Sparse Medical Image Analysis

Thesis submitted by

Siddhant Sharma
2019EE10531

under the guidance of

Prof. Jayadeva, Indian Institute of Technology Delhi

in partial fulfilment of the requirements
for the award of the degree of

Bachelor of Technology

Department Of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY DELHI

November 2022

THESIS CERTIFICATE

This is to certify that the thesis titled **Sparse Medical Image Analysis**, submitted by **Siddhant Sharma (2019EE10531)**, to the Indian Institute of Technology, Delhi, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Jayadeva
Professor
Dept. of Electrical Engineering
IIT-Delhi, 600 036

Place: New Delhi

Date: 27th November 2022

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Jayadeva for providing his invaluable guidance, comments and suggestions.
I would also like to thank our medical collaborators, Dr. Vineet Batta and Dr. Somesh Gupta. They, alongside their group, provided vital medical data and insights.
I would also like to acknowledge the other members of Prof. Jayadeva's research group who participated in group meetings and provided valuable assistance along the way.

ABSTRACT

KEYWORDS: Image Analysis, Data Augmentation, Low Complexity Models, Data Imbalance.

This thesis contains a self contained, all-in-one solution to the application of machine learning algorithms to medical data.

It has been observed frequently that standard machine learning algorithms are not able to work out of the box for many medical and clinical tasks. This, in turn, generates a need for having machine learning engineers to help tune these standard machine learning algorithms just for the use on medical datasets. This pipeline aims to do away with the need for any external help when running machine learning algorithms by medical professionals.

This pipeline tackles issues such as data imbalance, dirty data, lack of data features, highly complex input amongst other issues that plague medical datasets which make them tough to crack using standard algorithms.

Contents

List of Tables

List of Figures

List of Algorithms

ABBREVIATIONS

IITD	Indian Institute of Technology, Delhi
DNN	Deep Neural Network
AIML	Artificial Intelligence and Machine Learning
ML	Machine Learning
CV	Computer Vision
TP	True Positive
TN	True Negative
FN	False Negative
FP	False Positive
MLP	Multilayer Perceptron
SVM	Support Vector Machine
MCM	Minimal Complexity Machine
STM	Support Tensor Machine
SHTM	Support Higher-Order Tensor Machine
MCTM	Minimal Complexity Tensor Machine
ResNet	Residual Network
ConvNet	Convolutional Neural Network
LCNN	Low Complexity Neural Network
ViT	Visual Transformer
STN	Spatial Transformer Network
IR	Infrared
NIR	Near Infrared
RGB	Red Green Blue
NDVI	Normalized Difference Vegetation Index

Chapter 1

INTRODUCTION

The use of Artificial Intelligence and Machine Learning (AIML) for medicine and clinical reasons is one of the most nobel and important applications of AIML. Despite this, the advancement in medical applications of AIML has not been as optimal as one would hope. The clear reason for the same is the additional hurdles medical data poses to standard machine learning (ML) and computer vision (CV) algorithms. Medical data has the following attributes which make it hard for standard off-the-shelf ML and CV algorithms to work:

- Dirty unregularised data, with lots of outliers.
- Highly complex data, such as X-RAY and Infrared Images.
- Sparsity of meaningful labelled data and incomplete annotations.
- Skewed datasets since certain classes (such as common diseases) have more labelled data.

Additionally, for clinical tasks, we can't afford errors and thus there is an added constraint to have the highest possible accuracy, recall, precision and F1-score. Let us assume the prediction and ground truth values are as such:

Predicted Outcome

Actual Value	**p**	**n**	**Sum**
p′	(TP) True Positive	(FN) False Negative	P′
n′	(FP) False Positive	(TN) True Negative	N′
Sum	P	N	

Thus we have the following definitions for the gauging metrics for our ML models.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

To add to the complexity of the problem, most of the medical tasks are multi-class classification or regression. This additional (**fifth**) complication renders many ML algorithms inapplicable or infeasible due to time and space complexity.

And finally, a **sixth** complication is that sometimes datasets have specialised features only for a few datapoints and not all. For example, some RGB images may have the infrared counterpart whereas most don't. While we can ignore this specialised information to make standard algorithms work, it makes more sense to extend the remaining data to also get these specialised features automatically.

With a plethora of traits which make medical data and clinical tasks incompatible with most ML algorithms, there is a dire need for an ALL-IN-ONE solution. In this thesis, we propose a pipeline that handles all the unfavourable traits independently. In addition to solving the above five main problems, it is also independent of the domain of the clinical task.

We test this domain independence by testing it on two different datasets and tasks.

- Implant Identification: This dataset consists of X-Ray images of implants and we are tasked with identifying the company / type of implant.
- Skin Lesion Classification: This dataset consists of RGB (some with corresponding IR) images of skin lesions and we are tasked with classifying the disease.

Both the above datasets have a lot of classes and also have a large data imbalance between classes. This makes them the perfect candidates to test our pipeline as they have all of the above negative attributes we are planning to tackle.

This modular pipeline allows end users to utilise our pipeline for any task with data that showcases any (or all) of the above negative attributes.

In the following chapters, we discuss one module of the pipeline per chapter before tying it all together and then stating the final conclusions and results.

Chapter 2

Classifier

2.1 Introduction

We begin discussing the modular pipeline with the underlying framework we can utilise. For ease of discussion, we take the case of classifiers but it can be directly extended to regression models as well.

A classifier in machine learning is an algorithm that categorizes data into one or more of a set of classes. This is different to a regressor which assigns a numerical value to each data point. Most classifier architectures can be extended to regression tasks as well.

The two main classifier types have been listed in Sections 2.2 and 2.3.

2.2 Binary Classifiers

Binary classification is the task that is carried out by binary classifiers. They assign a class (out of a possible of exactly 2) to the datapoint using some form of model.

Since they have 2 possible output classes, they are not very useful for the vast tasks that are needed to be performed in the medical domain. We tackle this shortcoming in Section 2.3.

2.2.1 Tree Type Neural Network

Tree Type Neural Network [JDC02] is a multilayer perceptron (MLP) classifier network. The architecture for the MLP is a pyramidal delayed perceptron [MRRM90]. We adapt this network to get the following attributes:

- Less to no hyperparameters while keeping accuracy high.
- Interpretable weights are learnt.
- Minimizing risk and lowers complexity.
- Efficient use of sparse structure of images.

Architecture

The architecture is built using the following algorithm:

1. The parent neuron forms halfplane decision regions and tries to classify the dataset.
2. All positive (and negative) samples that are incorrectly classified are corrected by using a bias that is generated from a (created) child neuron trained similarly.
3. This recursion goes on until a limit is reached or all data samples are classified correctly.

Figure 2.1 showcases the final model that may be generated using the above algorithm.

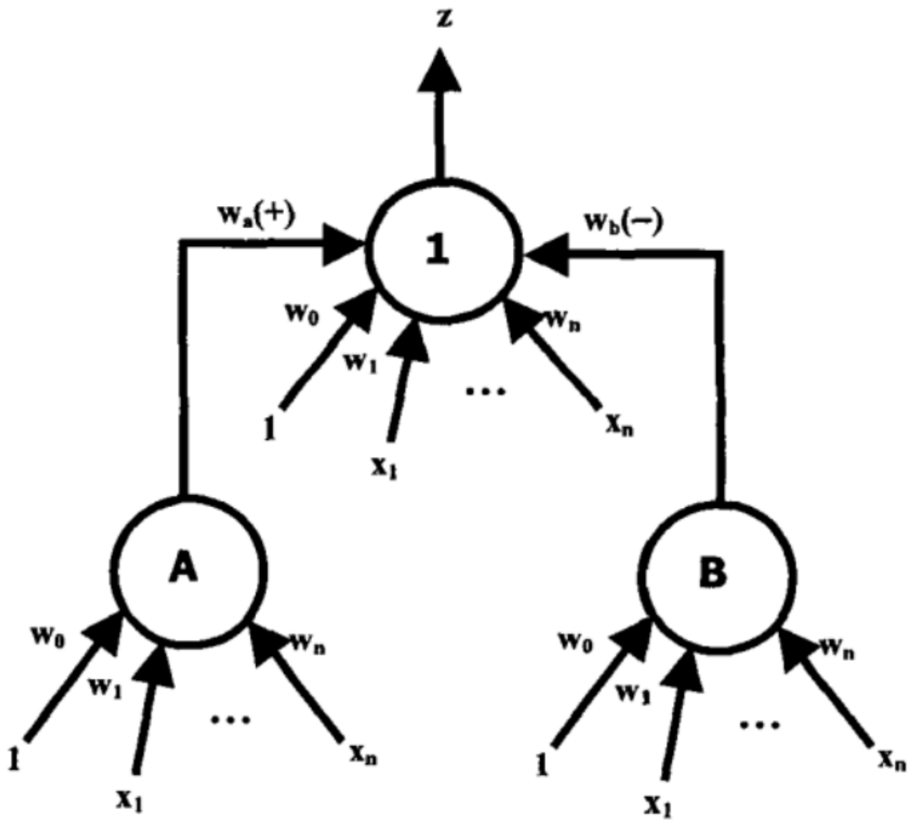

Figure 2.1: Tree architecture with a parent neuron and 2 child neurons [JDC02]

Each neuron of the tree (for example 1, A, B in figure 2.1) corresponds to a binary classifier. This binary classifier was originally only tested for Support Vector Machines (SVMs) [SSB+97, AW99, Vap99a] but now it has been extended to Minimal Complexity Machines (MCMs) [Jay15, SJSP17, JSPS20], Support Tensor Machines (STMs) [XJH+18, CBKW18, CBYW22], Support Higher-Order Tensor Machines (SHTMs) [KGP12, HHCY13] and even Neural Networks [Sch15, ZMH+94].

