# Explainability in Deep Learning Vision Models

DISSERTATION

Submitted in partial fulfilment of the requirements of the
MTech Data Science and Engineering Degree programme

*By*
Thikkireddi Sri Siva Narasimha Sainadh
BITS ID No. 2020sc04389

*Under the supervision of*
Geeta Phatak
Data Scientist
Marlabs Innovations Private Limited, Bengaluru


BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA

*Mar 8, 2023*

# BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

## CERTIFICATE

This is to certify that the Dissertation entitled **Explainability in Deep Learning Vision Models** and submitted by Mr./Ms. **Thikkireddi Sri Siva Narasimha Sainadh** ID No: **2020sc04389** in partial fulfilment of the requirements of **DSECLZG628T** Dissertation, embodies the work done by him/her under my supervision.

Signature of the Supervisor

Place: Bengaluru

Date: 12/03/2023

Contact: 9922433503

Geeta Phatak,

Data Scientist

Marlabs Innovations

# Acknowledgments

I would like to start by thanking Geeta Phatak, my M.Tech supervisor, for her support and guidance especially during the dissertation time. I've received not only theoretical help, but also was able to complete this significant project thanks to her helping me manage my work/Academic project better.

I would also like to give a huge thanks to Akash Thakur, my previous mentor, and Abishek U, my colleague, and friend. They were of huge help and support during my academic journey with BITS.

I would like to thank my Manager Venkatkrishnan B, for his support & guidance throughout my dissertation journey right from topic selection.

Finally, I would like to thank my Family, Friends, and colleagues at Marlabs for their support and understanding of my academic journey with BITS.

*"By far, the greatest danger of Artificial Intelligence is that people conclude too early that they understand it."*

*—Eliezer Yudkowsky*

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**
**FIRST SEMESTER 2022-23**

DSECLZG628T **DISSERTATION**

| | | |
|---|---|---|
| Dissertation Title | : | Explainability in Deep Learning Vision Models |
| Name of Supervisor | : | Geeta Phatak |
| Name of Student | : | Thikkireddi Sri Siva Narasimha Sainadh |
| ID No. of Student | : | 2020sc04389 |

# Abstract

Explainability in Deep models has been a black box for the most part since the dawn of implementation of first trainable perceptron. Being able to interpret a model increases trust and reliability on model output in real time. This becomes all the more important in scenarios involving critical and sensitive data use cases like healthcare. And critical decision-making in domains like law and banks (credit lending) etc. If a model is predicting if a skin condition is potential skin cancer, here small error in judgment could lead to a wrong diagnosis and could be fatal for the patient.

If we can understand the model, we can identify and diagnose if there is any bias present in the model. What is causing the model to mis-classify a data instance and what the model is looking at (vision data) while making that particular decision. If we have this understanding and control of the model, it can be tuned accordingly. If a vision model has been trained on the health imagery data from American population, it might not perform well for data from Asian people due to the difference in skin color/patterns or a variation/strain of the disease that is local to that population and unknown to the model. How do we debug what the model is doing wrong and right in this situation? If we know where the model is lacking, we can overcome its shortcoming in most scenarios by collecting more training data, adding data augmentations, tuning the model architecture, etc. So, Model Explainability becomes important while debugging a model and trusting the model to perform in real-world critical decisions. This can be implemented as an extension of the core CNN model to understand the model.

**Key Words: Image Analysis, CNN, Gradcam, Ingegrated Gradients, Grad Shap, Grad io, feedback, Docker, Healthcare.**

# Table of Contents

# Abbreviations

| | |
|---|---|
| DNN | Deep Neural Network |
| DL | Deep Learning |
| GD | Gradient Descent |
| BP | Back Propagation |
| CNN | Convolutional Neural Nets |
| EHR | Electronic Health Records |
| OCR | Optical Character Recognition |
| MRI | Magnetic Resonance Imaging |
| GAP | Global Average Pooling |
| LR | Learning Rate |
| API | Application Programming Interface |
| GPU | Graphics Processing Unit |

Other data specific terms:

- Actinic keratoses and intraepithelial carcinoma / Bowen's disease (`akiec`),
- basal cell carcinoma (`bcc`),
- benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, `bkl`),
- dermatofibroma (`df`),
- melanoma (`mel`),
- melanocytic nevi (`nv`) and
- vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, `vasc`).

# List of Tables

# Table of Figures

# Chapter – I

# Introduction

Neural Networks can have a large number of free parameters, and this gives them the flexibility to fit highly complex data that others typical ML Models are too naive to fit. This model complexity also brings with it the difficulties of training such a complex network and guaranteeing the resultant model generalizes to the new data. Debugging and tuning such a complex model is hard given sheer number of parameters in the model.

Models built for critical health care decisions often don't move to production and the model results are taken with a pinch of salt. The reason is simple – There is no way to trust the model and for a layman, the models innerworkings and reasoning is a complete black box. Even data scientists have hard time understanding what the model is doing right and wrong, this knowledge can only be obtained by dissecting the model inner workings and looking into the outputs of intermediate layers.

This helps understand if the vision model is looking at the right things when making a decision. Model Explainability is ongoing research area and understanding the model intelligence and flaws can be help tune the model better for the use case at hand.

Skin Cancer dataset is selected for experimenting and dissecting cnn models including well know architectures like resnet-18, resnet-34. Model understanding and tuning will help improve the model results and improve confidence in model decision.

The final algorithm will be able to bring model explainability to some extent, by isolating and highlighting the affected area using nothing more than model activations in the last convolution layer.

## 1.1 Deep Learning for Image Classification:

Deep learning is now being incorporated into a variety of use cases involving a variety of input data from structured to text to image/video. And yet Image is the more prominent area where deep learning excels over traditional machine learning thanks to its ability to learn features itself instead of manual extraction and spoon-feeding methods in ML.

## 1.2 Importance of Deep learning in Health Care

The rate at which data is being generated at health cate facilities and institutions is massive. Many tasks in health domain involve reports, prescriptions, bills, patient scan images and all of these can be maintained in digital format. So, a task like OCR to read a medical prescription to a complex ones like doing a preliminary diagnosis based on a MRI scan, identifying type of potential skin cancer based on the subtle patterns on medical imagery can be achieved through deep learning models.

But just building a model, training it on labeled data and hoping it makes the right call-in delicate situations is not the right way of incorporating AI into health care, there should be rational explanation and a potential feedback system from the end user (medical expert) for the model to learn from its flaws and re-train itself on its mistakes.

## 1.3 Model

### 1.3.1 Convolution Neural Nets:

Kernels are the core of CNN's, it can be interpreted as that the features that they extract are what the model sees in that layer.

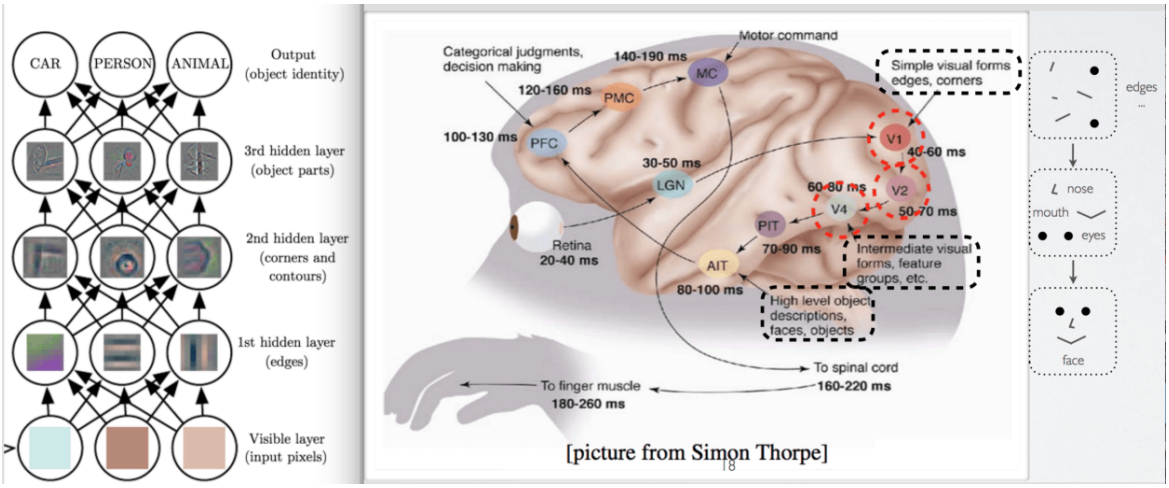



*Figure 1*
*(From Google)*

In other words, to debug and explain a vision model's output we need to understand how the model is processing the knowledge in the image we give as input, for example for a model that does object detection/classification, the model should be able to identify edges and texture of the object before it can classify the object. In a broad sense this is how we can interpret what a vision model is learning by seeing what it's trying to extract in each layer.

For example, take a well-trained classification model taking an image of a dragon as input would understand its input by putting together features it extracted in the initial layers to building more complex feature. (figure 2)

Edges/Gradients (lines on the skin)
↓
Textures/Patterns (large hatch pattern)
↓
Part of Objects (head similar to crocodile)
↓
Objects (it's a dragon!)

During model training, the convolution kernels and the training samples are not connected to each other. The convolution kernels are designed to match the local pattern present in the training images,

which is a rare occurrence. As the iterations progress, the convolution kernels become more effective at matching the local patterns in the training samples.

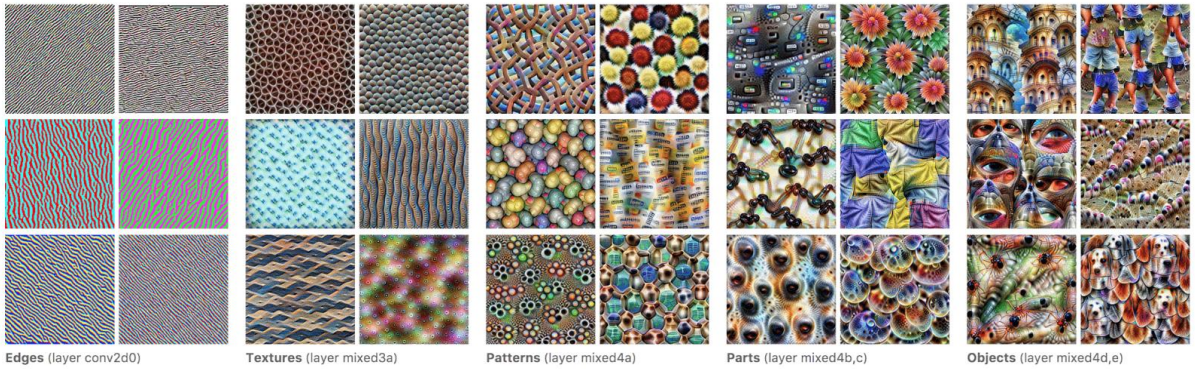



*Figure 2*
*(From Google)*

### 1.3.2 Model Engineering:

Designing Models require discipline, every step taken must have a clear purpose for achieving the final goal. There is often a trade-off while tuning the model for example, adding/removing layers or resizing input data. Various components build up the ideal model for the use case at hand. A few that really helped improve and optimize the model performance and make the model robust are like 1x1 convolution, and GAP (Global Average Pooling Layer) layers.

