimental outcomes. By relying solely on empirical evidence, the goal shifts towards recognizing historical trends in model efficacy for certain types of image data. This approach allows for a streamlined process in selecting models for testing, potentially reducing the extensive time spent on model selection and trial. Prioritizing models based on their empirical performance history enables a more efficient allocation of resources and computational power toward refining selected models. Consequently, this method optimizes research efforts by enhancing the chosen model's performance, paving the way for potential real-world deployment within the medical domain.

II. Performance Metrics

In this project, we calculate the following metrics for the models that are trained:

1) Time: Time refers to the duration required for a model to be trained on a dataset. It measures the computational efficiency of the model training process.
2) Accuracy: Accuracy calculates the ratio of correctly predicted instances to the total number of instances in the dataset. It measures the overall correctness of predictions made by the model.
3) Precision: Precision quantifies the accuracy of positive predictions made by the model. It is calculated as the ratio of true positive predictions to the sum of true positive and false positive predictions.
4) Recall: Recall measures the model's ability to identify all positive instances correctly. It calculates the ratio of true positive predictions to the sum of true positives and false negatives.
5) F1 Score: The F1 Score is the harmonic mean of precision and recall. It provides a balance between precision and recall, offering a single metric that considers both false positives and false negatives.

III. Datasets

In this project, we use the following 2 datasets.

A. Dataset 1[1]

The dataset contains 5824 chest X-ray images. It is a binary classification between people who have pneumonia and do not have pneumonia. There are 4077 training images, 1165 validation images, and 582 test images. Each image has the following pre-processing:

1) Auto-orientation of pixel data (with EXIF-orientation stripping)
2) Resize to 640x640 (Stretch)

B. Dataset 2[2]

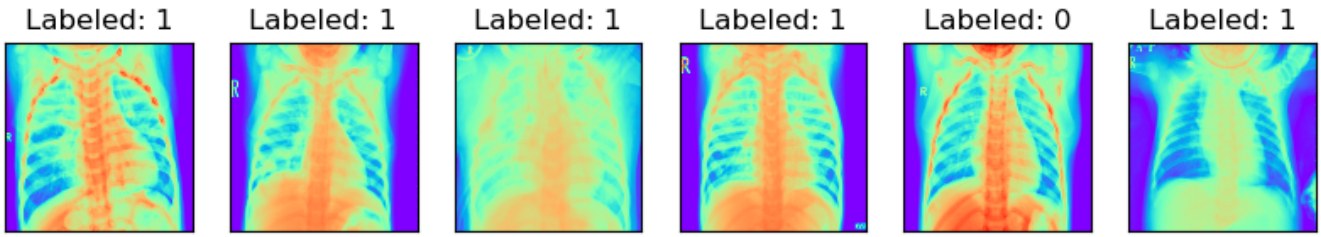
The dataset contains 112,120 frontal-view X-ray images of 30,805 unique patients. There are 15 classes: Fourteen common thoracic pathologies include Atelectasis, Consolidation, Infiltration, Pneumothorax, Edema, Emphysema, Fibrosis, Effusion, Pneumonia, Pleural thickening, Cardiomegaly, Nodule, Mass and Hernia, and No Finding. There are 86,524 training images and 25,596 test images. While some images can be labelled with multiple classes, due to computational constraints, I only used the first class labelled.

IV. Models

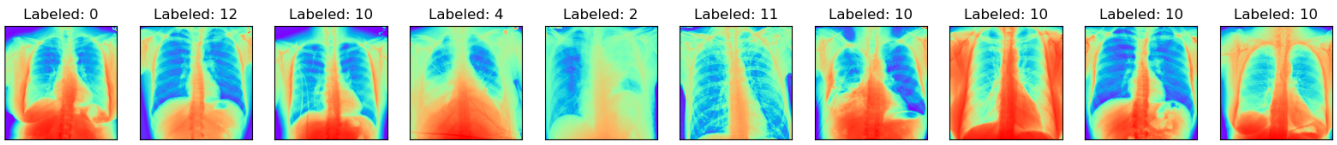
In this project, we use the following models:

A. Logistic Regression

Logistic Regression is a linear classification algorithm used for binary classification problems. Despite its name, it's a regression model adapted for classification by employing the logistic function to squash the output between 0 and 1, representing class probabilities. It's simple, interpretable, and efficient for linearly separable data. Logistic Regression is only used on dataset 1.