We even worked on a novel tensor version of MCM dubbed Minimal Complexity Tensor Machine (MCTM) which extended the MCM theory to tensor data.

The formulation for the classifiers are as follows, where M is total number of data points, w is the weights, b is the bias, ξ_i is the slack variable, C controls the *hardness* of the classifier, y_i is the ith data point's (X_i) target value.

$$\textbf{SVM} : \min_{w,b,\xi} J(w,b,\xi) = \frac{1}{2}||w||_F^2 + C \cdot \sum_{m=1}^{M} \xi_m$$

$$\text{where } y_m(w^T \cdot X_m + b) \geq 1 - \xi_m \text{ and } \xi_m \geq 0 \ \forall m \in [1, M]$$

$$\textbf{STM} : \min_{w,b,\xi} J(w,b,\xi) = \frac{1}{2}||w||_F^2 + C \cdot \sum_{m=1}^{M} \xi_m$$

$$\text{where } y_m(\langle w, X_m \rangle + b) \geq 1 - \xi_m \text{ and } \xi_m \geq 0 \ \forall m \in [1, M]$$

$$\textbf{SHTM} : \min_{w,b,\xi} J(w,b,\xi) = \frac{1}{2}||w||_F^2 + C \cdot \sum_{m=1}^{M} \xi_m$$

$$\text{where } y_m(\langle w, X_m \rangle_R + b) \geq 1 - \xi_m \text{ and } \xi_m \geq 0 \ \forall m \in [1, M]$$

$$\textbf{MCM} : \min_{w,b,\xi} J(h,w,b,\xi) = h + C \cdot \sum_{m=1}^{M} \xi_m$$

$$\text{where } h \geq y_m(w^T \cdot X_m + b) \geq 1 - \xi_m \text{ and } \xi_m \geq 0 \ \forall m \in [1, M]$$

$$\textbf{MTCM} : \min_{w,b,\xi} J(h,w,b,\xi) = h + C \cdot \sum_{m=1}^{M} \xi_m$$

$$\text{where } h \geq y_m(\langle w, X_m \rangle + b) \geq 1 - \xi_m \text{ and } \xi_m \geq 0 \ \forall m \in [1, M]$$

Here $A \cdot B$ denotes the dot product [LSS09], $\langle A, B \rangle$ denotes the inner product [Moo95], $\langle A, B \rangle_R = \langle A_R, B_R \rangle$ where A_R is the rank 1 decomposed matrix of A [HHCY13].

Learning

The learning for the entire network is straight forward. Each neuron's weights are learned independently starting from the children nodes. The parent node's w_a and w_b as shown in Figure 2.1 are also learnt when learning the weights ($w_i \ \forall i \in [0, N]$) of the parent node.

Class	Actual Output	Desired Output
C1	0	0
C2	1	1
C3	0	1
C4	1	0

Table 2.1: Class division based on parent neuron

The data used by each child neuron to learn its weights is according to table 2.2. The learning of each neuron is according to the innate classifier that the neuron contains such as STM, MCM, etc., using either gradient methods or numerical optimisation methods.

Class	Desired Output of A	Desired Output of B
C1	0	Don't Care
C2	Don't Care	0
C3	1	0
C4	0	1

Table 2.2: Child neuron's desired output

The classes mentioned in table 2.2 are defined in table 2.1.

Tree Pruning

As mentioned in Section 2.2.1, we need to define a method to stop the tree growth or else it will overfit the data and return poor test set performance. We introduce another novelty to the classifier network by pruning the tree automatically using risk minimisation [Vap99b].

- Empirical Risk:
  - It is the risk associated with misclassifying data points.
  - Most methods aim to just minimize empirical risk.
  - $R_{emp} = \sum_{i=0}^{M} |y_i - \hat{y_i}|$ where $\hat{y_i}$ is predicted output.
- Structural Risk:
  - It is the risk associated with a complex (and possibly overfitting) model.
  - Methods generally use a regularization method to keep this risk low.
  - $R_{struct} = \frac{\epsilon}{2}(1 + \sqrt{1 + \frac{4 \cdot R_{emp}}{\epsilon}})$ where $\epsilon = \frac{4}{l} \cdot ht \cdot (\log(\frac{2l}{h} + 1) - \log \eta)$ [Vap99b].

The above risk formulations are for binary classifiers only.

Using the above definitions of the two subtypes of risk, we can develop a formulation for the total risk for our tree.

$$R_{total} = R_{tree,emp} + R_{tree,struct} = R_{parent,emp} + \sum_{node \in tree} R_{node,struct}$$

Thus, we have the pruning heuristic as ***IF*** *the Total Risk of Parent without Child A is less than the Total Risk of Parent with Child A* ***THEN*** *Don't create Child A*.

Interpretability

Since all neurons are trained independently, we can look at the weights of each neuron to interpret what was learnt by that neuron to increase accuracy for the overall model.

To ensure that the weights of each neuron are in the same space as the input images (and thus would be interpretable by humans), we add additional constraints to the weights in addition to the constraints as stated by equations of classifiers in 2.2.1.

$$lb_i \leq w_i \leq ub_i \ \forall i$$

Here lb_i and ub_i are lower and upper bounds for the ith data feature over all data points.

Using this constraint, we can identify and analyse important (and unimportant) features by viewing the activation of the weights as they are in the same space as the inputs.

Results

We tested this model independently, so as to test the feasibility of it for our final ALL-IN-ONE pipeline. Let us first have a look at the accuracy on standard datasets.

Model	Accuracy
Tree Type	0.851
MLP	0.829
Linear	0.782

Table 2.3: Tree Type on Statlog Heart Data Set

Model	Accuracy
Tree Type	0.989
Convolutional Network	0.990
MLP	0.984

Table 2.4: Tree Type on MNIST 3 vs 8 Data Set

From the Tables 2.3 and 2.4 we can see that the tree type neural network performs just as good if not better than standard ML algorithms while also ensuring no time is lost in hyperparameter tuning.

In addition to the lack of an array of hyperparameters to tune, we also have interpretable weights as seen in Figure 2.2. It is clear that each neuron learns something unique and exactly what the neuron learns is also very interpretable. For example, the rightmost child neuron exclusively learns whenever the input is a 4.

2.3 Multi-class Classifiers

Multi-class classification is the task that is carried out by multi-class classifiers. They assign a class (out of a possible of any arbitary natural number) to the datapoint using some form of model.

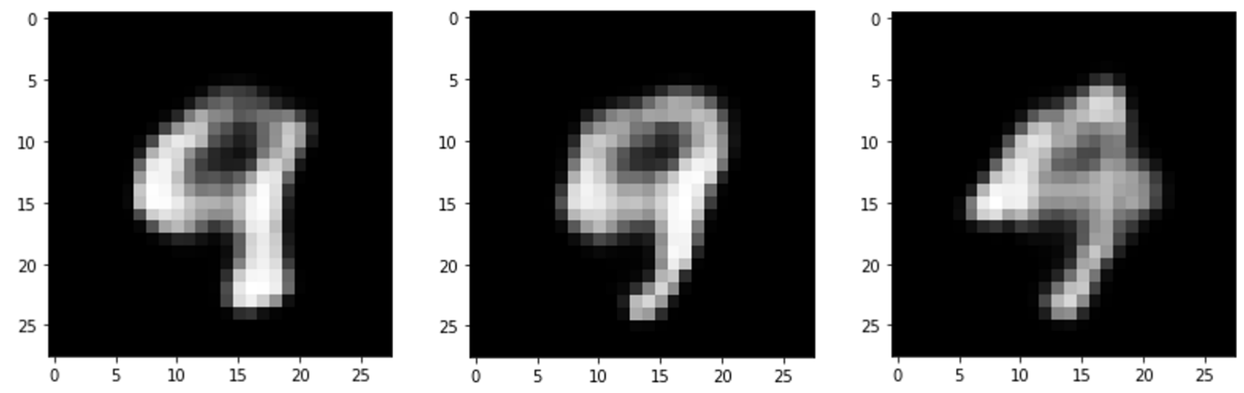

Figure 2.2: Leftmost child, root parent and rightmost child neuron's weights

This type of model is ideal for medical use, since the data generally consists of multiple classes and categories.