#### 1.3.2.1 1x1 Convolution:

After the convolution is done, we need to take a call about what to do with the values our kernel is providing us. Should we let all pass, or should we change them? This question of choice (of what to do with output) has kept us guessing. And as humans always feel, deciding what to pick and what to remove must have a complicated solution.

#### 1.3.2.2 Activation Functions:

Activation functions decide what to do with the values our kernel is providing us. Should we let all pass, or should we change them. Now deciding what to pick and what to remove is a crucial decision and is a key component in tuning the model. Thankfully a lot of research has already gone into selecting activation functions depending on the use case.

#### 1.3.2.3 GAP Layer

With scarce resources at hand, the GAP layer was a lifesaver to reduce the computational requirements by completely substituting the end fully connected layers with a GAP layer within deep neural networks, and using the resulting vector as input to the softmax layer for the purpose of classification.

#### 1.3.2.4 LR Scheduler

The learning rate is a crucial hyperparameter for your model which influence how fast the model converges. Every combination of model and data would require a different learning rate, so Instead of running countless experiments in picking out the right one we can start with something that works and let the scheduler take care of the rest instead of a constant learning rate.

### 1.3.3 Data Engineering:

The core thought process behind data engineering for the models is "Garbage in, garbage out", feeding the model good representative data is very crucial. Even the best vision architectures are wasted and will perform badly if the input data is of low quality or unrepresentative of real-world data. For vision models, one can even use domain knowledge to augment data to fill in for unseen variations in train data that would most like likely occur in real data.

#### 1.3.3.1 Image Normalization & Batch Normalization:

In image processing, normalization is a method that changes the range of pixel strength values. The idea is to produce a histogram with equal amounts of pixels at each intensity level.

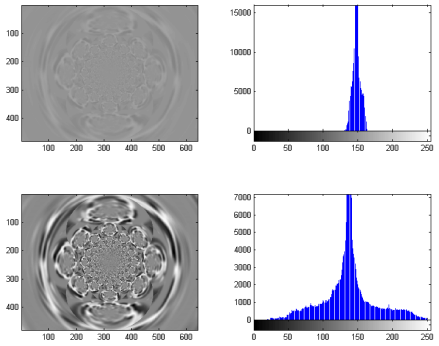


*Figure 3 (Ioffe & Szegedy, 2015)*

In 2015, Batch Normalization was introduced and has since become the established norm for all CNNs and RNNs. If features are found at a similar scale, then weights would be on a similar scale, and then backprop would work better. So, why limit normalization to images only then? BN addresses the Internal Covariate shift problem in the data. For example, imagine what would happen if one channel ranged from -1 to 1 and another between -1000 to 1000. Studies indicate that neural networks with a large number of layers can be trained more quickly and demonstrate improved generalization when the distribution of activations is maintained normalized throughout the BackProp process.

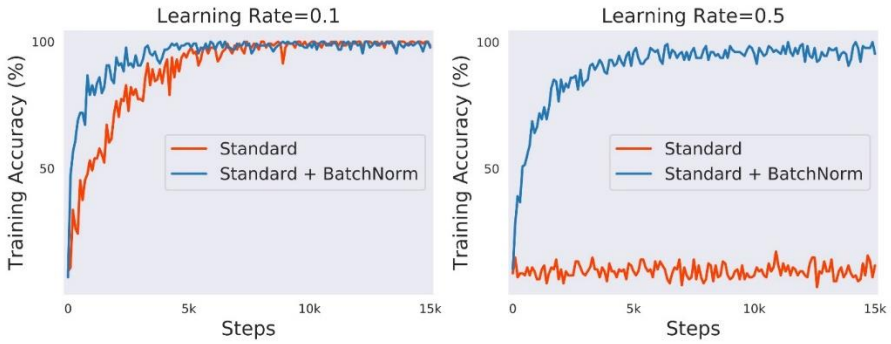



*Figure 4 (Ioffe & Szegedy, 2015)*

#### 1.3.3.2 Data Augmentations:

In general, deep neural networks perform better with larger amounts of data. However, limited data can result in over-fitting of the model, ultimately decreasing its ability to generalize well during validation phase. Model data could be hard to find/gather, Data augmentations help generate 'new' data using the data that we already have by adding, eliminating, and joining various things to existing data. Transformations of images (like rotation, scaling, saturation, and random noise) are a few techniques that help distort the train data to create new nosier data that also helps the model learn more variations and generalize better.

#### 1.3.3.3 Regularization

Deep learning is also not immune to the curse of model overfitting, Regularization strategies used like Early stopping, Dropout, L1, and L2 can be implemented in models to reduce the generalization error. Interestingly Data Augmentation strategies also act as regularizers for the model. Think of it as forcing the model to learn something new by adding random noise and keeping it from memorizing train data.

### 1.3.4 Pre-built Architectures (Res Net):

Complex Images/input data would ideally fit better by Increasing network depth, but training deep networks is a challenging task due to the well-known issue of the vanishing gradient. This arises because the gradient becomes exceptionally small after it is backpropagated to earlier layers, resulting in a repeated multiplication. Consequently, the network's performance tends to saturate and deteriorate swiftly as it becomes increasingly complex.

(He, Deep Residual Learning for Image Recognition., 2016)The authors of the ResNet paper argue, that the network's performance should not be negatively affected by layer stacking. This is because we can use identity mappings, which don't impact the network, to create a new architecture that would perform the same. As a result, the deeper model shouldn't have a higher training error than shallower models.

The core idea of behind ResNet architecture is introducing "identity shortcut connection or Skip connections" that skips layers and adds the output down the road to a different layer. A residual block is displayed in the following fig. ResNet can be considered an ensemble of smaller networks.

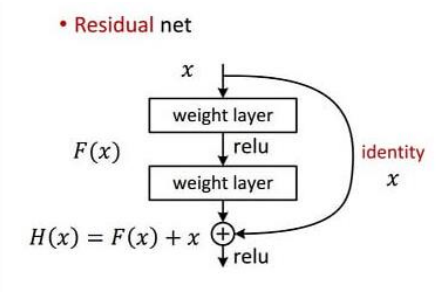



*Figure 5 (He, 2016)*

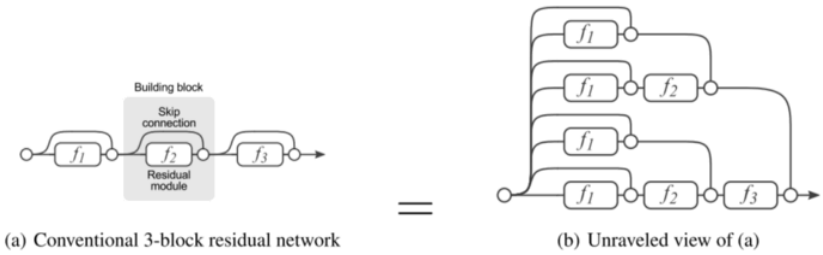



*Figure 6 (He, 2016)*

Resnet 34 will be one of the pre-built architectures that will be utilized to build the solution, the training of parameters will happen from scratch.

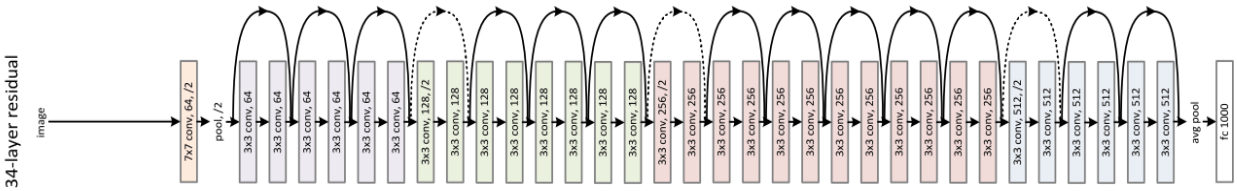



*Figure 7 (He, 2016)*

## 1.4 Explainability in Vision Models:

The core idea behind this project is model explainability. Understanding what the model is looking at while making a particular prediction is crucial for model maintenance, tuning and re-training. In vision models, this can be achieved either by gradients or by understanding feature importance with change in output.

### 1.4.1 GRAD-CAM

The model explainablity technique, Gradient-weighted Class Activation Mapping (gradcam) utilizes the gradients of a particular class of interest, such as 'dog', that flow into the last convolutional layer and does a weighted average of the activation maps to generate heatmap. This heat map indicates significant areas in the image that aided the model in predicting the target class.

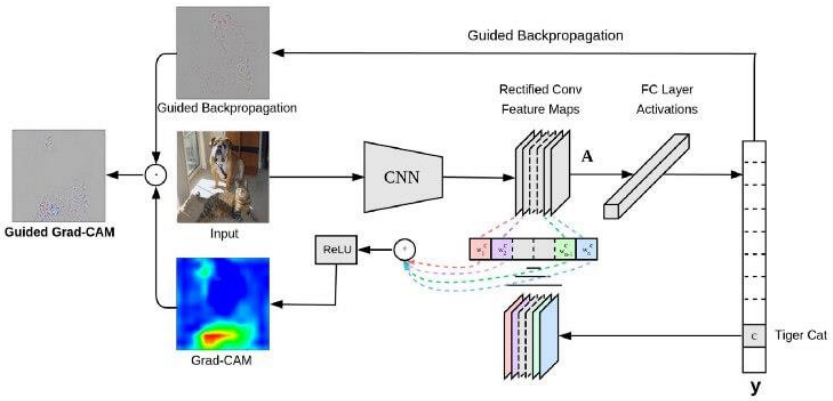



*Figure 8 (R. R. Selvaraju, 2017 )*

A few other approaches were also utilized and will be further touched upon in 4.4 Model Explainability.