(a) Sample Images for Dataset 1



(b) Sample Images for Dataset 2

Fig. 1: Sample Images for Datasets

B. Convolutional Neural Networks

CNNs are a class of deep neural networks widely used for image recognition, classification, and computer vision tasks. They consist of convolutional layers that extract features from input images through filters and pooling layers that downsample the extracted features. CNNs are adept at learning hierarchical representations and patterns within images. CNNs from scratch are used only on dataset 1.

C. DeiT[3]

DeiT is a vision transformer model that applies the transformer architecture initially developed for natural language processing (NLP) tasks to image classification. It relies on self-attention mechanisms to capture long-range dependencies and relationships within the image pixels, achieving competitive performance with less data compared to other models. DeiT is used on both datasets.

D. ResNet 50[4]

ResNet is a deep convolutional neural network architecture known for introducing residual connections. These connections allow the network to learn residual mappings, enabling the training of deeper networks without facing the vanishing gradient problem. ResNet-50 is a specific variant with 50 layers, and it has been successful in various image recognition tasks. ResNet 50 is used on both datasets.

E. GoogLeNet[5]

GoogLeNet, also known as Inception v1, is a CNN architecture notable for its inception modules that allow the network to learn and combine features at multiple spatial scales. It uses 1x1 convolutions, along with larger convolutions, reducing computational complexity while capturing diverse features. GoogLeNet is used on both datasets.

F. VGG19[6]

VGG19 is a deep CNN architecture known for its simplicity and uniform architecture. It consists of 19 layers (16 convolutional and 3 fully connected layers) with small (3x3) convolutional filters throughout the network. VGG models have shown strong performance in image classification tasks. VGG19 is used on both datasets.

G. DenseNet 121[7]

DenseNet is a CNN architecture characterized by dense connections between layers. Each layer receives direct inputs from all preceding layers and passes its feature maps to all subsequent layers. DenseNet-121 is a specific variant with 121 layers, and its dense connectivity promotes feature reuse and gradient flow. DenseNet 121 is used on both datasets.

H. Efficient Net b0[8]

EfficientNet is a family of CNN architectures developed to achieve better performance with increased efficiency in terms of accuracy and computational resources. EfficientNet B0 is the baseline model with a compound scaling method that balances network depth, width, and resolution to optimize performance while minimizing computational cost. Efficient Net b0 is used on both datasets.

V. Methodology

A. CNN from scratch

I trained CNNs from scratch on dataset 1. Each layer had the following structure:

- First convolution layer had 3 input channels, 128 output channels with 5x5 filters.
- All other convolution layers had 128 input channels, 128 output channels wth 5x5 filters.
- All pooling layers were Max Pooling with kernel size = 2 and stride = 2.

The number of layers were varied from 2 layers of convolution and pooling followed by a fully connected layer to 7 layers of convolution and pooling followed by a fully connected layer. Batch size was 8, learning rate was 0.001, epochs was 16, loss function was cross entropy loss and optimizer was Adam.

B. Logistic Regression

I trained logistic regression models on dataset 1. A grid search was performed with the following hyperparameters and ranges:

- batch size: 16, 32, 64
- epochs: 10, 20, 30
- learning rate: 0.0001, 0.001, 0.01, 0.1, 1
- Optimizer: SGD, Adam

The loss function was cross entropy loss.

C. DeiT

1) Dataset 1: I froze the pre-trained weights and added the following to the end:

1) Linear layer from number of output of previous last layer to 512
2) ReLU
3) Dropout 0.3 of the nodes
4) Linear from 512 to 2

There were two transforms to the input image:

- default transform = transforms.Compose([transforms.Resize (256,interpolation=3), transforms.CenterCrop(224), transforms.ToTensor(), transforms.Normalize (IMAGENET DEFAULT MEAN, IMAGENET DEFAULT STD),])
- custom transform = transforms.Compose([transforms.Resize (256,interpolation=3), transforms.CenterCrop(224), transforms.ToTensor(), transforms.Normalize (calculated mean, calculated std),])

For the DeiT model I performed a grid search with the following hyperparameters and ranges:

- epochs: 4, 8, 16
- learning rates: 0.0001, 0.001, 0.01
- Optimizer: SGD, Adam
- Transform type: default, custom

Batch size was 8 and loss function was cross entropy loss.

2) Dataset 2: I froze the pre-trained weights and added the following to the end:

1) Linear layer from number of output of previous last layer to 512
2) ReLU
3) Dropout 0.3 of the nodes
4) Linear from 512 to 15

Batch size was 32, learning rate was 0.0001, epochs was 6, loss function was cross entropy loss, optimizer was Adam and transform type was custom.