2.3.1 Innate Multi-class Classifiers

Classifiers that can inherently predict more than 2 classes are defined as innate multi-class classifiers. These can be used directly for multi-class classification tasks without any modifications. While a lot of techniques exist such as Decision Trees [PP18], K-Nearest Neighbours [FH89], Naive Bayes [VBT14], etc., we will be looking at neural network techniques.

Neural Networks can be extended to multi-class classification tasks by having N neurons in the output layer instead of just one. Thus, the neuron with the largest value predicts the classification of the input as that neuron's class.

Residual Network

When dealing with convolutional neural networks (ConvNet) [AMAZ17], we tend to have a drop in accuracy on increasing the number of layers. Residual Network (ResNet) [HZRS15] alleviates that problem by creating skip connections that skip layers as seen in Figure 2.3.

After multiple ResNet blocks, the ConvNet has fully connected neural network blocks which naturally give way to a multi-class classifier.

ResNet results

We can create ConvNets which are exactly the same as the ResNet counterpart just without the skip connections. Let us compare the results of having and removing these skip connections which are the salient feature of a ResNet.

We see in Table 2.5 that increasing the number of layers has a positive affect on the

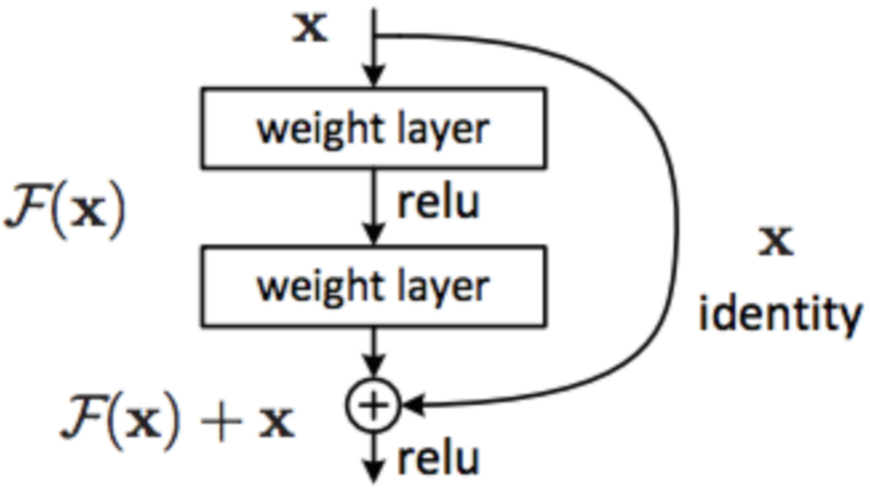
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.3: Singular ResNet Block with skip connection [HZRS15]

Model	Accuracy
34 Layer ConvNet	71.46
34 Layer ResNet	75.81
152 Layer ResNet	78.53

Table 2.5: ResNet Accuracy on ImageNet [HZRS15]

accuracy of the model for ResNet whereas we know empirically that it has a negative effect on ConvNets. Additionally, ResNet has a better accuracy than ConvNet even for the same layer structure.

2.3.2 Binary to Multi-class Classifiers

To use Binary Classifiers for clinical tasks, we need to extend it to Multi-class Classification. We can use the following heuristic methodologies to do the same.

- One-vs-All: For an N classes dataset, we generate N binary classifier models.
- One-vs-One: For an N classes dataset, we generate $\frac{N \cdot (N-1)}{2}$ binary classifier models.

One-vs-All

We train a binary classifier per class, with the data points of that class as positive samples and all other data points as negative samples. Thus, we generate N different binary classifiers, each which outputs a confidence value of the (test) data point being in that model's class. The model with the highest confidence is chosen and that model's class is assigned to the data point.

While this technique only increases the time and space complexity a multiplicative factor of $O(n)$ on average, it suffers from certain drawbacks such as:

- Scale of confidence outputs may not be the same across models and thus certain models (and thus classes) will be chosen more frequently than necessary.
- Due to the large negative samples, we observe various negatives of skewed data [LMHD19] even if the original dataset wasn't.

One-vs-One

In this technique, we split a multi-class classification problem into binary classification problems as in One-vs-All. However, we split the multi-class classification problem into one binary classification problem for each of the N classes vs all of the other classes taken one at a time. Thus we have $\binom{N}{2} = \frac{N\cdot(N-1)}{2}$ different binary classification problems. We then tally the votes of each binary classifier, and the class with the highest votes is chosen as the final prediction of the ensemble.

Clearly One-vs-One is extremely poor in its time complexity as it takes an additional multiplicative factor of $O(n^2)$ and is often rejected in lieu of One-vs-All technique despite the slightly more robust prediction process.

2.4 Conclusion

We now test the techniques discussed above to our own datasets to gauge their potential as the backbone model for our pipeline.

2.4.1 Experimentation on Medical Datasets

Tree Type Neural Network

This technique had two solving methodologies to test, namely:

- Numerical Optimisation: It was unable to handle such a large dataset and thus was declared infeasible for the current datasets.
- Gradient Descent: It was unable to provide any meaningful results. This approach takes away one of the salient features of Tree Type Neural Network, which is the lack of hyperparameters.

Considering the negatives of both these approaches, lack of generalisability and the added time complexity of One-vs-All or One-vs-One, this architecture was removed from consideration for the backbone of the pipeline.

Residual Network

Model	Accuracy
3-layer ConvNet	0.53
18-layer ResNet	0.91
50-layer ResNet	0.85

Table 2.6: Accuracy on majority Implant X-Ray classes

We can see from Table 2.6 that an 18-layer ResNet has the highest accuracy on our dataset. Additionally, we see that ResNet is far superior to a simple ConvNet layers. Amongst ResNets, 18 layers performed better due to a more simplistic and less complex structure. The benefit of having low complexity is given in Chapter 3.

2.4.2 Backbone of Pipeline

Due to better generalisability and empirical results on the medical datasets, we decided to utilise, the innate multi-class classifier, ResNet-18 for the backbone of our pipeline.

Chapter 3

Complexity of the Network

3.1 Introduction

Working with medical data, we are bound to get complex data such as IR images. This will render the ResNet backbone classifier with complex weights and structure too.

We know that model complexity hurts test set performance [WS06, Yin19] and thus we need to bring down the complexity of the model. This can be achieved using the following ways:

- Model Structure: We can simplify the model structure. Since we have already decided on utilising ResNet architecture, we can reduce the number of layers to curtail the complexity of the model.
- Model Weights: We can also make multiple weights redundant by reducing them to 0. This lowers number of connections between hidden layers that are relevant. Thus the overall complexity of the model is lowered.

The following section focuses on the second point of the above reasons since the first point is trivially solved by utilising a ResNet architecture with a lower number of layers.

3.2 Low Complexity Neural Network

Controlling the complexity of neural networks has been studied for a long time, however the current standard methods include regularising the weights of the network [IJ18, SP22]. This implies that the weights are small but it does not equate to the vanishing of many weights. To ensure that many weights are reduced to 0 (essentially taking the L0-norm), we utilise the Low Complexity Neural Network (LCNN) Loss [JPS$^+$17].

3.2.1 Formulation

The LCNN Loss is formulated as

$$LCNN = \sum_{i=0}^{M} \sum_{j=0}^{K} (net_i^j)^2$$

where net_i^j is the final layer activation of the jth class (out of K) for the ith sample (out of N). This can be extended to any hidden layer neurons as well.

3.2.2 Result

The LCNN Loss has an effect similar to an L0-norm and thus many weights tend to 0. The vanishing of weights further lowers the complexity of the model, making it more generalisable and faster to train.

We can check Figure 3.1 and 3.2, where all models were trained on the standard UCL dataset.