## 1.5 Solution Design and Plan

In this solution I plan to build a model that is not only robust and lightweight but also explainable and locally interpretable through Local interpretation: i.e. understanding how the model makes decisions for a single instance

Implement and showcase Explainability in deep models through techniques from active research areas like grad cam. Use explainability to improve on the first established model. The goal is to build a model that is not only robust and lightweight but also explainable and locally interpretable.

### 1.5.1 Business process flow:

1. Research
2. Data Collection
3. Experimentation
4. Implementation
5. Simple UI for Demos (Streamlit or Grad io)
6. Dockerization
7. Amazon ECS deployment

### 1.5.2 High-level Solution Architecture

If the final solution is to be divided into layers from the inside out, I would consist of the following layers.

Deep Model - Model Layer
Stramlit/ grad IO - UI Layer
Docker - Docker Layer
AWS ECS - Deployment layer

The following figure (10) describes the vision and flow for the final solution.

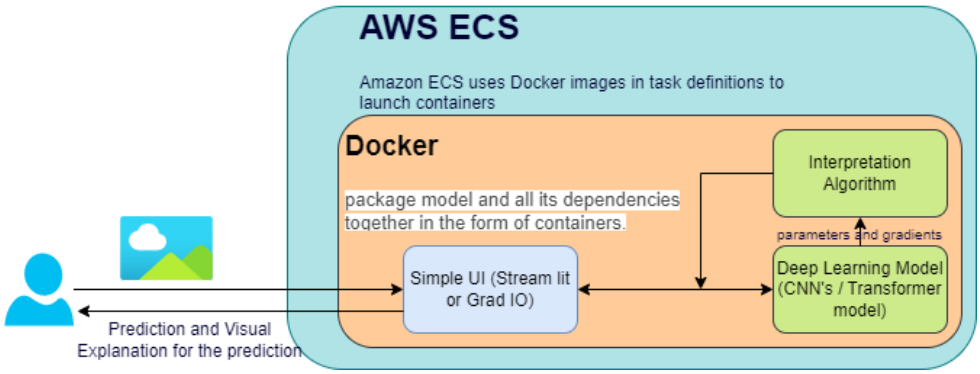

AWS ECS
Amazon ECS uses Docker images in task definitions to launch containers
Docker
package model and all its dependencies together in the form of containers.
Interpretation Algorithm
parameters and gradients
Simple UI (Stream lit or Grad IO)
Deep Learning Model (CNN's / Transformer model)
Prediction and Visual Explanation for the prediction

*Figure 9*

# Chapter 2

# Literature Survey

The health care domain has a huge potential to incorporate deep learning solutions. One such use case is Skin cancer detection. The use of deep learning technology in the healthcare industry is still in its early stages, and most of the projects in health care that are focused on implementing deep learning in important decision-making processes are small-scale pilot projects or research initiatives that have not yet been commercialized. Nonetheless, deep learning is slowly becoming integrated into innovative tools that have significant real-world clinical benefits.

(Sijie Yang1, 2021) The biomedical datasets are complex, disorganized, contain errors, and have a lot of missing information, which makes it difficult to extract valuable insights from them. To analyze these datasets effectively, a suitable methodology is required. Currently, deep learning is a popular artificial intelligence technique in the machine learning community that can be used to process large amounts of biomedical data. Although deep learning is sometimes criticized for its inability to explain its results, it is still widely used in image recognition and other domains where interpretability is not a major issue. The understandability of models is particularly critical when it comes to healthcare. This is because doctors will only have faith in a model that can furnish them with dependable information, which in turn leads to accurate and suitable decisions. Furthermore, an interpretable model can help patients get a more complete comprehension of their situation. The piece additionally highlights the difficulties and issues encountered when utilizing deep learning in computational medicine. As such, it serves as a guide and a means of enhancing the adoption of deep learning in the medical and health domains in the days to come.

(Wu Y, 2022) In this study, The author provides a detailed summary of the latest advancements in deep learning algorithms for identifying and categorizing skin cancer in this research. The study offers a thorough examination of skin cancer classification, concentrating on modern deep-learning techniques. The paper also looks at the use of conventional convolutional neural network (CNN) methodologies for skin cancer classification. Additionally, it explores the challenges involved in classifying skin cancer, such as data imbalance and restriction, cross-domain adaptability, model durability, and model efficiency, as well as relevant deep learning-based solutions.

The most frequently utilized deep learning models for the classification of skin cancer using CNN-based methods are VGGNet, GoogleNet, ResNet, and their variations. However, there are a number of issues that still need to be addressed, including uneven datasets, insufficient labeled data, the ability to generalize across different domains, the presence of noisy data from heterogeneous images and devices, and the development of effective models for complex classification tasks.

To tackle the issues, techniques such as generative adversarial networks, data expansion, creation of fresh loss functions, transfer learning, nonstop learning, adversarial training, light-weight CNN, trimming tactics, knowledge transfer, and transformer are employed. It is anticipated that AI is capable of contributing to a significant shift in the identification of skin cancer and diagnosis in the near future.

(Mohammad Ali Kadampur, 2020) In this paper, a CNN model was proposed for skin cancer classification use cases. The article utilized a Model Driven Architecture called Deep Learning Studio to facilitate Deep Learning. It introduced the capabilities of the DLS tool while showcasing how to develop a Deep Learning Model using it. The paper included the process of preparing data from dermal cell images and how it was incorporated in the DLS model for identifying cancer cells. Remarkably, the DLS models garnered an AUC of 99.77% accuracy in detecting cancer cells from the images.

Below are the experimental results from various models trained and tested on the dataset. They have used a total of 5 models for experimentation purposes including models like Inception and Resnet.

Table 1. Metric values of pre-trained deep learning classifiers.

| Model | (Precision) | (F1-Score) | (ROC AUC) |
|---|---|---|---|
| MODEL2(resnet) | 94.24 | 94.22 | 98.61 |
| MODEL3(squeeznet) | 97.40 | 94.57 | 99.77 |
| MODEL4(densenet) | 97.51 | 96.27 | 99.09 |
| MODEL5(inceptionv3) | 98.19 | 95.74 | 99.23 |

*Figure 10 (Mohammad Ali Kadampur, 2020)*

The article also noted that the programming specialist can obtain the programming code for the model for additional investigation. The article recognized that the option to download the trained model and create enterprise-level applications is a remarkable foundation for future research.

# Chapter 3

# The Data

## 3.1 Data Collection and Understanding

The most prevalent type of cancer in USA is skin cancer. It is frequently categorised as melanoma or nonmelanoma skin cancer (NMSC). Due to a lack of diagnostic criteria and underreporting, it is challenging to pinpoint the exact incidence of skin cancer. But during the past few decades, epidemiologic investigations have revealed rising rates of both NMSC and melanoma.

The regions of the head and neck that are exposed to the sun often have skin malignancies. Caucasians are considerably more often at risk due to the photoprotective properties of epidermal melanin. Experts with 3 years of advanced dermatology training post-medical school frequently carry out the evaluation. The great majority of concerning lesions found during physical examination will be subjected to a skin biopsy in the hospital.

Dermatoscopic data from several sources of typical pigmented skin lesions make up this dataset. This has been acquired through Harvard Dataverse & Kaggle. Histopathology confirms more than 50% of lesions, with follow-up inspection, expert consensus (consensus), or in-vivo confocal microscopy confirmation providing the final word in the remaining cases (confocal). The lesion id-column in the HAM10000 metadata file serves as a unique identification for the data entries for lesions that have numerous pictures.. (Gruber & Zito., 2022)

## 3.2. Data Samples

There are 10015 dermatoscopic images in total in the image dataset. The cases comprise of good representative collection of all significant pigmented lesion diagnostic subtypes.

- Actinic keratoses and intraepithelial carcinoma / Bowen's disease (AKIEC)

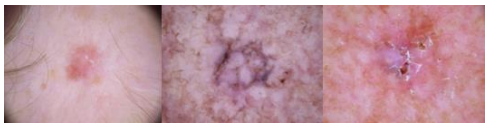


*Figure 11*

- basal cell carcinoma (BCC)

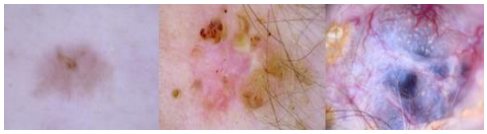


*Figure 12*

- benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus-like keratoses, BKL)

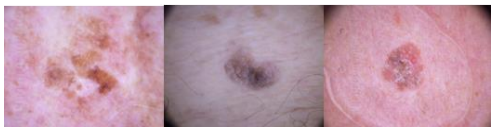


*Figure 13*

- dermatofibroma (DF)

*Figure 14*

- melanoma (MEL)

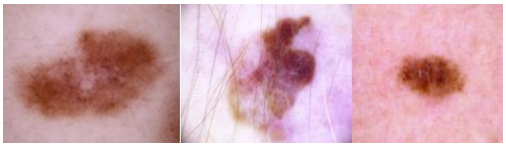


*Figure 15*

- melanocytic nevi (NV)

*Figure 16*

- vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, VASC)

*Figure 17*

## 3.3 Data Augmentations

Data augmentation (DA) is an effective alternative to obtaining valid data and generating new labeled data based on existing data. Designing appropriate DA policies requires can improve the model learning by a good margin. Deep neural networks typically perform better with more data.

### 3.3.1 Rotate, Gaussian noise, and Random Crop:

These are the basic image transformations used on image data before pumping the data into the model. These transformations add some amount of variation and noise to the data. This in turn helps the model generalize better on new data. Below is a sample of the actual image from the training dataset and the augmented data next to it.

*Figure 18*

### 3.3.2 Cut-out

Cutout is a straightforward yet very effective data augmentation technique. It removes a small portion of the image and forces the model to learn the features from what is left of the image. This can be thought of as a type of regularization as we are forcing the model to learn from a different part of the image every time, we push it through the model. This helps the model understand the patterns in the images better instead of memorizing one bit/pattern of the input image. Below is a sample of the actual image from the training dataset and the augmented data next to it.