D. Other models

For the remaining models (ResNet50, GoogLeNet, VGG19, DenseNet 121, Efficient Net b0) the following was repeated.

1) Dataset 1: I froze the pre-trained weights and added the following to the end:

1) Linear layer from number of output of previous last layer to 512
2) ReLU
3) Dropout 0.3 of the nodes
4) Linear from 512 to 2

Batch size was 8, learning rate was 0.0001, epochs was 16, loss function was cross entropy loss, optimizer was Adam and transform type was custom.

2) Dataset 2: I froze the pre-trained weights and added the following to the end:

1) Linear layer from number of output of previous last layer to 512
2) ReLU
3) Dropout 0.3 of the nodes
4) Linear from 512 to 15

Batch size was 32, learning rate was 0.0001, epochs was 6, loss function was cross entropy loss, optimizer was Adam and transform type was custom.

VI. Experimental Results

In this section, I show the results obtained and mention observations made from my code[9].

A. Logistic Regression

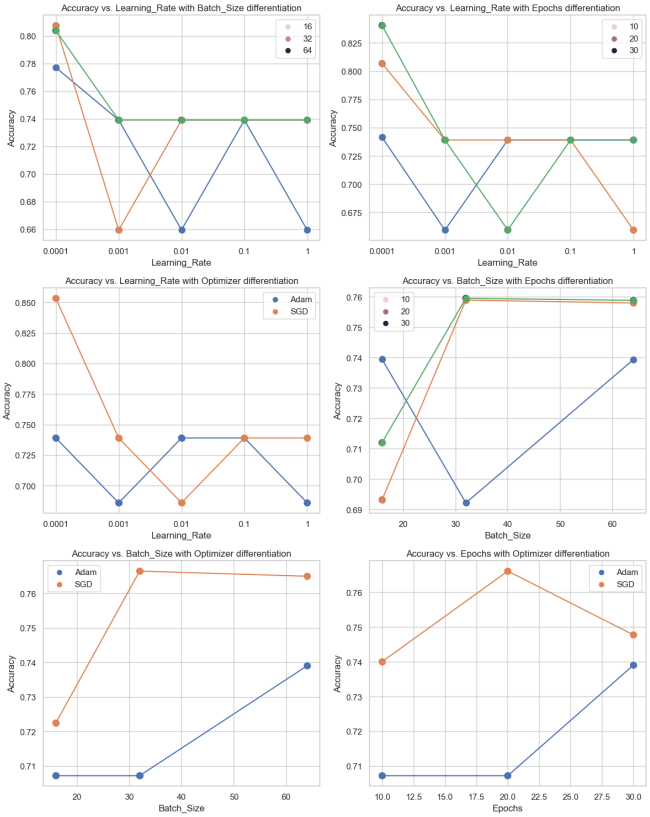


Fig. 3: Accuracy for Logistic Regression (Dataset 1)

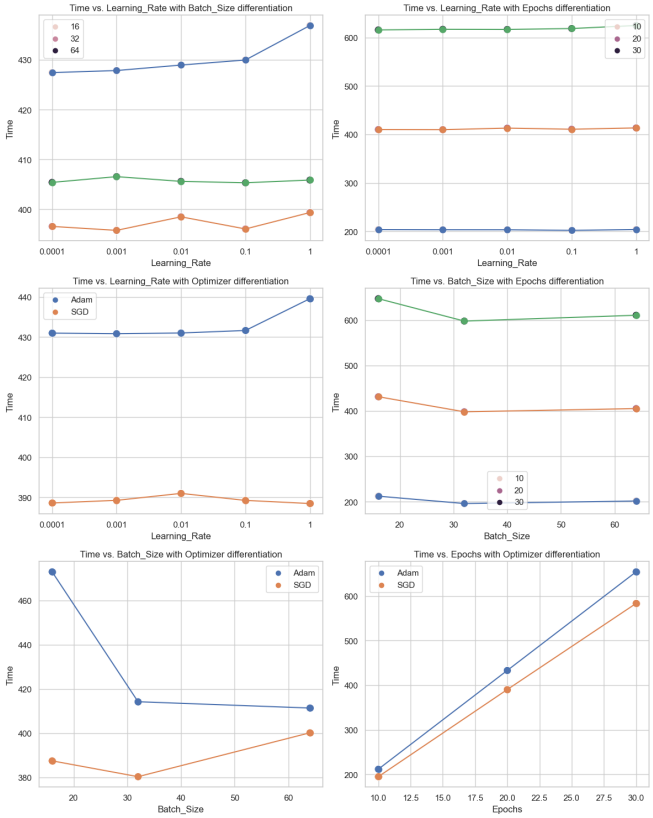


Fig. 2: Time taken to train for Logistic Regression (Dataset 1)