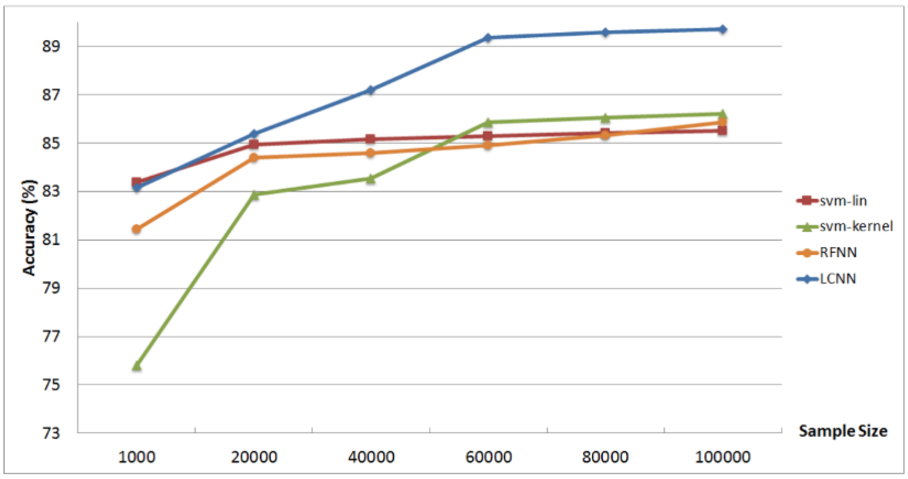

Figure 3.1: LCNN Accuracy on UCL Dataset [JPS+17]

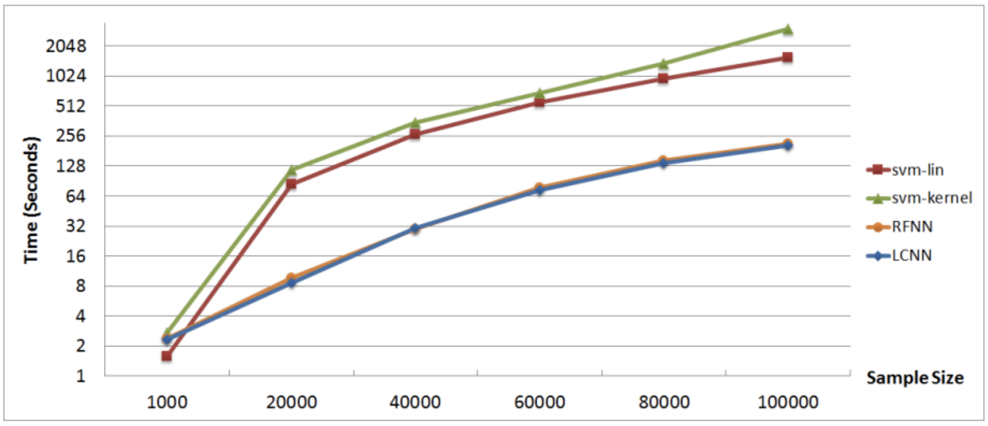

Figure 3.2: LCNN Time Taken on UCL Dataset [JPS+17]

It is clear that the Low Complexity Neural Network Loss helps increase the accuracy (3.1) of the model and also decreases the time taken (3.2) to train the model.

3.3 Conclusion

We now test the techniques discussed above to our own datasets to gauge their potential as the backbone model for our pipeline.

3.3.1 LCNN Experimentation on Medical Datasets

Dataset	LCNN?	Accuracy
Skin Lesions	No	0.35
Skin Lesions	Yes	0.43
Implant X-Ray	No	0.47
Implant X-Ray	Yes	0.83

Table 3.1: Accuracy results on full datasets

The poor accuracy is because this test was done in isolation of the remaining pipeline and just used the data directly without any augmentation or cleaning. However, from Table 3.1 we can conclude that adding LCNN loss helped the accuracy and thus we include it in our pipeline alongside less complex structures from Chapter 2.

3.3.2 Pipeline Inclusion

We used ResNet-18 (one of the smallest layer ResNet architecture) to combat the model structural complexity. On top of that, we have utilised the LCNN loss to drive many weights to 0 which further reduced the complexity of the model.

Both of these additions to the pipeline are modular and generic, while also bettering accuracy and time to train.

Chapter 4

Image Transformers

4.1 Introduction

A pertinent issue with medical data is the apparent dirtiness and lack of order. Unlike with standard datasets, real life medical datasets have uncentered images which are rarely aligned rotationally.

To fix this dirty data and make the images become more standardised, and the classifier does not need to learn the various ways an image can be dirty. This relaxation of computational overhead from the classifier helps in speeding up the entire process and increases accuracy as well.

We have two main methodologies to transform the images,

- Spatial Transformers: They transform the image spatially, such as with rotation, translation, etc.
- Visual Transformers: They transform the image in a more generic way by splitting it in patches, and allow us to attend to the important information of the image.

4.2 Spatial Transformers

Spatial Transformer Netowrks (STNs) [JSZK15] increase a model's spatial invariance against transformations such as translation, scaling, rotation, cropping, shearing, etc. They accomplish spatial invariance by adaptively changing their input to a canonical and standard form, thus enhancing classification performance.

4.2.1 Architecture

The main block of an STN is given in Figure 4.1 which can be placed into any existing ConvNet either just after the input layer or somewhere in the middle of the ConvNet. We shall be using it just after the input layer so we can also interpret how the image changes.

The datapoint is used to predict a transformation matrix via the localisation net. That is used on itself to produce the transformed image using the following transformation algorithm,

$$V = A_\theta \cdot U$$

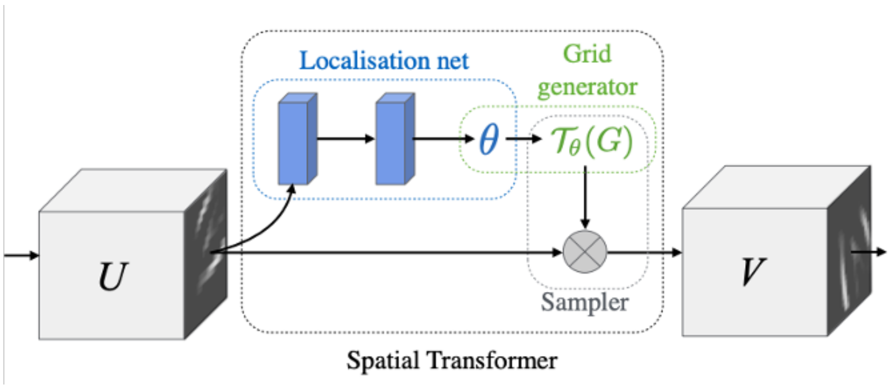

Figure 4.1: Spatial Transformer Network Architecture [JSZK15]

where V is the output image's coordinates, $U = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ is the input image's coordinates, $A_\theta = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{pmatrix}$ is the transformation matrix.

Spatial transformers networks can be trained directly using backpropagation and do not need any special tuning or hyperparameters. This feature of the STN makes it ideal for our pipeline.

4.2.2 Results

STNs fix any spatial transformation such as rotation, cropping, etc. as can be seen in Figure 4.2.

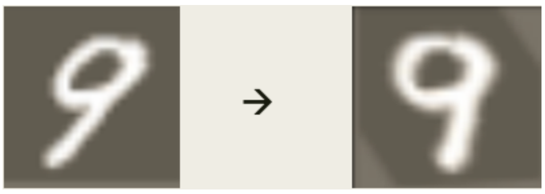

Figure 4.2: Input Image (left) and STN Corrected Image (right)

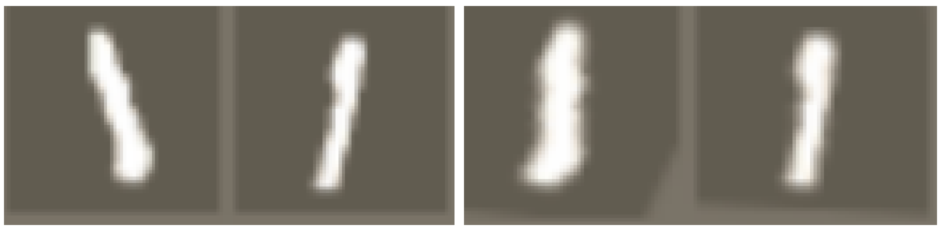

Figure 4.3: Input Image (both left) and STN Corrected Image (both right)

From Figure 4.3 we can see that STN only corrects the data's orientation IF it is needed and keeps it the same when it already is in corerct orientation (Image 1 corresponds to Image 3, and Image 2 to Image 4 in Figure 4.3).

Therefore, for all common types of distortions, STNs will work perfectly for our dataset and fit nicely with our pipeline as well.

4.3 Visual Transformers

Visual Transformers (ViT) [DBK+20] are more expensive to train as compared to Spatial Transformers but are robust to more types of input distortions such as adversarial patches, scale change, permutations, along with the distortions that Spatial Transforms handle.

4.3.1 Architecture

The salient feature is visualised in Figure 4.4, which is the source of the added complexity.

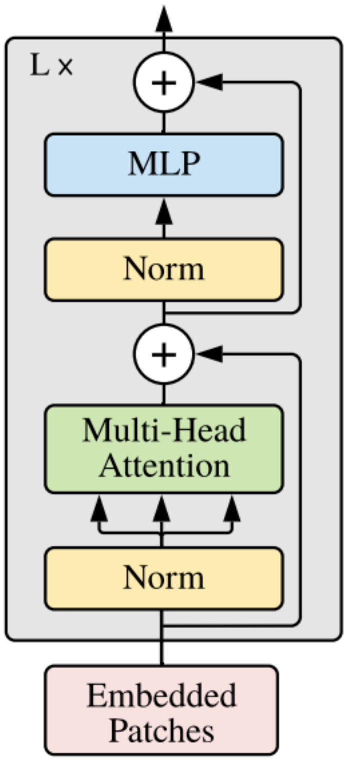

Figure 4.4: Transformer Encoder of Visual Transformer [DBK+20]

This encoder is then used as given in Figure 4.5 to create the entire visual transformer model.