*Figure 19*

The reason why cut-out a is a very good data augmentation technique: (Terrance DeVries, 2017 )

| Method | C10 | C10+ | C100 | C100+ | SVHN |
|---|---|---|---|---|---|
| ResNet18 [5] | $10.63 \pm 0.26$ | $4.72 \pm 0.21$ | $36.68 \pm 0.57$ | $22.46 \pm 0.31$ | - |
| ResNet18 + cutout | $9.31 \pm 0.18$ | $3.99 \pm 0.13$ | $34.98 \pm 0.29$ | $21.96 \pm 0.24$ | - |
| WideResNet [22] | $6.97 \pm 0.22$ | $3.87 \pm 0.08$ | $26.06 \pm 0.22$ | $18.8 \pm 0.08$ | $1.60 \pm 0.05$ |
| WideResNet + cutout | $\mathbf{5.54 \pm 0.08}$ | $3.08 \pm 0.16$ | $\mathbf{23.94 \pm 0.15}$ | $18.41 \pm 0.27$ | $\mathbf{1.30 \pm 0.03}$ |
| Shake-shake regularization [4] | - | 2.86 | - | 15.85 | - |
| Shake-shake regularization + cutout | - | $\mathbf{2.56 \pm 0.07}$ | - | $\mathbf{15.20 \pm 0.21}$ | - |

Table 1: Test error rates (%) on CIFAR (C10, C100) and SVHN datasets. "+" indicates standard data augmentation (mirror + crop). Results averaged over five runs, with the exception of shake-shake regularization which only had three runs each. Baseline shake-shake regularization results taken from [4].

*Table 1 (Terrance DeVries, 2017 )*

### 3.3.3 Hue & Colour jitter:

The samples are collected from people with different skin tones, but still, the model might be missing out few combinations of skin conditions and skin tones. To mitigate this issue a random hue and saturation are introduced into trained data. This generates and augments various skin tones and lighting conditions in the dataset.

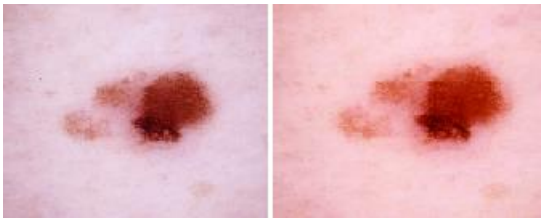


*Figure 20*

# Chapter 4

## The Solution & Experiments

The goal is to build a robust vision model capable of accurate and reliable classification and generalizing well for unseen data. A sensitive use case like Skin cancer classification would benefit from the model providing an explainable output as an extension to the model output, this would help both model diagnosis and improve model reliability. The model should have good specificity, recall and f1 scores compared to the start variant of the model. As this is a medical diagnosis model, false negatives in model classification are not acceptable and the model will be tuned in a way to minimize the same.

### 4.1 Model

I have experimented with multiple models, from building a custom model from scratch with just 51,221 parameters to utilizing pre-built architectures like resnet-18 and restnet-34 which have millions of parameters. I have also experimented with multiple Data Augmentation techniques, optimizers, Learning rate schedulers, and other regularizing techniques. I'll showcase and explain my findings in the upcoming sections.

The Model building started with a base CNN model built from scratch using the idea of a receptive field.

CNN Model: (base)

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 8, 224, 224] | 224 |
| ReLU-2 | [-1, 8, 224, 224] | 0 |
| BatchNorm2d-3 | [-1, 8, 224, 224] | 16 |
| Dropout-4 | [-1, 8, 224, 224] | 0 |
| Conv2d-5 | [-1, 16, 224, 224] | 1,168 |
| ReLU-6 | [-1, 16, 224, 224] | 0 |
| BatchNorm2d-7 | [-1, 16, 224, 224] | 32 |
| Dropout-8 | [-1, 16, 224, 224] | 0 |
| Conv2d-9 | [-1, 8, 224, 224] | 136 |
| ReLU-10 | [-1, 8, 224, 224] | 0 |
| BatchNorm2d-11 | [-1, 8, 224, 224] | 16 |
| Dropout-12 | [-1, 8, 224, 224] | 0 |
| MaxPool2d-13 | [-1, 8, 112, 112] | 0 |
| Conv2d-14 | [-1, 16, 112, 112] | 1,168 |
| ReLU-15 | [-1, 16, 112, 112] | 0 |
| BatchNorm2d-16 | [-1, 16, 112, 112] | 32 |
| Dropout-17 | [-1, 16, 112, 112] | 0 |
| Conv2d-18 | [-1, 16, 112, 112] | 2,320 |
| ReLU-19 | [-1, 16, 112, 112] | 0 |
| BatchNorm2d-20 | [-1, 16, 112, 112] | 32 |
| Dropout-21 | [-1, 16, 112, 112] | 0 |
| Conv2d-22 | [-1, 32, 112, 112] | 4,640 |
| ReLU-23 | [-1, 32, 112, 112] | 0 |
| BatchNorm2d-24 | [-1, 32, 112, 112] | 64 |

```
           Dropout-25         [-1, 32, 112, 112]               0
            Conv2d-26          [-1, 8, 112, 112]             264
              ReLU-27          [-1, 8, 112, 112]               0
       BatchNorm2d-28          [-1, 8, 112, 112]              16
           Dropout-29          [-1, 8, 112, 112]               0
         MaxPool2d-30            [-1, 8, 56, 56]               0
            Conv2d-31            [-1, 8, 56, 56]             584
              ReLU-32            [-1, 8, 56, 56]               0
       BatchNorm2d-33            [-1, 8, 56, 56]              16
           Dropout-34            [-1, 8, 56, 56]               0
            Conv2d-35           [-1, 16, 56, 56]           1,168
              ReLU-36           [-1, 16, 56, 56]               0
       BatchNorm2d-37           [-1, 16, 56, 56]              32
           Dropout-38           [-1, 16, 56, 56]               0
            Conv2d-39           [-1, 32, 56, 56]           4,640
              ReLU-40           [-1, 32, 56, 56]               0
       BatchNorm2d-41           [-1, 32, 56, 56]              64
           Dropout-42           [-1, 32, 56, 56]               0
            Conv2d-43            [-1, 8, 56, 56]             264
              ReLU-44            [-1, 8, 56, 56]               0
       BatchNorm2d-45            [-1, 8, 56, 56]              16
           Dropout-46            [-1, 8, 56, 56]               0
         MaxPool2d-47            [-1, 8, 28, 28]               0
            Conv2d-48           [-1, 16, 28, 28]           1,168
              ReLU-49           [-1, 16, 28, 28]               0
       BatchNorm2d-50           [-1, 16, 28, 28]              32
           Dropout-51           [-1, 16, 28, 28]               0
            Conv2d-52           [-1, 32, 28, 28]           4,640
              ReLU-53           [-1, 32, 28, 28]               0
       BatchNorm2d-54           [-1, 32, 28, 28]              64
           Dropout-55           [-1, 32, 28, 28]               0
         MaxPool2d-56           [-1, 32, 14, 14]               0
            Conv2d-57           [-1, 32, 14, 14]           9,248
              ReLU-58           [-1, 32, 14, 14]               0
       BatchNorm2d-59           [-1, 32, 14, 14]              64
           Dropout-60           [-1, 32, 14, 14]               0
            Conv2d-61           [-1, 64, 14, 14]          18,496
              ReLU-62           [-1, 64, 14, 14]               0
       BatchNorm2d-63           [-1, 64, 14, 14]             128
           Dropout-64           [-1, 64, 14, 14]               0
         MaxPool2d-65             [-1, 64, 7, 7]               0
            Conv2d-66              [-1, 7, 7, 7]             455
              ReLU-67              [-1, 7, 7, 7]               0
       BatchNorm2d-68              [-1, 7, 7, 7]              14
           Dropout-69              [-1, 7, 7, 7]               0
AdaptiveAvgPool2d-70              [-1, 7, 1, 1]               0
================================================================
Total params: 51,221
Trainable params: 51,221
Non-trainable params: 0
```

*Table 2*

I also experimented with model arcs like Resnet -18 and Resnet -34 by reducing the input size significantly to (64,64) while experimenting with the dataset to pick the best architecture to build the solution.

Resnet -18 Arc:
This architecture was made up of 11M parameters and is exponentially huge compared to the previo us model.

```
==========================================================================
Layer (type:depth-idx)                                  Param #
==========================================================================
├─Conv2d: 1-1                                          1,728
├─BatchNorm2d: 1-2                                     128
├─Sequential: 1-3                                      --
|    └─BasicBlock: 2-1                                 --
|    |    └─Conv2d: 3-1                                36,864
|    |    └─BatchNorm2d: 3-2                           128
|    |    └─Conv2d: 3-3                                36,864
|    |    └─BatchNorm2d: 3-4                           128
|    |    └─Sequential: 3-5                            --
|    └─BasicBlock: 2-2                                 --
|    |    └─Conv2d: 3-6                                36,864
|    |    └─BatchNorm2d: 3-7                           128
|    |    └─Conv2d: 3-8                                36,864
|    |    └─BatchNorm2d: 3-9                           128
|    |    └─Sequential: 3-10                           --
├─Sequential: 1-4                                      --
|    └─BasicBlock: 2-3                                 --
|    |    └─Conv2d: 3-11                               73,728
|    |    └─BatchNorm2d: 3-12                          256
|    |    └─Conv2d: 3-13                               147,456
|    |    └─BatchNorm2d: 3-14                          256
|    |    └─Sequential: 3-15                           8,448
|    └─BasicBlock: 2-4                                 --
|    |    └─Conv2d: 3-16                               147,456
|    |    └─BatchNorm2d: 3-17                          256
|    |    └─Conv2d: 3-18                               147,456
|    |    └─BatchNorm2d: 3-19                          256
|    |    └─Sequential: 3-20                           --
├─Sequential: 1-5                                      --
|    └─BasicBlock: 2-5                                 --
|    |    └─Conv2d: 3-21                               294,912
|    |    └─BatchNorm2d: 3-22                          512
|    |    └─Conv2d: 3-23                               589,824
|    |    └─BatchNorm2d: 3-24                          512
|    |    └─Sequential: 3-25                           33,280
|    └─BasicBlock: 2-6                                 --
|    |    └─Conv2d: 3-26                               589,824
|    |    └─BatchNorm2d: 3-27                          512
|    |    └─Conv2d: 3-28                               589,824
|    |    └─BatchNorm2d: 3-29                          512
|    |    └─Sequential: 3-30                           --
├─Sequential: 1-6                                      --
|    └─BasicBlock: 2-7                                 --
|    |    └─Conv2d: 3-31                               1,179,648
|    |    └─BatchNorm2d: 3-32                          1,024
|    |    └─Conv2d: 3-33                               2,359,296
|    |    └─BatchNorm2d: 3-34                          1,024
|    |    └─Sequential: 3-35                           132,096
|    └─BasicBlock: 2-8                                 --
|    |    └─Conv2d: 3-36                               2,359,296
|    |    └─BatchNorm2d: 3-37                          1,024
```

```
|    |    └─Conv2d: 3-38                      2,359,296
|    |    └─BatchNorm2d: 3-39                 1,024
|    |    └─Sequential: 3-40                  --
├─Linear: 1-7                                3,591
==========================================================================
Total params: 11,172,423
Trainable params: 11,172,423
Non-trainable params: 0
==========================================================================
```

*Table 3*

## 4.2 Model Tuning

I started with a base model that I had built using my knowledge of CNNs and receptive fields and started tuning and adding more layers and complexities like 1x1 convolution to reduce the number of channels, Batch Normalization to deal with exploding & vanishing gradients, Data Augmentations and Regularization to keep the model generalized, Global Average Pooling instead of huge linear layers to reduce the computations and parameters and even yield better accuracy. More details are in Appendix A -Experimental results.