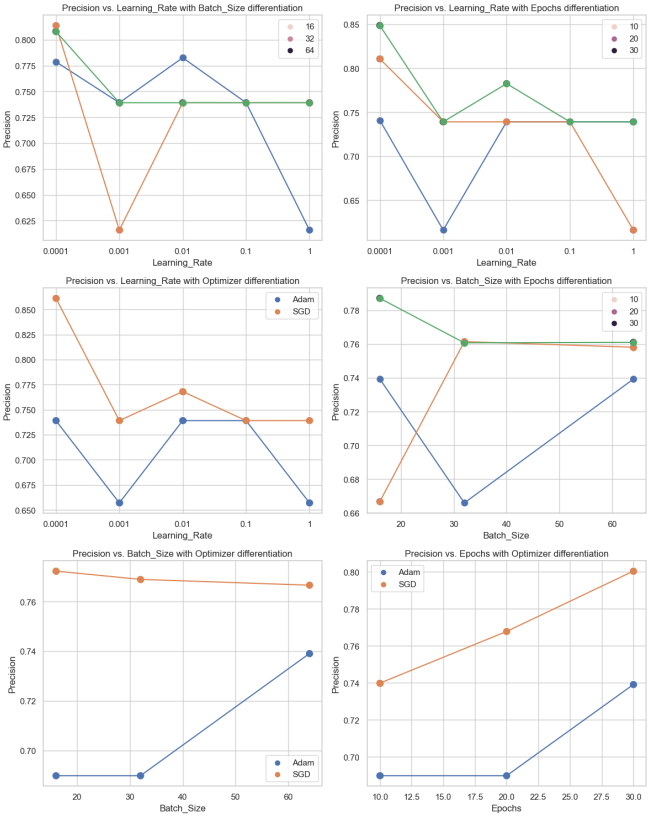


Fig. 4: Precision for Logistic Regression (Dataset 1)

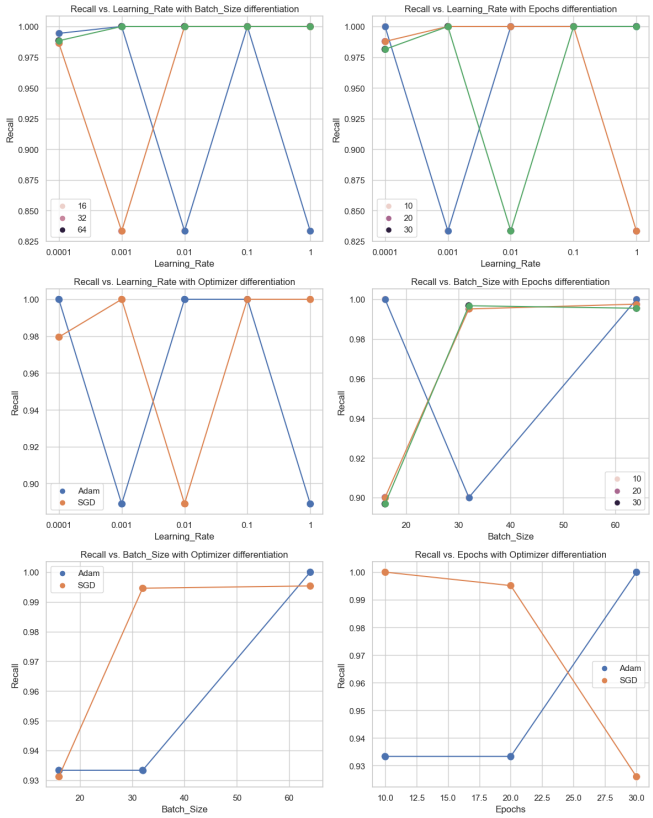


Fig. 5: Recall for Logistic Regression (Dataset 1)

