# Melanoma Skin Cancer Detection

## Abstract

In the realm of cancer, there exist over 200 distinct forms, with melanoma standing out as the most lethal type of skin cancer among them. The diagnostic protocol for melanoma typically initiates with clinical screening, followed by dermoscopic analysis and histopathological examination. Early detection of melanoma skin cancer is pivotal, as it significantly enhances the chances of successful treatment. The initial step in diagnosing melanoma skin cancer involves visually inspecting the affected area of the skin. Dermatologists capture dermatoscopic images of the skin lesions using high-speed cameras, which yield diagnostic accuracies ranging from 65% to 80% for melanoma without supplementary technical assistance. Through further visual assessment by oncologists and dermatoscopic image analysis, the overall predictive accuracy of melanoma diagnosis can be elevated to 75% to 84%. The objective of the project is to construct an automated classification system leveraging image processing techniques to classify skin cancer based on images of skin lesions.

## Problem statement

To build a CNN based model which can accurately detect melanoma. Melanoma is a type of cancer that can be deadly if not detected early. It accounts for 75% of skin cancer deaths. A solution which can evaluate images and alert the dermatologists about the presence of melanoma has the potential to reduce a lot of manual effort needed in diagnosis.

## Table of Contents

- [General Info](#general-information)

- [Model Architecture](#model-architecture)

- [Model Summary](#model-summary)

- [Model Evaluation](#model-evaluation)

- [Technologies Used](#technologies-used)

- [Acknowledgements](#acknowledgements)

- [Collaborators](#collaborators)

<!-- You can include any other section that is pertinent to your problem -->

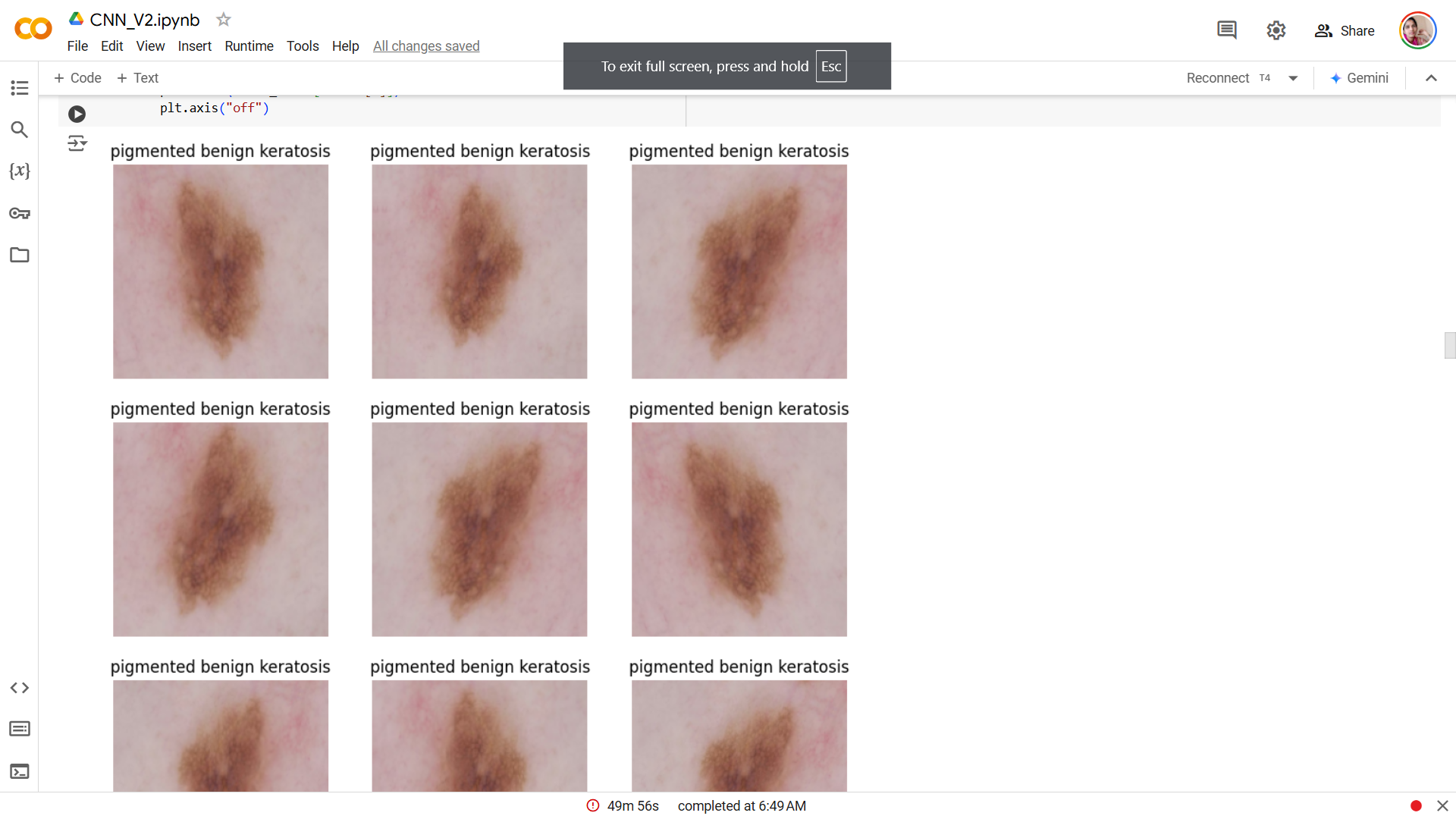
## General Information

The dataset comprises 2357 images depicting malignant and benign oncological conditions, sourced from the International Skin Imaging Collaboration (ISIC). These images were categorized based on the classification provided by ISIC, with each subset containing an equal number of images.