### 4.2.1 Receptive Field & Batch Normalization

The receptive Field can be thought of as the size of the input region that generates the feature. It measures how closely an output feature (of any layer) is related to the input region. (Adaloglou, n.d.).

To build the base model that would look at the entire image before making the classification, the idea of receptive fields was crucial. This concept helped me add enough layers to reach the final receptive field equal to the size of the image. And to counteract the problems like exploding gradient with deeper networks I needed batch normalization.

Each layer's output is explicitly scaled by BatchNorm. BN process normalises the activations of each input variable per mini-batch. The term "normalisation" refers to rescaling data so that its mean and standard deviation are both equal to one. It would be a step towards achieving the stable input distributions that would eliminate the negative effects of the internal covariate shift if the inputs to each layer were made brighter**.** (Singla, n.d.). More details are in Appendix A -Experimental results.

### 4.2.2 Data Augmentation & Regularization

Select data augmentation techniques played a huge role in improving the overall accuracy of the model by generating more varied train data for the model to learn. Using Data Augmentation techniques, I could achieve a better accuracy/ generalized model with a smaller initial training set.

Regularization methods like dropout with a probability of 0.01, l1& l2 regularizations have been used to experiment and tune the model to generalize better on unseen data. More details in Appendix A -Experimental results

### 4.2.3 Optimizers and LR Schedulers

Different optimizers have been tested for the model including SGD with momentum, Adam, and RMSprop. SGD with Momentum yielded the best results in minimizing the loss and finding/reaching global minima loss for a  set no of epochs. More details are in Appendix A -Experimental results.

## 4.3 Model Output

I have experimented with multiple combinations of data pre-processing techniques and model configurations to create a robust solution with good performance using only the limited resources that I have. These models can be pushed even further in terms of performance and reliability, more on this in Directions for future work.

The input images were of dimensions (600,450,3) and are resized to (224,224,3) & (64,64,3) and fed to models, for most of the model configurations the train data were sampled equally across all the 7 classes in data. Below are the model results of the final CNN model.

| Test Accuracies | | | |
|---|---|---|---|
| Model | Base Model | With Batch Normalization | BN + With Select Data Augmentations + Tuning |
| Custom CNN | 0.678 | 0.723 | 0.739 |
| Resnet-18 | 0.82 | NA | 0.889 |
| Resnet-34 | 0.8321 | NA | 0.8742 |

*Table 4*

| Optimizers | SGD with Momentum | RMSprop | Adam |
|---|---|---|---|
| Resnet-18 Base (20 Epoch) | 0.82 | 0.734 | 0.76 |

*Table 5*

## 4.4 Model Explainability

Explainability of the vision model is the core goal in implementing this solution, I have explored multiple model explainability methods that are still in the nascent phase of implementation across the AI community. A few of them that I experimented with and implemented in this solution are Grad cam, Integrated Gradients, and Gradient Shap. I've integrated grad cam into the final UI as it has the most understandable way of outputting model understanding through a heatmap.

### 4.4.1 Grad Cam

Gradient-weighted Class Activation Mapping (GradCAM) creates a coarse localization map that emphasises the key areas in the image that were utilized by model for the class prediction, by using the gradients w.r.t target class flowing into the final convolutional layer. (R. R. Selvaraju, 2017 )

To visualize this, I took the last convolutional feature map and weighed each channel with the gradient of the corresponding classified class. By measuring how important each channel is in relation to the class, we can determine how strongly the input image activates various channels.

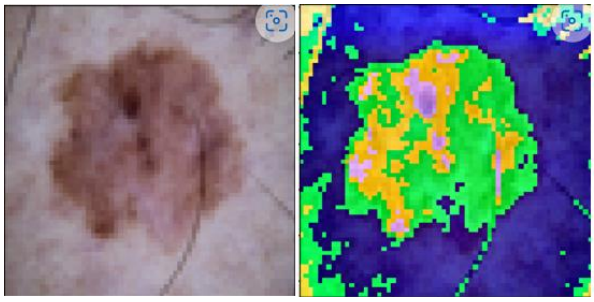


*Figure 21 - 4.4.1*

### 4.4.2 Integrated Gradients

Visualizes how important input features are to the model's prediction, Integrated Gradient computes the gradient of the model's prediction output to its input features. To quantify the correlation between changes to a feature and changes in the model's predictions, gradients are calculated in this experiment. Which pixel has the greatest impact on the class probabilities predicted by the model is revealed by the gradient. (Khandelwal, n.d.)

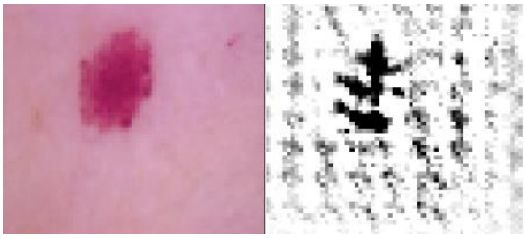


*Figure 22*

### 4.4.3 Gradient Shap

Each feature's SHAP (SHapley Additive exPlanation) value indicates how the predicted model prediction changed when that feature was taken into consideration.. They explain how to get from the base value E[f(z)] that would be predicted if we did not know any features of the current output f(x). It is the mean of the marginal contributions for all combinations. (Scott M. Lundberg)

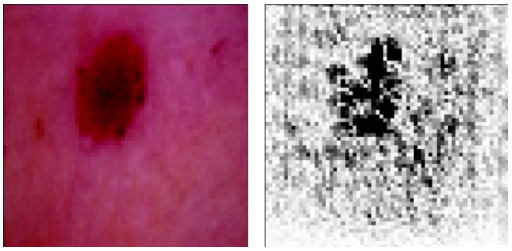


*Figure 23*

## 4.5 Conclusion and Final Results

Resnet-18 was the model that was performing the best with good specificity, recall, and f1 scores for all the critical classes. This model yielded the best results in predicting the skin cancer type in unseen data with about 88.9% test/validation accuracy. The model took smaller resized input of (64,64) yet was able to exceed the performance of other models in just about all the aspects.

| | class | sensitivity | specificity |
|---|---|---|---|
| **0** | akiec | 0.999171 | 0.994845 |
| **1** | bcc | 0.996647 | 0.990338 |
| **2** | bkl | 0.988468 | 0.924731 |
| **3** | df | 0.999155 | 1.000000 |
| **4** | mel | 0.990909 | 0.957895 |
| **5** | nv | 0.992487 | 0.920792 |
| **6** | vasc | 0.999163 | 1.000000 |

*Table 6*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **akiec** | 0.990099 | 0.980392 | 0.985222 | 204.000000 |
| **bcc** | 0.995327 | 0.995327 | 0.995327 | 214.000000 |
| **bkl** | 0.944724 | 0.935323 | 0.940000 | 201.000000 |
| **df** | 0.989418 | 1.000000 | 0.994681 | 187.000000 |
| **mel** | 0.934343 | 0.953608 | 0.943878 | 194.000000 |
| **nv** | 0.957895 | 0.947917 | 0.952880 | 192.000000 |
| **vasc** | 1.000000 | 1.000000 | 1.000000 | 208.000000 |
| **accuracy** | 0.973571 | 0.973571 | 0.973571 | 0.973571 |
| **macro avg** | 0.973115 | 0.973224 | 0.973141 | 1400.000000 |
| **weighted avg** | 0.973621 | 0.973571 | 0.973568 | 1400.000000 |

*Table 7*

Below are graphs depicting the accuracy and loss patterns during the training phase on the train and test test/validation data of the final model.

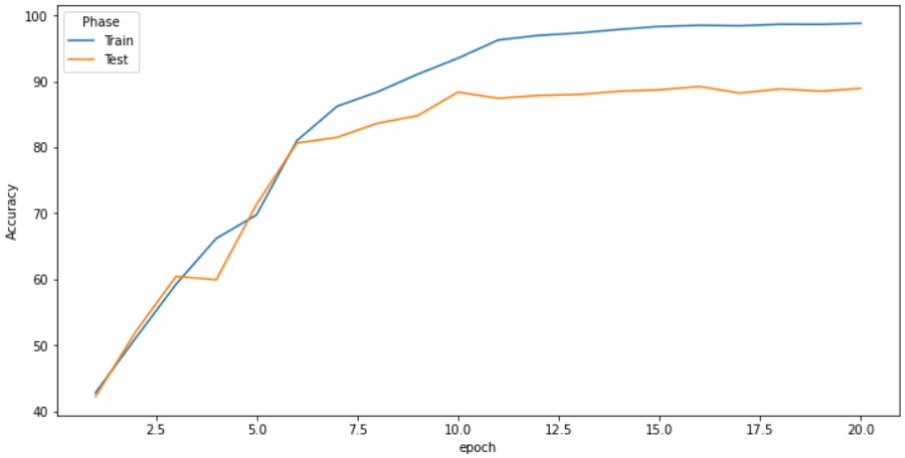



*Figure 24*

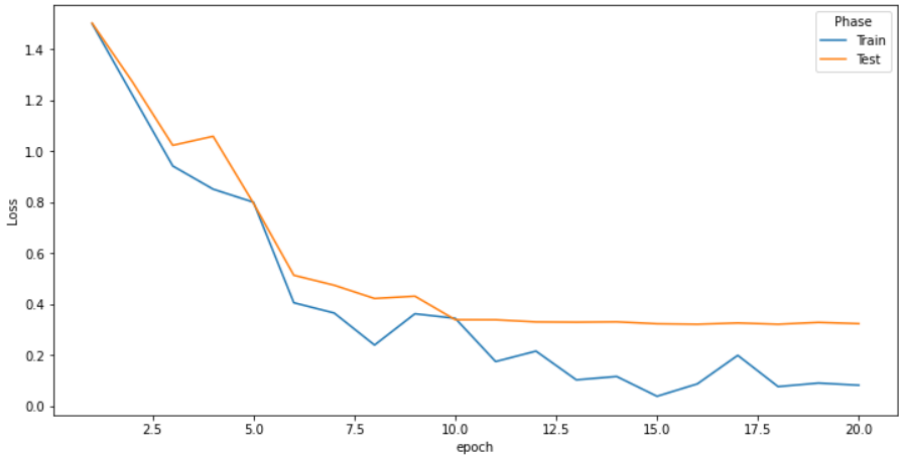



*Figure 25*

Machine learning systems need a lot of iteration loops and ongoing development, therefore the final solution would feature a feedback loop rather than the ship-and-forget pattern found in conventional software. The end user can flag the output if he sees that the prediction is incorrect, this can be utilized as feedback to the model on new data. Every 'x' day's this feedback labels can be clubbed with train data and can be used to re-train the model to keep the model learning from its mistakes and generalize better on future outputs.