We can replace the MLP head with a ConvNet if needed, to integrate the Visual Transformer in our pipeline if we decide to add it.

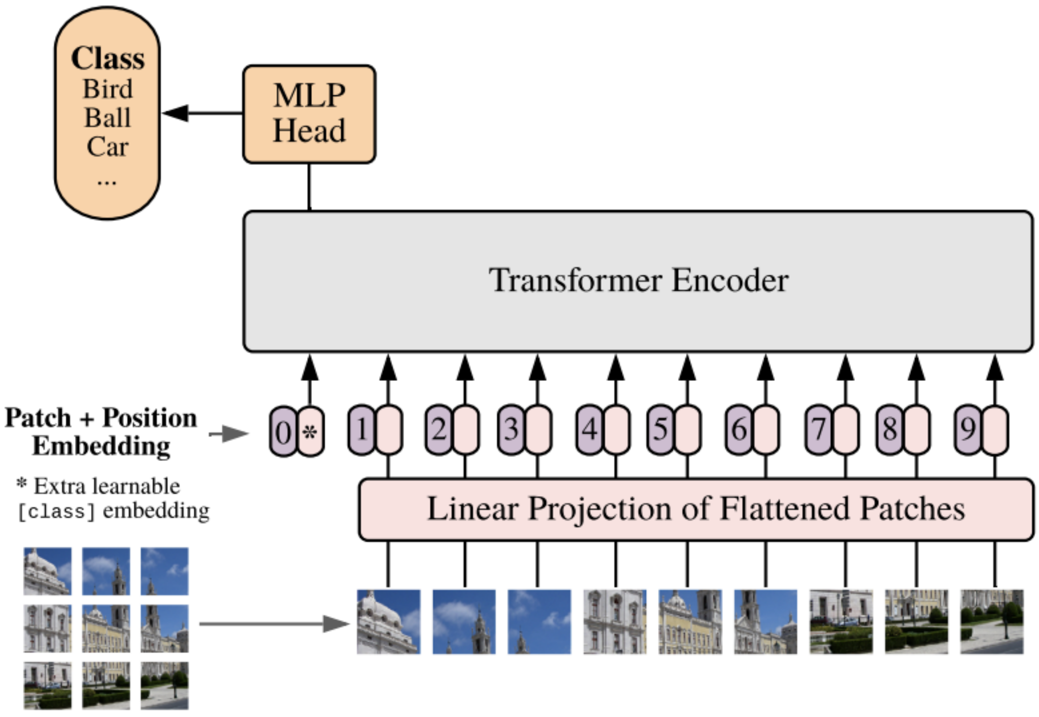
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches

Figure 4.5: Visual Transformer Architecture [DBK+20]

4.3.2 Results

We can use a ViT either independently or combined with another ConvNet. For the following results, we use it independently to showcase the extent it can work.

Dataset	ViT Accuracy	ResNet Accuracy
ImageNet	88.55	87.54
CIFAR-10	99.50	99.37

Table 4.1: Comparision of ViT and ResNet [DBK+20]

We see in Table 4.1 that the visual transformer does perform better than a simple ResNet.

4.4 Conclusion

We now test the techniques on our own data to figure out the viability for our medical pipeline.

The ViT, despite being superior in terms of accuracy to STN, is needlessly complex which goes against the low complexity pipeline we are trying to build. Additionally, the distortions that STN can't handle will be extremely rare in medical data and most realistic distortions should be handled without invoking ViTs.

4.4.1 STN Experimentation on Medical Datasets

Applying STN helped standardise the dataset as seen in Figure 4.6 and helped decrease training time as well.

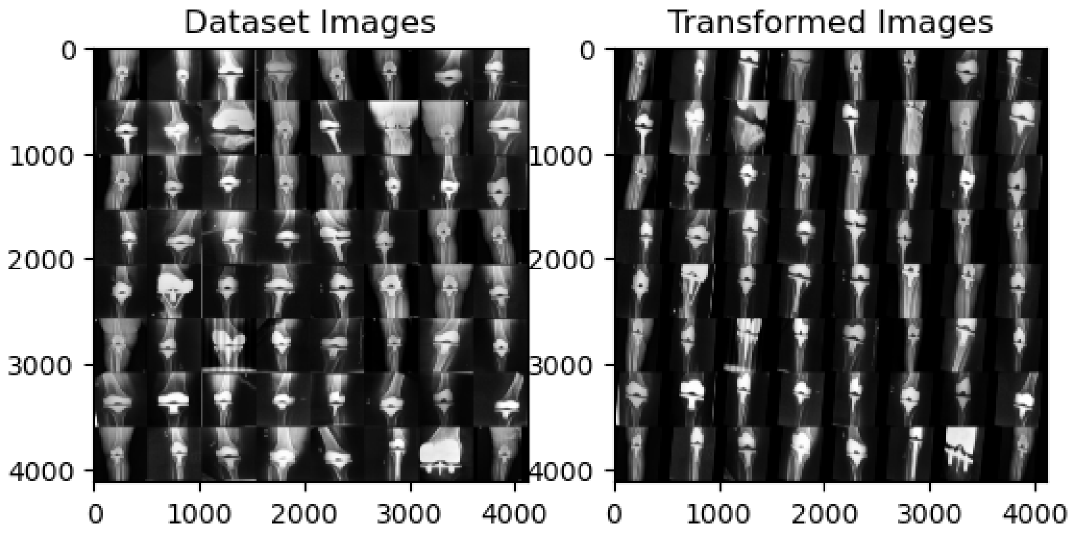

Figure 4.6: Spatitial Transformations on Implant X-Ray dataset.

The images on the left in 4.6 have random zooms and tilts. However, the transformed images all have similar positioning in the image and a similar zoom.

4.4.2 Pipeline Inclusion

We use Spatial Transformer Networks in our pipeline to clean the data before it is worked on by the classifier. STNs are ideal as they are lightweight, reduce load and complexity from the classifier, and can be used modularly much more easily than ViTs. Plus, an STN is almost gaurenteed to never hamper the accuracy of a model and thus has been chosen for our pipeline.

Chapter 5

Data Augmentation

5.1 Introduction

In this chapter, we tackle the case of lack of data (in one or more classes). This issue is faced very commonly in clinical tasks since more common classes (such as diseases) are not as important to classify as compared to the more rarer classes. But, the catch-22 situation is that these rarer, more important classes will always have lesser data samples. And thus, they are bound to have poorer accuracy as compared to the common classes, especially using standard ML algorithms.

Data augmentation is a method for synthesising new data points from existing data points by modifying the current data. While most famous techniques utilise changing the available datapoints in the same space as the input, EigenSample [JSS18] finds similar data points in a lower dimension before projecting those newly generated artificial data points back to the input space.

EigenBag and EigenSample are techniques that not only alleviate this data imbalance by generating more data for the rarer classes, they can also boost the data samples for all classes. This ultimately implies that the total accuracy of the model should empirically increase.

5.2 EigenSample

The main idea behind EigenSample is to generate artificial datapoints without disturbing the eigen-structure of the data. Thus the data augmentation algorithm proposed generates actual artificial data which least changes the underlying structure of the dataset.

5.2.1 Algorithm

We look at the entire detailed algorithm 1 which delves into the exact methodology used to obtain new data points that don't change the eigen-structure of the instrinsic data.

Algorithm 1 EigenSample Algorithm [JSS18]

```
Require: Dataset $A \in R^{(M,N)}$, labels $Y \in \mathbb{N}$.
    $A^i = A^i - \mu$ where $\mu = \sum_{j=0}^{M} A^j$.
    Find correlation matrix $\Sigma$.
    Find eigenvalues $\lambda$ of $\Sigma$.
    Select $k$ eigenvalues so $\frac{\sum_{j=1}^{k} \lambda_j^2}{\sum_{j=1}^{n} \lambda_j^2} \geq 90\%$.
    Find projection matrix $G = [e^1, \cdots e^k]$.
    Project data using $X = G \cdot A$.
    Find cluster centers for $X$ and join $X^i$ to cluster centers.
    Pick mid point of line joining $X^i$ and cluster center as artificially generated data point.
    Find label using a classifier in dimension of $X$.
    Project all $X^i$ and new data to original dimension, $Z = G(\mu - \lambda) + (\gamma - \delta)$ where
    $\begin{pmatrix} \mu \\ \lambda \\ \gamma \\ \delta \end{pmatrix} = pinv(\begin{pmatrix} -(G^TG + 1/|C|) & G^TG & G & -G \\ G^TG & -(G^TG + 1/|C|) & -G & G \end{pmatrix} \cdot \begin{pmatrix} x^i \\ -x^i \end{pmatrix}$
    Add the mean back to the new data, $Z = Z + \mu$ and return it.
```

5.2.2 Results

Take a look at Figure 5.1 which showcases the benefits of using EigenSample for creating artificial data. The generated dataset actually looks cleaner that the original one.