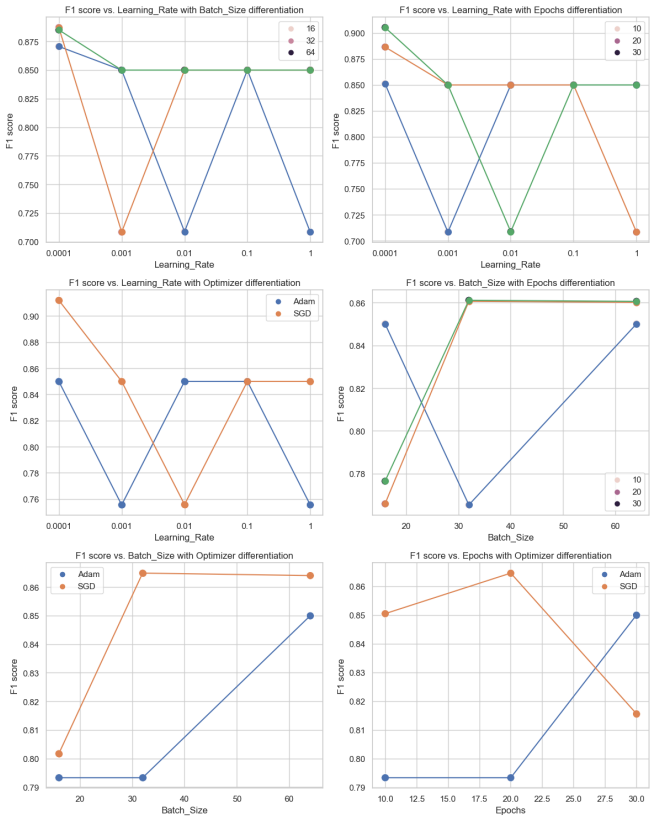


Fig. 6: F1 Score for Logistic Regression (Dataset 1)

Top 5 Ascending Time Taken:

	Time	Accuracy	Precision	Recall	F1 score	Batch_Size	Epochs	Learning_Rate	Optimizer
37	189.129552	0.739056	0.739056	1.0	0.849951	32	10	0.1000	SGD
1	189.161883	0.742489	0.741602	1.0	0.851632	16	10	0.0001	SGD
33	189.325735	0.739056	0.739056	1.0	0.849951	32	10	0.0010	SGD
35	189.966387	0.739056	0.739056	1.0	0.849951	32	10	0.0100	SGD
39	190.556179	0.739056	0.739056	1.0	0.849951	32	10	1.0000	SGD

Top 5 Descending Accuracy:

	Time	Accuracy	Precision	Recall	F1 score	Batch_Size	Epochs	Learning_Rate	Optimizer
21	582.546098	0.945064	0.959631	0.966318	0.962963	16	30	0.0001	SGD
51	569.976229	0.943348	0.956372	0.967480	0.961894	32	30	0.0001	SGD
41	378.212637	0.937339	0.963529	0.951220	0.957335	32	20	0.0001	SGD
81	594.528762	0.936481	0.959160	0.954704	0.956927	64	30	0.0001	SGD
71	399.724700	0.927897	0.930233	0.975610	0.952381	64	20	0.0001	SGD

Top 5 Descending Precision:

	Time	Accuracy	Precision	Recall	F1 score	Batch_Size	Epochs	Learning_Rate	Optimizer
25	577.637810	0.261803	1.000000	0.001161	0.002320	16	30	0.0100	SGD
41	378.212637	0.937339	0.963529	0.951220	0.957335	32	20	0.0001	SGD
21	582.546098	0.945064	0.959631	0.966318	0.962963	16	30	0.0001	SGD
81	594.528762	0.936481	0.959160	0.954704	0.956927	64	30	0.0001	SGD
51	569.976229	0.943348	0.956372	0.967480	0.961894	32	30	0.0001	SGD

Top 5 Descending Recall:

	Time	Accuracy	Precision	Recall	F1 score	Batch_Size	Epochs	Learning_Rate	Optimizer
0	230.200818	0.739056	0.739056	1.0	0.849951	16	10	0.0001	Adam
56	626.281371	0.739056	0.739056	1.0	0.849951	32	30	0.1000	Adam
64	201.059876	0.739056	0.739056	1.0	0.849951	64	10	0.0100	Adam
63	202.768922	0.739056	0.739056	1.0	0.849951	64	10	0.0010	SGD
62	203.246585	0.739056	0.739056	1.0	0.849951	64	10	0.0010	Adam

Top 5 Descending F1 Score:

	Time	Accuracy	Precision	Recall	F1 score	Batch_Size	Epochs	Learning_Rate	Optimizer
21	582.546098	0.945064	0.959631	0.966318	0.962963	16	30	0.0001	SGD
51	569.976229	0.943348	0.956372	0.967480	0.961894	32	30	0.0001	SGD
41	378.212637	0.937339	0.963529	0.951220	0.957335	32	20	0.0001	SGD
81	594.528762	0.936481	0.959160	0.954704	0.956927	64	30	0.0001	SGD
71	399.724700	0.927897	0.930233	0.975610	0.952381	64	20	0.0001	SGD

Fig. 7: Best performing logistic regression models (Dataset 1)

Something to note is that some models had high recall(1) and low accuracy, as they always only guessed one class.