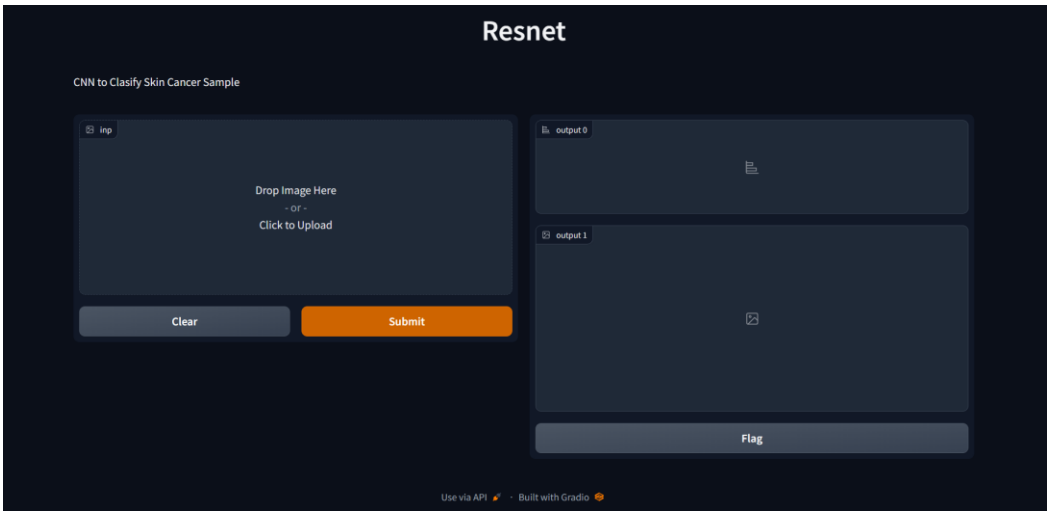



*Figure 26 (UI -Gradio)*

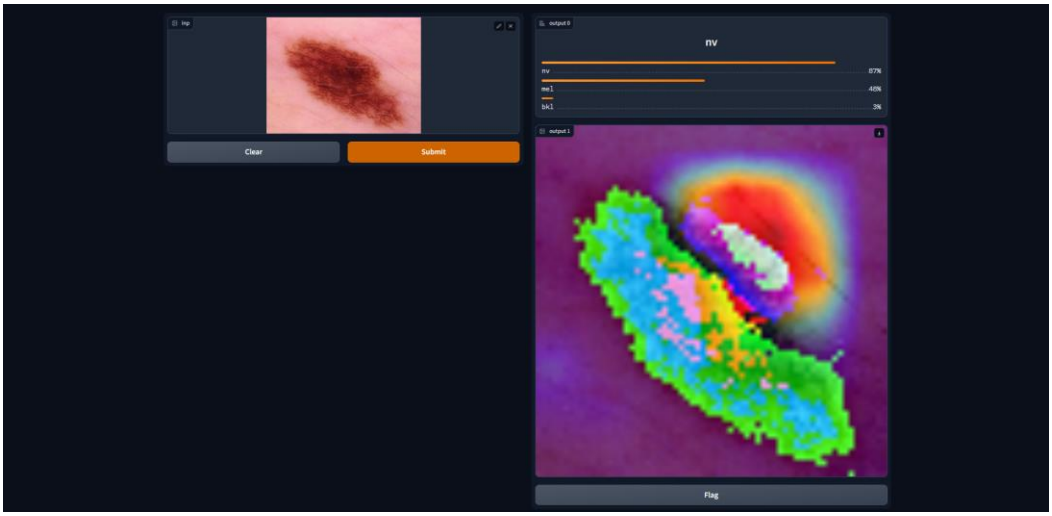



*Figure 27 (Model Output Example 1-Gradio)*

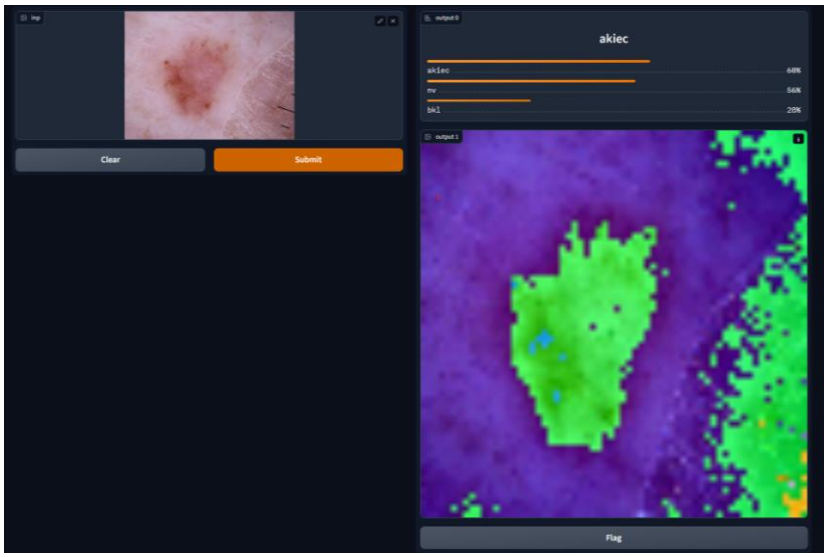



*Figure 28 (Model Output Example 2)*

This final solution in all honesty is not supposed to replace the medical experts of the particular field, but to assist them and speed up the process of preliminary diagnosis. As of date, there are ways to fool DL model like FGSM, so to trust it blindly in critical scenarios such as medical diagnosis is not recommended. But I'm sure the solution would be of great help to any medical expert in sorting and classifying huge amounts of data which would otherwise be very time-consuming and in final decision-making to a good extent.

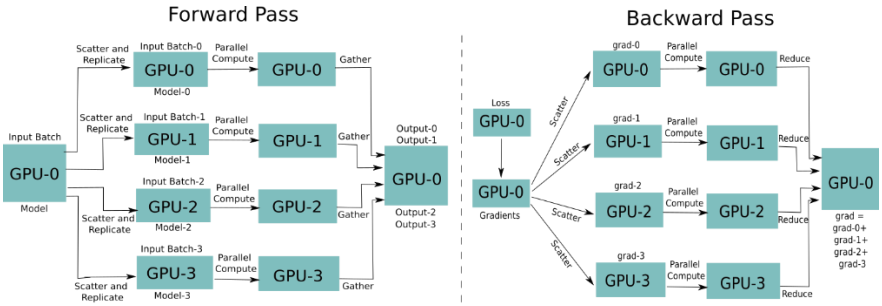
# Chapter 5

# Directions for future work

The models were built and trained in my local system with a 1660ti Nvidia card, this was very limiting in terms of raw power to train larger and more complex models. I hope I can extend this work into building a better solution by moving the training part to the cloud, using single GPU sources like google colab pro, or doing a distributed training on the cluster using DistributedDataParallel, PyTorch Lightning.

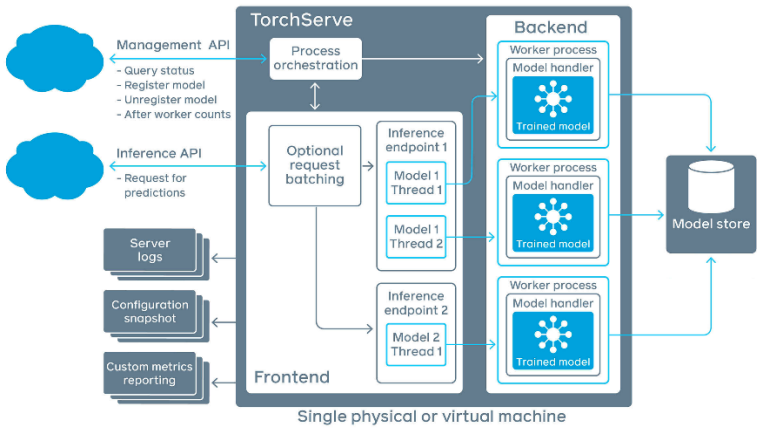
```
from torch.nn.parallel import DistributedDataParallel as DDP
```

Under the hood, Sharded Training utilises Data Parallel Training, but optimizer states and gradients are still distributed across GPUs for the model to train properly.




Decentralized Training over the Internet using Hivemind (Research, n.d.)

The next stage would be model serving aka model as a service, which would make this model available through API and allow it to be used for inference on the go by live applications . Furthermore, the architecture would be de-coupled i.e., the application and model need not be linked and model can be accessed directly through an API, streamlining organisational processes, enabling seamless updates with phased deployments, and enabling the application and model servers to have different hardware (CPU, GPU, and FPGA) and independently scale based on the requests at any given point of time.



# Appendix A

# Experimental results

## A.1 Resnet 18 vs Resnet 34

Resnet 18 despite being a smaller model in a number of parameters, did well when tuned compared to its heavier counterpart renet34. Below are the results that showcase the confusion matrix of each model.