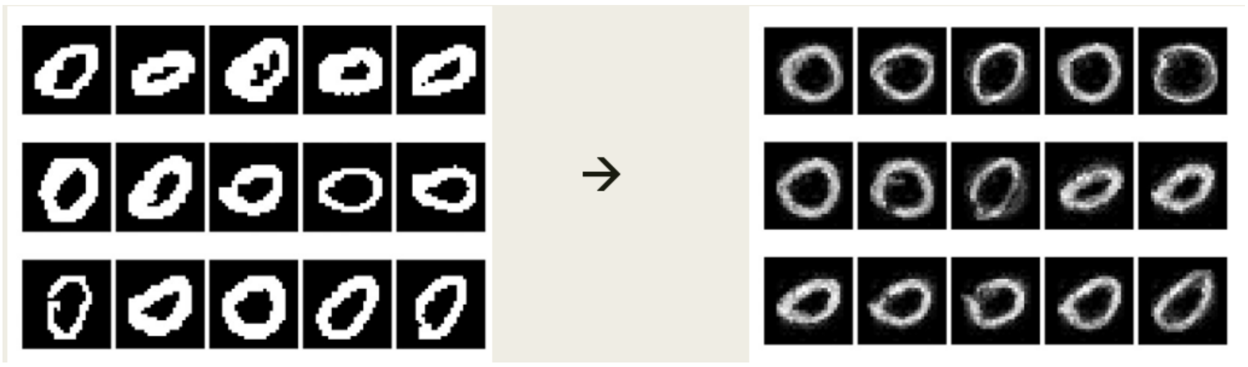

Figure 5.1: Original Dataset (left) and the Artificially Generated Dataset (right) [JSS18]

Clearly, Eigen-Technique is a much superior data augmentation technique compared to the various other techniques [BTR15] as instead of letting bad input data ruin the output, the output is actually bettered.

5.3 EigenBag

EigenBag is an extension of EigenSample which generates more samples via bagging. EigenSample simply doubles the training data set, but EigenBag repeatedly performs the Eigen-

Sample algorithm on subsets of the data that were chosen using various combinations.

5.3.1 Algorithm

As mentioned above, EigenSample would only double the data but EigenBag can generate $r \cdot \binom{M}{r}$ artificial similarly distributed samples. Here M is the total datapoints in the dataset and r is the number of datapoints we are taking at a time to apply EigenSample to.

Algorithm 2 EigenBag Algorithm

```
Require: Dataset $A \in R^{(M,N)}$, labels $Y \in \mathbb{N}$, $r$ datapoints selected.
  $B = r$ random samples from $A$.
  if cosine similarity between $A$ and $B >$ threshold then
      $A^i = A^i - \mu$ where $\mu = \sum_{j=0}^{M} A^j$.
      Find correlation matrix $\Sigma$.
      Find eigenvalues $\lambda$ of $\Sigma$.
      Select $k$ eigenvalues so $\frac{\sum_{j=1}^{k} \lambda_j^2}{\sum_{j=1}^{n} \lambda_j^2} \geq 90\%$.
      Find projection matrix $G = [e^1, \cdots e^k]$.
      Project data using $X = G \cdot A$.
      Find cluster centers for $X$ and join $X^i$ to cluster centers.
      Pick mid point of line joining $X^i$ and cluster center as artificially generated data point.
      Find label using a classifier in dimension of $X$.
      Project all $X^i$ and new data to original dimension, $Z = G(\mu - \lambda) + (\gamma - \delta)$ where
  $\begin{pmatrix} \mu \\ \lambda \\ \gamma \\ \delta \end{pmatrix} = pinv(\begin{pmatrix} -(G^TG + 1/|C|) & G^TG & G & -G \\ G^TG & -(G^TG + 1/|C|) & -G & G \end{pmatrix} \cdot \begin{pmatrix} x^i \\ -x^i \end{pmatrix}$
      Add the mean back to the new data, $Z = Z + \mu$.
      if cosine similarity between $Z$ and $A >$ threshold then
          Augment dataset $A$ with $Z$
      end if
  end if
```

The entire algorithm is given by Algorithm 2. The algorithm presented utilises numerical optimisation but the projection of X to Z can be done using gradient descent methods as well.

Additionally, the projection of X to Z can be done with care, ensuring that Z is constraint to the ranges of input as A. For example, if $A[:, 0]$ is always in between $[0, 1]$ then we will have a constraint on $Z[:, 0]$ such that it is smaller than 1 and larger than 0. This constraint is straight forward to add in the numerical optimisation method given in 2 but more complex in the gradient descent approach as we will need to truncate the values if they go out of range every single step.

5.4 Conclusion

We saw the amazing results from both EigenBag and EigenSample. Even though EigenBag tends to be very costly in terms of time complexity, it is vital to generate a lot of data samples. The main limitation of EigenSample to only double the dataset rendered it not very useful for our applications of adding a data augmentation module in our pipeline. We can combat the main drawback of EigenBag by switching to a gradient method of learning the matrices rather than mathematically doing so.

5.4.1 EigenBag Experimentation on Medical Dataset

We used EigenBag on the raw data. From a quick analysis, we can see that the Skin Lesions dataset has a much higher variance than the Implant X-Ray dataset. Therefore, the results of EigenBag on Implant X-Ray in Figure 5.2 are better than the Skin Lesions one in Figure 5.3.

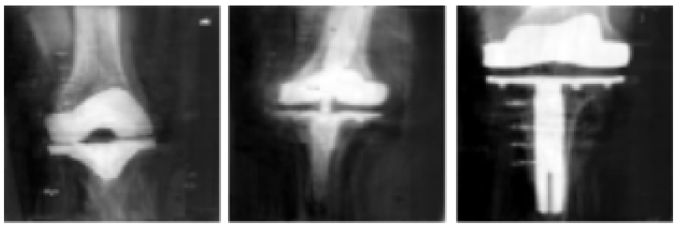

Figure 5.2: EigenBag Data Augmentation on Implant X-Ray

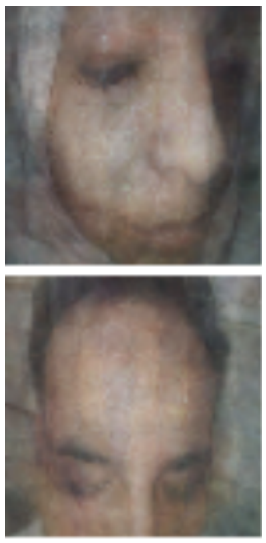

Figure 5.3: EigenBag Data Augmentation on Skin Lesions

The augmentation on Implant X-Ray dataset 5.2 is sharper and looks more like actual images as compared to the Skin Lesions 5.3 which looks very hazy.

However, since EigenBag works well on Implant X-Ray dataset as it is more standardised already as compared to Skin Lesions dataset, we can be sure that using EigenBag after the data cleaning step of the pipeline will generate good datapoints for Skin Lesions dataset as well.

5.4.2 Inclusion in Pipeline

Looking at the sheer importance of additional data samples, especially in the medical field, we add EigenBag to the overall pipeline. However, the numerical optimisers take too long and thus we need to go ahead with the gradient descent version of EigenBag.

This addition to the pipeline allows us to quickly find new data points for the classes that matter which are not related to any of the other data samples.

Chapter 6

Infrared Images

6.1 Introduction

A lot of times, medical datasets do not contain consistent features for their data. That implies that some data points are feature-rich, while others are feature-deficit. So far, it is common practise to truncate the feature-rich samples to the same space as the feature-deficit. However, it makes more sense to lift up the feature-deficit samples to match the feature-rich ones, so that we don't lose any information.

We propose a methodology of generating IR images (thus more features) for all those samples that don't have an associated IR image along with the regular RGB features. For this task, we can only use the Skin Lesion dataset and not the Implant X-Ray images.

This task will be performed before any other operation of the pipeline and thus can easily be removed (or added) to the workflow without needing to change any other aspect of the pipeline.

6.2 RGB2IR

Converting RGB images to Infrared Images [ASD+21] has shown great improvements in accuracy for various tasks.

6.2.1 Algorithm

We utilise a UNet ConvNet [RFB15] that takes in an RGB image and outputs the corresponding IR image. This is trained independently and before the remaining pipeline is trained as the loss function is completely different for this section of the pipeline.