B. CNN from scratch

	Time	Accuracy	Precision	Recall	F1 score	Layers	% trained
output	52804.279888	0.962232	0.971165	0.977933	0.974537	4	100.00
output	10939.655208	0.959657	0.969977	0.975610	0.972785	3	100.00
outputmodel2	4485.359551	0.944206	0.939294	0.988386	0.963214	2	6.25
output	3344.319836	0.914163	0.920442	0.967480	0.943375	6	6.25
output	3392.674581	0.890129	0.954151	0.894309	0.923261	5	6.25
output	3661.627886	0.739056	0.739056	1.000000	0.849951	7	6.25

Fig. 8: Best performing CNN models (Dataset 1)

Something to note is that single layer CNN gave an error as it required too much memory. Only CNN with 3 and 4 layers were fully trained since they could finish within reasonable times. For the remaining (2,5,6,7 layers), I trained it for only 1 epoch.

C. DeiT Grid Search

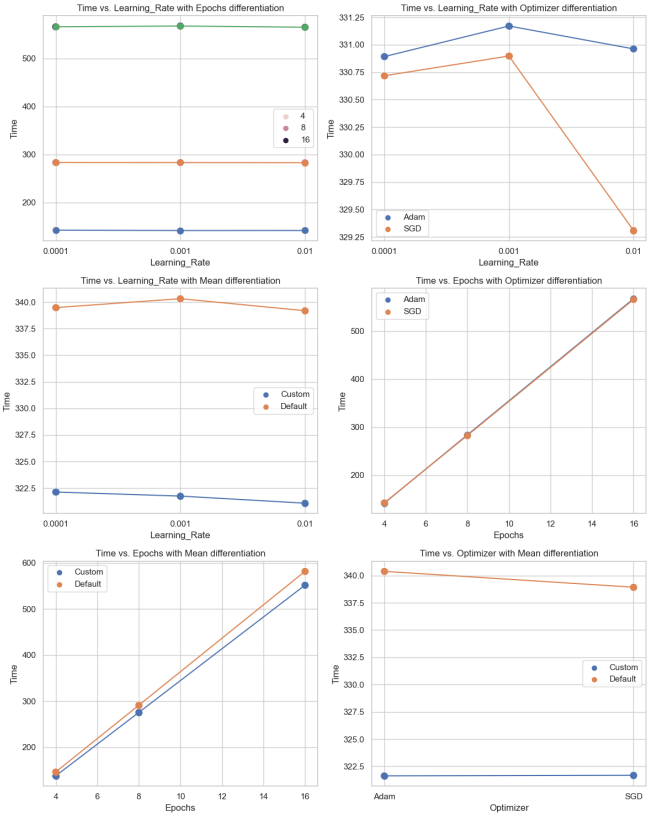


Fig. 9: Time taken to train for DeiT (Dataset 1)

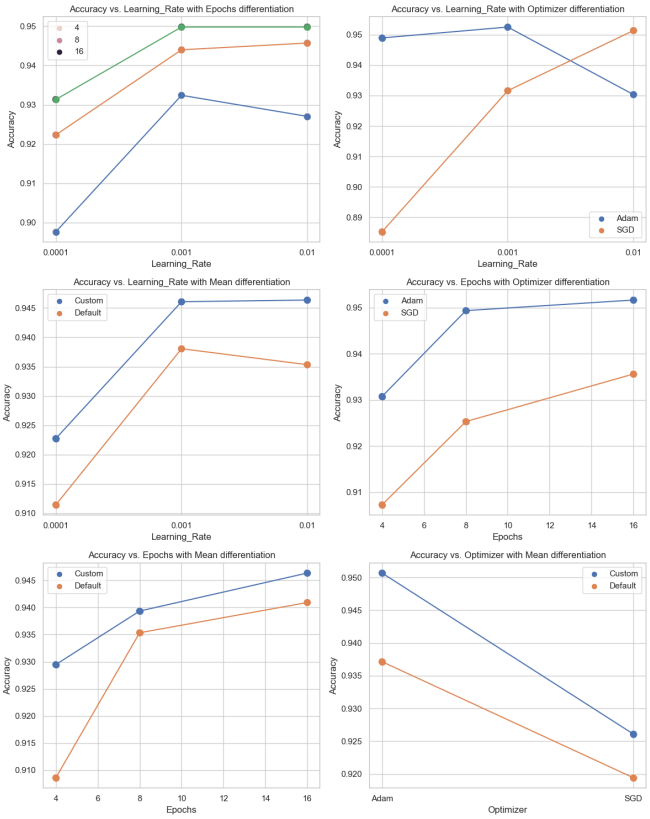


Fig. 10: Accuracy for DeiT (Dataset 1)