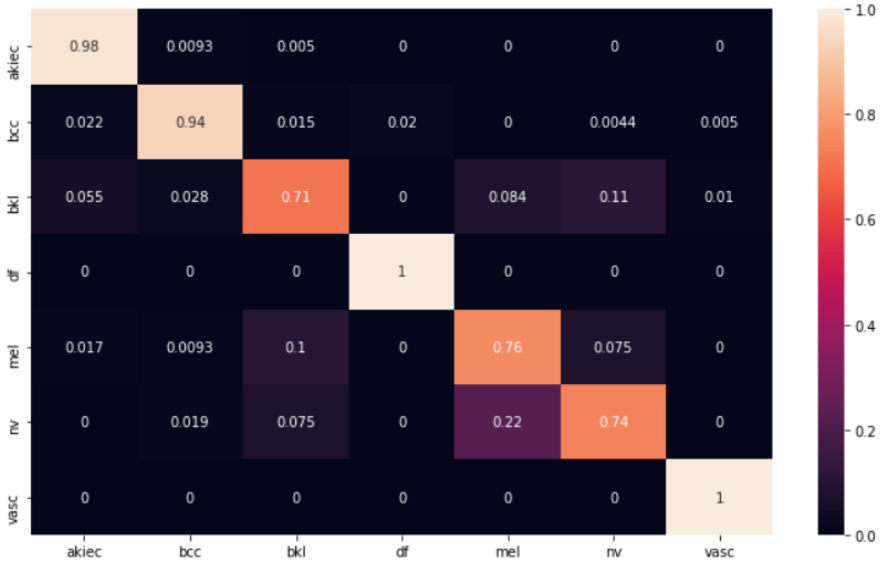



*Figure 29 Resnet 34 confusion Matrix*

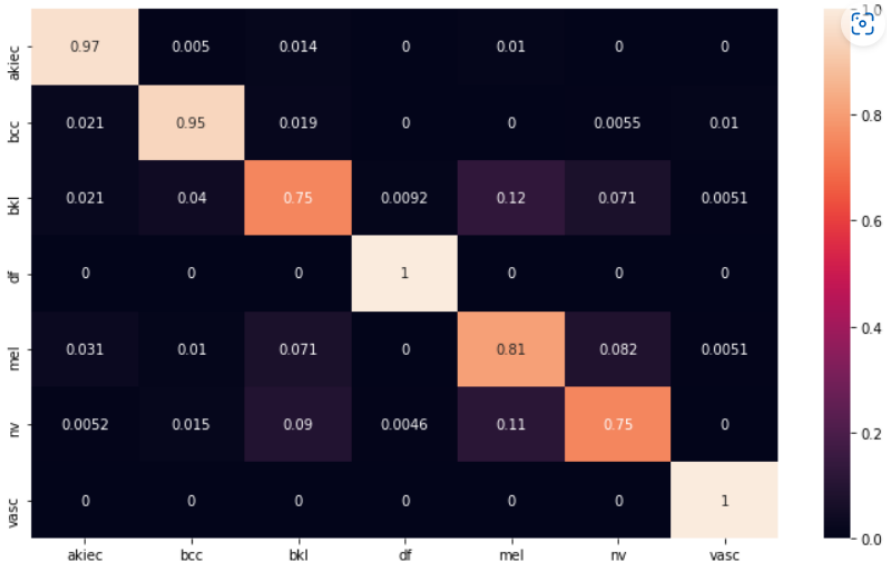



*Figure 30 Resnet 18 confusion Matrix*

## A.2 Impact of Batch Normalization

To normalize an image, we simply choose the pixel with the highest intensity and one with the lowest intensity, and then we calculate a factor that scales the highest intensity to white and the lowest intensity to black. This produces the normalised image by applying it to each and every pixel in the image.

Batch normalization undoubtedly positively impacted model learning, especially while using very deep models like resnet-18. Below is an example of how the model performed slightly better when trained with batch normalization after every conv2d layer.

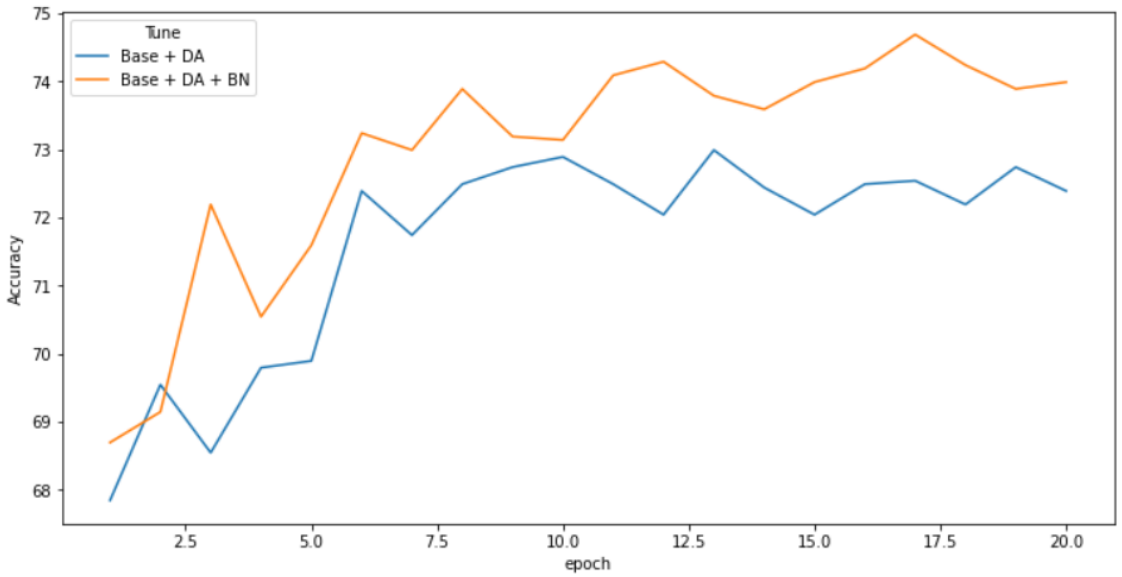



*Figure 31*

## A.3 Impact of Data Augmentation

Data Augmentation played a crucial role in making the model more generalized and increasing the test accuracy scores. DA helped in adding a lot of variation to the data, what DA policies to keep and what to drop was mostly decided from domain knowledge, experimenting, and sometimes from the model explainable output.

```
if train:
    if horizontal_flip_prob > 0:  # Horizontal Flip
        transforms_list += [A.HorizontalFlip(p=horizontal_flip_prob)]
    if vertical_flip_prob > 0:  # Vertical Flip
        transforms_list += [A.VerticalFlip(p=vertical_flip_prob)]
    if gaussian_blur_prob > 0:  # Patch Gaussian Augmentation
        transforms_list += [A.GaussianBlur(p=gaussian_blur_prob)]
    if rotate_degree > 0:  # Rotate image
        transforms_list += [A.Rotate(limit=rotate_degree)]

    if cutout > 0:  # CutOut
        transforms_list += [A.CoarseDropout(
            p=cutout, max_holes=1, fill_value=tuple([x * 255.0 for x in mean]),
            max_height=cutout_height, max_width=cutout_width, min_height=1, min_width=1
        )]

    transforms_list +=[A.augmentations.transforms.ColorJitter(brightness=(0.01,0.1),contrast=(0.01,0.1),
                                                              saturation=(0.01,0.1), hue=(-0.15,0.15), always_apply=False
```

*Figure 32(code)*

## A.4 Optimizers and LR schedulers

I have experimented with three optimizers – SGD with Momentum, RMSProp, and Adam. And two LR schedulers – StepLR & ReduceLROnPlateau. The result variation due to the LR scheduler change is very minimal to showcase. The results for swapping optimizers can be seen in the fig below. SGD with momentum yielded the best results out of the bunch by converging significantly faster compared to the other two.

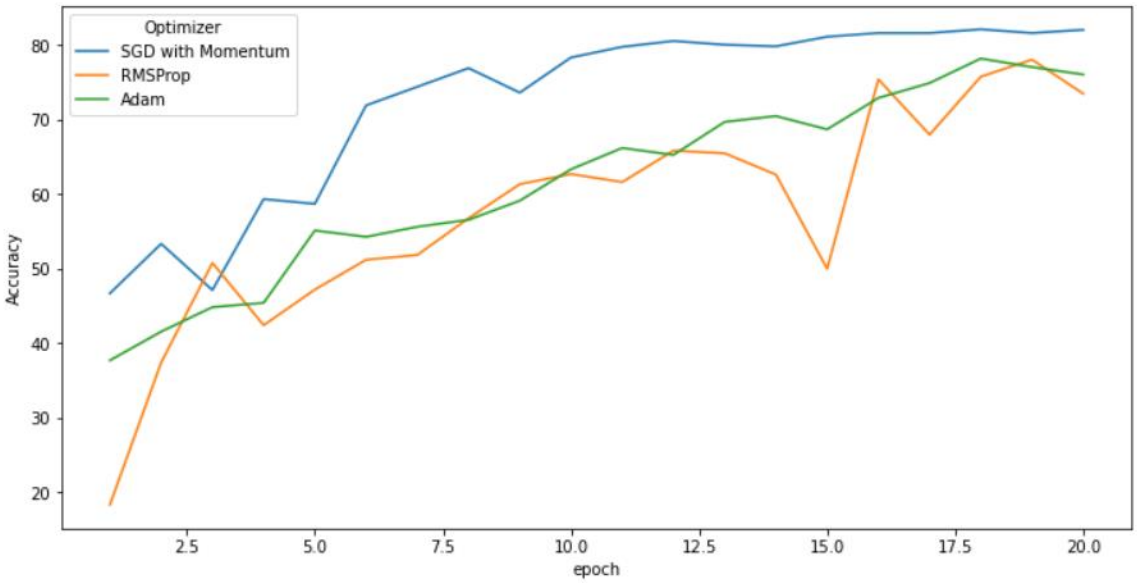



*Figure 33*

# Appendix B

## Mathematical Concepts

### B.1 Receptive Field

Neuron's receptive field is the patch of the total field of view. In other words, what information a single neuron has access to can be considered its receptive field. The general idea is to make sure the neurons in the final layers have seen the entire image. To add enough layers to satisfy the condition we can use the following formulae.