The architecture for the UNet is given by Figure 6.1 and is trained using a Mean-Squared-Error Loss function on the output image and the ground truth IR image for all the data samples that have a corresponding IR image.

Please note that the additional feature that is learnt by this algorithm need not be an IR image and can be an X-Ray or something else entirely.

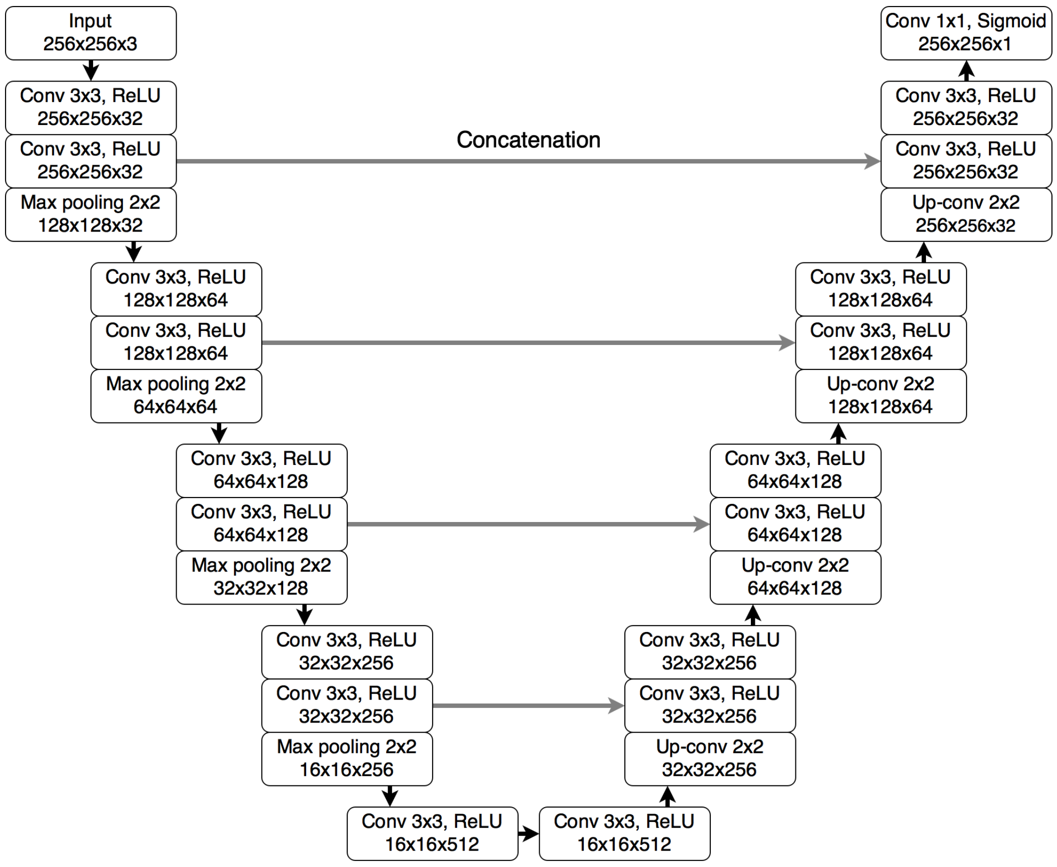

Figure 6.1: UNet Architecture for RGB2IR

6.2.2 Results

We can see from Figure 6.2 that machine learning techniques can accurately predict NIR images and Normalized Difference Vegetation Index (NDVI) images (which, as seen, are also additional features for a data sample like IR images).

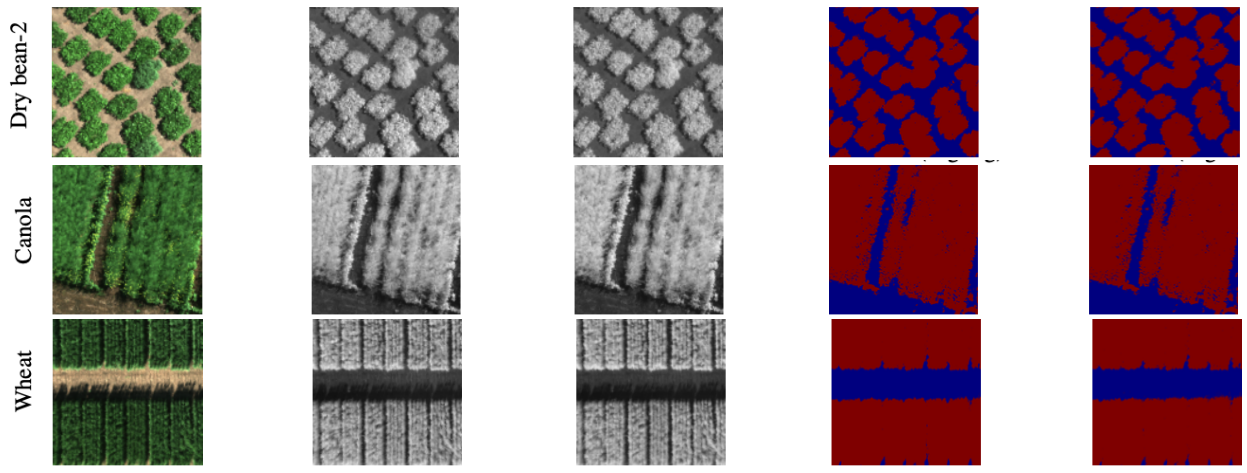

Figure 6.2: RGB, Real NIR, Pred NIR, Real NDVI, Pred NDVI respectively [ASD^{+}21]

From Figure 6.3 and 6.4 we can then see how the UNet ConvNet works in converting regular RGB images to IR images for our Skin Lesion Dataset which has a few data samples with both RGB and IR images, while the rest are only RGB images.

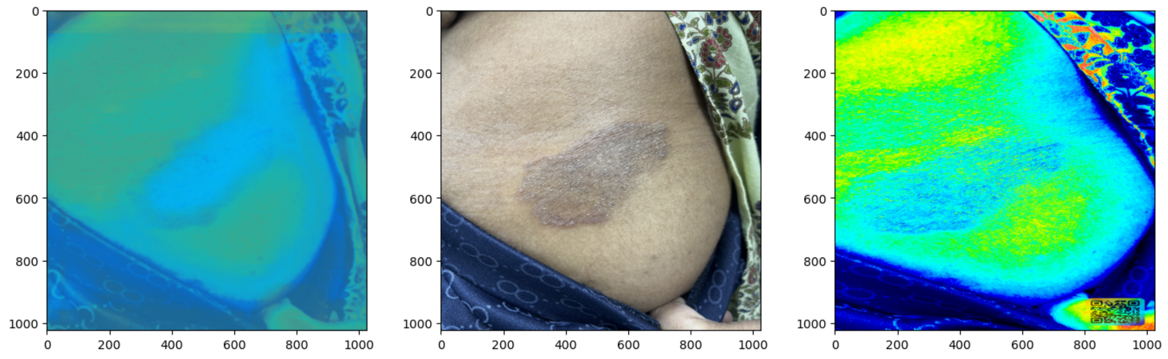

Figure 6.3: Predicted IR image (left), actual image (middle) and actual IR image (right)

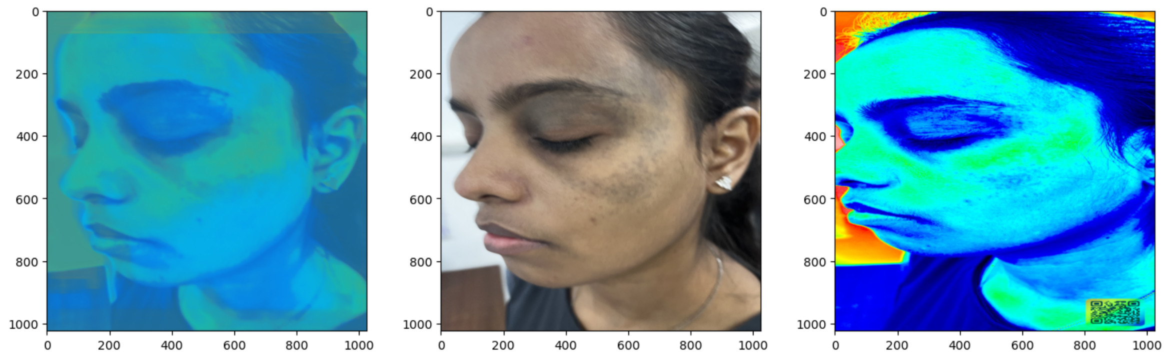

Figure 6.4: Predicted IR image2 (left), actual image2 (middle) and actual IR image2 (right)

Both of the current results suffer from lack of data and more data is being generated to create a more robust RGB to IR image converter. The current model struggles in masking background and auxiliary parts of an image but is fairly well tuned for masking the actual area of concern. The pipeline is in place and can be used for any dataset for doing its task.