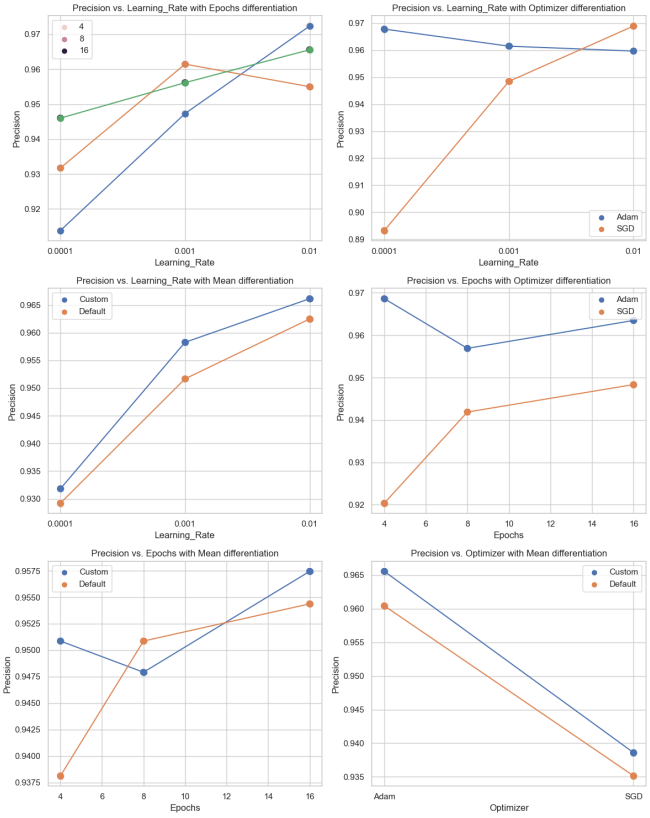


Fig. 11: Precision for DeiT (Dataset 1)

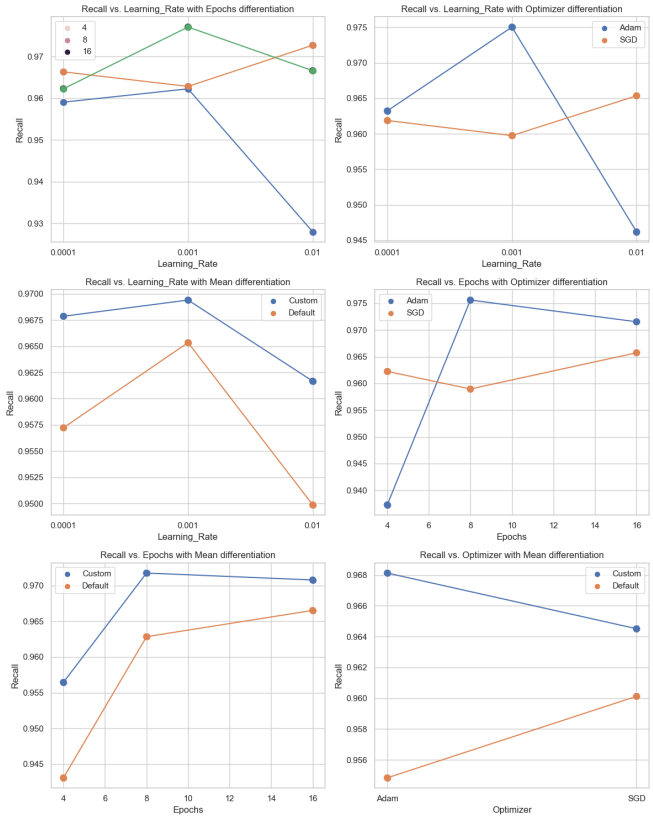


Fig. 12: Recall for DeiT (Dataset 1)