*Equation 1*

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

$n_{in}$: number of input features
$n_{out}$: number of output features
$k$: convolution kernel size
$p$: convolution padding size
$s$: convolution stride size

You can see the calculations of RF while building the model in the below images (commented RF value, jump value)

```
class Net(nn.Module):
    def __init__(self):
        """ This function instantiates all the model layers """
        super(Net, self).__init__()

        dropout_rate = 0.01

        self.convblock1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(8),
            nn.Dropout(dropout_rate)
        ) # Input: 224x224x3 | Output: 224x224x3 | RF: 3x3

        self.convblock2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.Dropout(dropout_rate)
        ) # Input: 224x224x3 | Output: 224x224x3 | RF: 5x5

        self.convblock21 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=8, kernel_size=1),
            nn.ReLU(),
            nn.BatchNorm2d(8),
            nn.Dropout(dropout_rate)
        ) # Input: 6x6x16 | Output: 6x6x10 | RF: 18x18


        self.pool1 = nn.MaxPool2d(2, 2)  # Input: 224x224x3 | Output: 112x112x3 | RF: 6x6      jump = 1 -> 2

        self.convblock3 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.Dropout(dropout_rate)
        ) # Input: 112x112x3   | Output:  112x112x8   | RF: 10x10    (+4)

        self.convblock4 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=16, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.Dropout(dropout_rate)
        ) # Input: 112x112x8    | Output:  112x112x8    | RF: 14x14

        self.convblock5 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Dropout(dropout_rate)
        ) # Input: 112x112x8    | Output:  112x112x8    | RF: 18x18

        self.convblock51 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=8, kernel_size=1),
            nn.ReLU(),
            nn.BatchNorm2d(8),
            nn.Dropout(dropout_rate)
        ) # Input: 112x112x8    | Output:  112x112x8    | RF: 18x18


        self.pool2 = nn.MaxPool2d(2, 2)  # Input: 112x112x8 | Output: 56x56x5 | RF: 20x20       jump = 2 -> 4
```

```
        self.convblock6 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(8),
            nn.Dropout(dropout_rate)
        ) # Input: 56x56x5 | Output:  56x56x8 | RF: 28x28   (+8)

        self.convblock7 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.Dropout(dropout_rate)
        ) # Input: 56x56x8 | Output:  56x56x16 | RF: 36x36

        self.convblock8 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Dropout(dropout_rate)
        ) # Input: 56x56x16 | Output:  56x56x32 | RF: 44x44

        self.convblock81 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=8, kernel_size=1),
            nn.ReLU(),
            nn.BatchNorm2d(8),
            nn.Dropout(dropout_rate)
        ) # Input: 56x56x32 | Output:  56x56x8 | RF: 44x44


        self.pool3 = nn.MaxPool2d(2, 2)  # Input: 56x56x16 | Output: 28x28x16 | RF: 48x48      jump = 4 -> 8

        self.convblock9 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(16),
            nn.Dropout(dropout_rate)
        ) # Input: 28x28x16 | Output: 28x28x16 | RF: 64x64         (+16)

        self.convblock10 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Dropout(dropout_rate)
        ) # Input: Input: 28x28x16 | Output: 28x28x16 | RF: 80x80


        self.pool4 = nn.MaxPool2d(2, 2)  # Input: 56x56x16 | Output: 28x28x16 | RF: 88x88      jump = 8 -> 16

        self.convblock11 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Dropout(dropout_rate)
        ) # Input: 14x14x32 | Output: 14x14x32 | RF: 120x120     (+32)

        self.convblock12 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding = "same"),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Dropout(dropout_rate)
        ) # Input: 14x14x32 | Output: 14x14x64 | RF: 152x152


        self.pool5 = nn.MaxPool2d(2, 2)  # Input: 28x28x64 | Output: 14x14x64 | RF: 168x168      jump = 16  > 32
```

*Figure 34(code)*

```
self.convblock13 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=7, kernel_size=1, padding = "same"),
    nn.ReLU(),
    nn.BatchNorm2d(7),
    nn.Dropout(dropout_rate)
) # Input: 7x7x64 | Output: 7x7x7 | RF: 232x232   (+64)


self.gap = nn.Sequential(
    nn.AdaptiveAvgPool2d(1)
) # Input: 6x6x10 | Output: 1x1x10 | RF: 28x28
```

*Figure 35(code)*

## B.2 Batch Normalization

To normalize an image, we simply choose the pixel with the highest intensity and one with the lowest intensity, and then we calculate a factor that scales the highest intensity to white and the lowest intensity to black. This produces the normalised image by applying it to each and every pixel in the image.

*Equation 2*

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

## B.3 SGD with Momentum

Stochastic Gradient Descent refers to an algorithm that operates on a batch size equal to 1. Adding momentum to the intensity of updates almost always works better and faster than regular SGD. So, By adding an exponentially weighted moving average for the gradients, a more reliable solution can be built.

The concept is simple: to prevent an abrupt change when the surface becomes flat, we can link a portion of the gradient's history to the correction factor rather than just taking into account the gradient's current value.

*Equation 3*

$$\begin{cases} v^{(t+1)} = \mu v^{(t)} - \alpha \nabla_\theta L(\bar{\theta}) \\ \bar{\theta}^{(t+1)} = \bar{\theta}^{(t)} + v^{(t+1)} \end{cases}$$

## B.4 RMSProp

This approach, put out by G. Hinton, is based on the notion of tailoring the correction factor for each parameter, increasing the influence on parameters with gradual changes and decreasing it on parameters with large changes in magnitude. This method can significantly improve a deep network's performance, but it is slightly more expensive than Momentum since we must compute a speed term for each parameter:

*Equation 4*

$$v^{(t+1)}(\theta_i) = \mu v^{(t)}(\theta_i) + (1-\mu)\left(\nabla_\theta L(\bar{\theta})\right)^2$$

## B.5 Adam

A possible extension of RMSProp is the adaptive algorithm known as Adam. It computes the same value for the gradient itself rather than just taking into account the gradient square's single exponentially weighted moving average. (Goodfellow I.)

Like in the other algorithms, forgetting factors μ1 and μ2 are used here too. The author suggests to use numbers higher than 0.9. Adam also provides a bias correction for both terms since both terms are moving estimates of the first and second moment and could be biased.

*Equation 5*

$$\begin{cases} g^{(t+1)}(\theta_i) = \mu_1 g^{(t)}(\theta_i) + (1-\mu_1)\nabla_\theta L(\bar{\theta}) \\ v^{(t+1)}(\theta_i) = \mu_2 v^{(t)}(\theta_i) + (1-\mu_2)\left(\nabla_\theta L(\bar{\theta})\right)^2 \end{cases}$$

*Equation 6*

$$\begin{cases} \hat{g}(\theta_i) = \dfrac{g^{(t+1)}(\theta_i)}{1-{\mu_1}^{(t)}} \\ \hat{v}(\theta_i) = \dfrac{v^{(t+1)}(\theta_i)}{1-{\mu_2}^{(t)}} \end{cases}$$

The parameter update rule becomes:

*Equation 7*

$$\bar{\theta}^{(t+1)} = \bar{\theta}^{(t)} - \frac{\alpha g^{(t+1)}(\bar{\theta})}{\sqrt{v^{(t+1)}(\bar{\theta}) + \delta}}$$

# Bibliography

(n.d.). Retrieved from HTTPS://WWW.BONACCORSO.EU/2017/10/03/A-BRIEF-AND-COMPREHENSIVE-GUIDE-TO-STOCHASTIC-GRADIENT-DESCENT-ALGORITHMS/

Adaloglou, N. (n.d.). *Understanding the receptive field of deep convolutional networks*. Retrieved from https://theaisummer.com/receptive-field/

Goodfellow I., B. Y. (n.d.). Deep Learning. *The MIT Press*.

Gruber, P., & Zito., P. M. (2022). Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK441949/

He, K. &. (2016). Deep Residual Learning for Image Recognition.

Ioffe, S. &., & Szegedy. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of Machine Learning Research*.

Khandelwal, R. (n.d.). *Understanding Deep Learning Models with Integrated Gradients*. Retrieved from Towards Data Science: https://towardsdatascience.com/understanding-deep-learning-models-with-integrated-gradients-24ddce643dbf

R. R. Selvaraju, M. C. (2017 ). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *IEEE International Conference on Computer Vision (ICCV), Venice, Italy*.

Research, Y. (n.d.). *Moshpit SGD: Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices*. Retrieved from Github: https://github.com/yandex-research/moshpit-sgd

S. ALBAWI, T. A.-Z. (2017). *"UNDERSTANDING OF A CONVOLUTIONAL NEURAL NETWORK,"*. I,INTERNATIONAL CONFERENCE ON ENGINEERING AND TECHNOLOGY (ICET), ANTALYA, TURKEY, 2017, PP. 1-6, .

Saxena, S. (n.d.). Retrieved from https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

Scott M. Lundberg, S.-I. L. (n.d.). A unified approach to interpreting model predictions.

Singla, S. (n.d.). Retrieved from Medium: https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf

Terrance DeVries, G. W. (2017 ). Improved Regularization of Convolutional Neural Networks with Cutout.

## Checklist

| S.NO | **Task** | **Expected date of completion** | **Names of Deliverables** | **Current Status** | **Comments** |
|---|---|---|---|---|---|
| 1 | Data Acquisition and preliminary Analysis | | | Completed | **kaggle/ datasets/ skin-cancer-dataset** |
| 2 | Exploration, research and Model experimentation | Jan-5-2022 | Documentation of research and solution/architecture. | Completed | |
| 3 | Model building& training. Solution building for explainable output | Jan-20-2022 | Model and pipeline building. Model Interpretation algorithm. **Model V1** | Completed | CNN Model built from scratch for this use case. Pre-built resnet also trained for the same. Notebooks attached |
| 4 | Integration testing and Fixing issues | Feb-4-2023 | Bug-free Pipeline flow | Completed | |
| 5 | Model fine tuning/architecture changes. Additional data collection/ data augmentations | Feb-12-2023 | Final Model and weights **Model V2** | Completed | Completed for Custom CNN and Resnet18. |
| 6 | Build UI for taking images as input to the model and Dockerize the entire code | Feb-20-2023 | UI and Dockerized code **Model V3** | Partial | Dockerization is yet to be completed |
| 7 | Deployment on a cloud platform and end-to-end testing | Feb-25-2023 | Solution Ready for a demo **Model V3.1** | Completed | |
| 8 | Draft Report Preparation | Mar-01-2023 | | Completed | |
| 9 | Final Report | Mar-08-2023 | | Completed | |

# Additional References


1. S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.
2. Wang, L.; Zhang, Y.; Xi, R. Study on image classification with convolution neural networks. In Proceedings of the 5th International Conference on Intelligence Science and Big Data Engineering, Suzhou, China, 14–16 June 2015; Volume 9242, pp. 310–319.
3. Tompson, J.; Goroshin, R.; Jain, A.; LeCun, Y.; Bregler, C. Efficient object localization using Convolutional Networks. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 648–656.
4. arXiv:1312.4400 [cs.NE]
5. Stats StackExchange: https://stats.stackexchange.com/questions/194142/what-does-1x1-convolution-mean-in-a-neural-network
6. Google Inception Architecture: Going Deeper with Convolutions by Lin et al.
7. Article: https://iamaaditya.github.io/2016/03/one-by-one-convolution/
8. https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural-networks/
9. https://jmlr.org/proceedings/papers/v37/ioffe15.pdf
10. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra arXiv:1610.02391
11. https://www.bonaccorso.eu/2017/10/03/a-brief-and-comprehensive-guide-to-stochastic-gradient-descent-algorithms/ - ADAM