6.3 Conclusion

Thus we see that Chapter 6 took a different approach as the methodology proposed here was very niche. Regardless, this methodology will help us tackle clinical tasks even better without having to wait a long time for IR images and without wasting all the previous data we already have collected.

Therefore, we include RGN2IR in our pipeline as another modular technique that can be used whenever relevant.

Chapter 7

Pipeline

7.1 Introduction

This chapter binds together all the previous chapters 2,3,4,5,6 to explain the entire pipeline devised for the medical and clinical tasks. As can be seen from the results, clearly this pipeline is quite generic and works well on a diverse set of tasks.

7.2 Description

The entire pipeline has the following components:

- Underlying model for classification/regression 2.
- Methodology to lower complexity and increase generalizability 3.
- Data cleaning algorithms 4.
- Data augmentation techniques 5.
- Features enhancement for the dataset 6.

The entire pipeline can be seen from the schematic in Figure 7.1. The low complexity model machinery is not expanded as the pipeline is robust to changes in it. We currently use ResNet and LCNN Loss as major backbones for the machinery.

7.3 Conclusion

Having all the techniques together, we see the following results as stated in Table 7.1.

Dataset	Accuracy
Implant X-Ray	0.97
Skin Lesion	0.49

Table 7.1: Results via entire pipeline

We perform substantially better using the entire pipeline than if we did not, as seen from the previous results in Chapters 2,3,4,5,6.

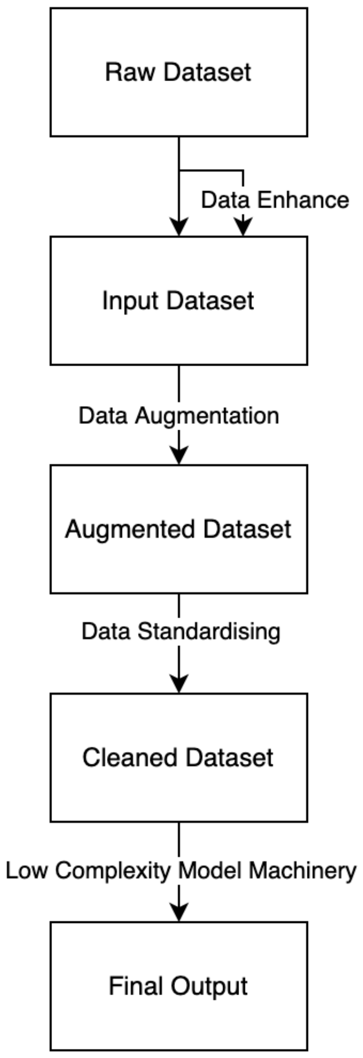

Figure 7.1: Final Devised Pipeline

The low comparative score for the Skin Lesion Dataset is due to the lower quality data augmentation for it. Work to increase the accuracy for it is underwhy and part of future work.

We can have minor changes in the ordering of the pipeline due to its modularity, and thus the data augmentation step can come after data cleaning module. This switch is important as seen in Chapter 5 if the original data is very dirty since EigenBag requires some form of standardisation before it can be applied.

Chapter 8

Discussion and Conclusion

This thesis outlines the creation (and various testing done) of an ALL-IN-ONE pipeline that has primarily been created for image datasets and for clinical tasks. It consists of various modular components that can be added or removed from the pipeline depending on the need of the dataset and task.

However, as seen from all the modular components, we can utilise this pipeline for a lot of other tasks as well. This pipeline can work well on image segmentation tasks (such as cement segmentation tasks [ZSG+22]), object recognition tasks and even for non image datasets.

The creation of this pipeline will allow for an easy-to-use solution for most medical tasks and thus medical professionals don't need to invest a lot of time to achieve superior machine learning performance.

Bibliography

[AMAZ17] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[ASD$^+$21] Masoomeh Aslahishahri, Kevin G. Stanley, Hema Duddu, Steve Shirtliffe, Sally Vail, Kirstin Bett, Curtis Pozniak, and Ian Stavness. From rgb to nir: Predicting of near infrared reflectance from visible spectrum aerial images of crops. In *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 1312–1322, 2021.

[AW99] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[BTR15] Paula Branco, Luis Torgo, and Rita Ribeiro. A survey of predictive modelling under imbalanced distributions, 2015.

[CBKW18] Cong Chen, Kim Batselier, Ching-Yun Ko, and Ngai Wong. A support tensor train machine, 2018.

[CBYW22] Cong Chen, Kim Batselier, Wenjian Yu, and Ngai Wong. Kernelized support tensor train machines. *Pattern Recognition*, 122:108337, 2022.

[DBK$^+$20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[FH89] Evelyn Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247, 1989.

[HHCY13] Zhifeng Hao, Lifang He, Bingqian Chen, and Xiaowei Yang. A linear support higher-order tensor machine for classification. *IEEE Transactions on Image Processing*, 22(7):2911–2920, 2013.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[IJ18] Nusrat Ismoilov and Sung-Bong Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10:648, 11 2018.

[Jay15] Jayadeva. Learning a hyperplane classifier by minimizing an exact bound on the vc dimension1. *Neurocomputing*, 149:683–689, 2015.

[JDC02] Jayadeva, A.K. Deb, and S. Chandra. Binary classification by svm based tree type neural networks. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, volume 3, pages 2773–2778 vol.3, 2002.

[JPS$^+$17] Jayadeva, Himanshu Pant, Mayank Sharma, Abhimanyu Dubey, Sumit Soman, Suraj Tripathi, Sai Guruju, and Nihal Goalla. Learning neural network classifiers with low model complexity, 2017.

[JSPS20] Jayadeva, Sumit Soman, Himanshu Pant, and Mayank Sharma. Qmcm: Minimizing vapnik's bound on the vc dimension. *Neurocomputing*, 399:352–360, 2020.

[JSS18] Jayadeva, Sumit Soman, and Soumya Saxena. Eigensample: A non-iterative technique for adding samples to small datasets. *Applied Soft Computing*, 70:1064–1077, 2018.

[JSZK15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2015.

[KGP12] Irene Kotsia, Weiwei Guo, and Ioannis Patras. Higher rank support tensor machines for visual recognition. *Pattern Recognition*, 45(12):4192–4203, 2012.

[LMHD19] Aisyah Larasati, Apif M Hajji, and Anik Dwiastuti. The relationship between data skewness and accuracy of artificial neural network predictive model. *IOP Conference Series: Materials Science and Engineering*, 523:012070, 07 2019.

[LSS09] Seymour Lipschutz, Murray R Spiegel, and Dennis Spellman. *Schaum's Outline of Vector Analysis.* McGraw Hill Professional, 2009.

[Moo95] Gregory H. Moore. The axiomatization of linear algebra: 1875-1940. *Historia Mathematica*, 22(3):262–303, 1995.

[MRRM90] G. Martinelli, L.P. Ricotti, S. Ragazzini, and F.M. Mascioli. A pyramidal delayed perceptron. *IEEE Transactions on Circuits and Systems*, 37(9):1176–1181, 1990.

[PP18] Harsh Patel and Purvi Prajapati. Study and analysis of decision tree based classification algorithms. *International Journal of Computer Sciences and Engineering*, 6:74–78, 10 2018.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[SJSP17] Mayank Sharma, Jayadeva, Sumit Soman, and Himanshu Pant. Large-scale minimal complexity machines using explicit feature maps. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(10):2653–2662, 2017.

[SP22] Claudio Filipi Gonç alves Dos Santos and João Paulo Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Computing Surveys*, 54(10s):1–25, jan 2022.

[SSB+97] B. Scholkopf, Kah-Kay Sung, C.J.C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765, 1997.

[Vap99a] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[Vap99b] V.N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.

[VBT14] Vikramkumar, Vijaykumar B, and Trilochan. Bayes and naive bayes classifier, 2014.

[WS06] Hao Wu and Jonathan L. Shapiro. Does overfitting affect performance in estimation of distribution algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 433–434, New York, NY, USA, 2006. Association for Computing Machinery.

[XJH+18] Y. Xiang, Q. Jiang, J. He, X. Jin, L. Wu, and S. Yao. The advance of support tensor machine. In *2018 IEEE 16th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 121–128, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

[Yin19] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, 02 2019.

[ZMH+94] Andreas Zell, Niels Mache, Ralf Hübner, Günter Mamier, Michael Vogt, Michael Schmalzl, and Kai-Uwe Herrmann. *SNNS (Stuttgart Neural Network Simulator)*, pages 165–186. Springer US, Boston, MA, 1994.

[ZSG+22] Mohd Zaki, Siddhant Sharma, Sunil Kumar Gurjar, Raju Goyal, Jayadeva, and N. M. Anoop Krishnan. Cementron: Machine learning the constituent phases in cement clinker from optical images, 2022.