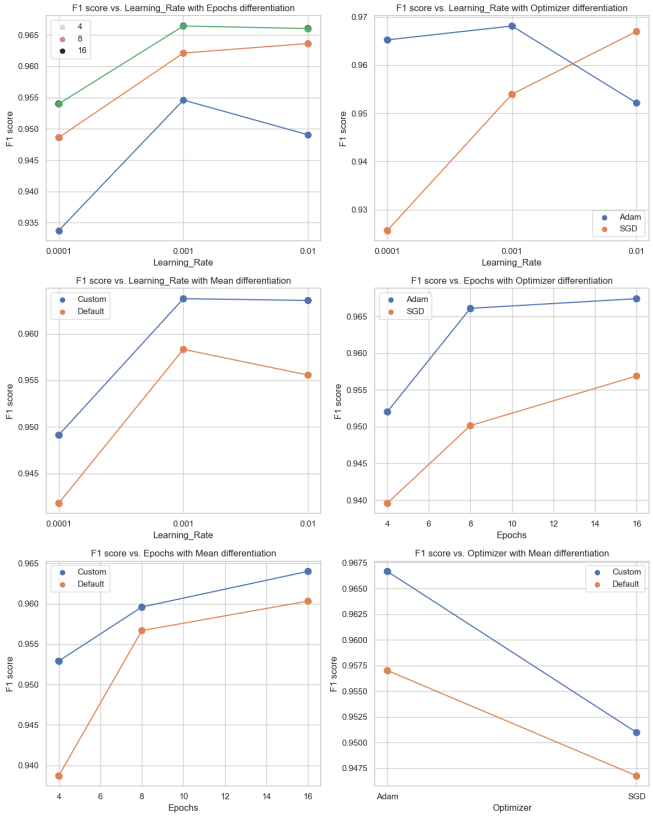


Fig. 13: F1 Score for DeiT (Dataset 1)

The custom mean and standard deviation performed better than the image net mean and standard deviation. This is why I have used the custom values for the particular dataset going forward.

D. Comparing pre-trained models

	Time	Accuracy	Precision	Recall	F1 score
densenet121	588.103437	0.959657	0.962500	0.983740	0.973004
outputDietModel	599.620598	0.959657	0.968894	0.976771	0.972817
ResNet50	589.977690	0.957082	0.972061	0.969803	0.970930
effnetb0	576.169114	0.956223	0.967667	0.973287	0.970469
googlenet	578.404567	0.951073	0.984337	0.948897	0.966292
vgg19	689.380888	0.945923	0.954442	0.973287	0.963772

Fig. 15: Best performing pre-trained models(Dataset 1)

In general, I noticed that the pre-trained models had very good performance with relatively little time to train. This is why I am only able to use these models for dataset 2 since it contains many more images.

	Time	Accuracy	Precision	Recall	F1 score
ResNet50	9740.341128	0.390178	0.096391	0.074327	0.052046
effnetb0	8805.336313	0.389201	0.114003	0.082181	0.066499
googlenet	8956.019262	0.388264	0.130006	0.073305	0.052059
outputDietModel	8812.900594	0.387678	0.167613	0.080327	0.066212
densenet121	9663.202121	0.384552	0.122500	0.083345	0.066295
vgg19	13012.023785	0.384083	0.078998	0.069193	0.043648

Fig. 16: Best performing pre-trained models(Dataset 2)

While the accuracy is significantly higher than random guessing(0.067), it could probably be higher if it could account for an image belonging to multiple classes at the same time.

VII. Discussion

A. Learning Outcomes

- Learnt how to train various models on image data.
- Built a better intuition on which models perform better and which hyperparameters have more impact on the outcome for these datasets.
- Improved skills in handling large amounts of image data.
- Improved data visualizing and analyzing skills.

B. Future Works

- Try more models with more grid searches to find better solutions.
- Implementing a way to handle one image belonging to multiple classes.

- With the help of segmented data and diagnosis, try to make a model that can explain why a person is diagnosed with something.

C. Conclusion

In this project, we explored various models and saw how these models performed on the two medical image datasets. Further study in this area can help reveal better models and maybe even the reasoning behind why these models perform better. This can then help us avoid wasting time searching for good models and allow us to focus on tuning hyperparameters for suitable models.

REFERENCES

[1] K. G. M. Kermany, Daniel; Zhang, "Labeled optical coherence tomography (oct) and chest x-ray images for classification," 2018.

[2] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, July 2017.

[3] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers distillation through attention," 2021.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[7] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[8] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[9] R. Juyal, "GitHub PRML Project," 2023.