![datasetgraph](./class\_distribution.png)

In order to address the challenge of class imbalance, the Augmentor Python package (https://augmentor.readthedocs.io/en/master/) was employed to augment the dataset. This involved generating additional samples for all classes, ensuring that none of the classes had insufficient representation.

## Pictorial representation of skin types



The aim of this task is to assign a specific class label to a particular type of skin cancer.

## **Model Architecture**

The break down of the final provided CNN architecture step by step:

1. \*\*Data Augmentation\*\*: The `augmentation\_data` variable refers to the augmentation techniques applied to the training data. Data augmentation is used to artificially increase the diversity of the training dataset by applying random transformations such as rotation, scaling, and flipping to the images. This helps in improving the generalization capability of the model.

2. \*\*Normalization\*\*: The `Rescaling(1./255)` layer is added to normalize the pixel values of the input images. Normalization typically involves scaling the pixel values to a range between 0 and 1, which helps in stabilizing the training process and speeding up convergence.

3. \*\*Convolutional Layers\*\*: Three convolutional layers are added sequentially using the `Conv2D` function. Each convolutional layer is followed by a rectified linear unit (ReLU) activation function, which introduces non-linearity into the model. The `padding='same'` argument ensures that the spatial dimensions of the feature maps remain the same after convolution. The number within each `Conv2D` layer (16, 32, 64) represents the number of filters or kernels used in each layer, determining the depth of the feature maps.

4. \*\*Pooling Layers\*\*: After each convolutional layer, a max-pooling layer (`MaxPooling2D`) is added to downsample the feature maps, reducing their spatial dimensions while retaining the most important information. Max-pooling helps in reducing computational complexity and controlling overfitting.

5. \*\*Dropout Layer\*\*: A dropout layer (`Dropout`) with a dropout rate of 0.2 is added after the last max-pooling layer. Dropout is a regularization technique used to prevent overfitting by randomly dropping a fraction of the neurons during training.

6. \*\*Flatten Layer\*\*: The `Flatten` layer is added to flatten the 2D feature maps into a 1D vector, preparing the data for input into the fully connected layers.

7. \*\*Fully Connected Layers\*\*: Two fully connected (dense) layers (`Dense`) are added with ReLU activation functions. The first dense layer consists of 128 neurons, and the second dense layer outputs the final classification probabilities for each class label.

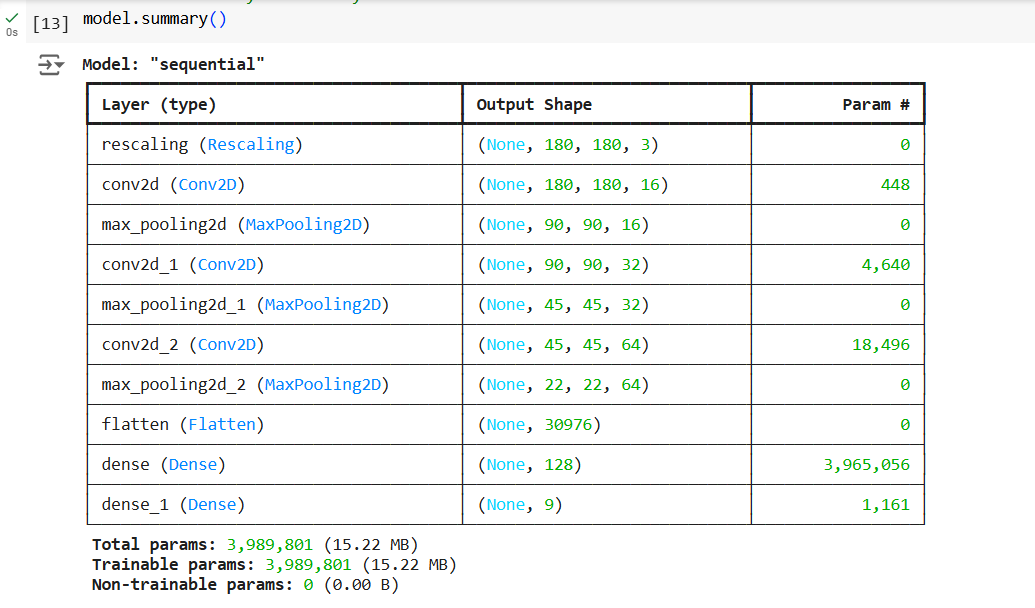
8. \*\*Output Layer\*\*: The number of neurons in the output layer is determined by the `target\_labels` variable, representing the number of classes in the classification task. The output layer does not have an activation function specified, as it is followed by the loss function during training.

9. \*\*Model Compilation\*\*: The model is compiled using the Adam optimizer (`optimizer='adam'`) and the Sparse Categorical Crossentropy loss function (`loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)`), which is suitable for multi-class classification problems. Additionally, accuracy is chosen as the evaluation metric (`metrics=['accuracy']`).

10. \*\*Training\*\*: The model is trained using the `fit` method with the specified number of epochs (`epochs=50`). The `ModelCheckpoint` and `EarlyStopping` callbacks are employed to monitor the validation accuracy during training. The `ModelCheckpoint` callback saves the model with the best validation accuracy, while the `EarlyStopping` callback stops training if the validation accuracy does not improve for a specified number of epochs (patience=5 in this case). These callbacks help prevent overfitting and ensure that the model converges to the best possible solution.

## Model Summary

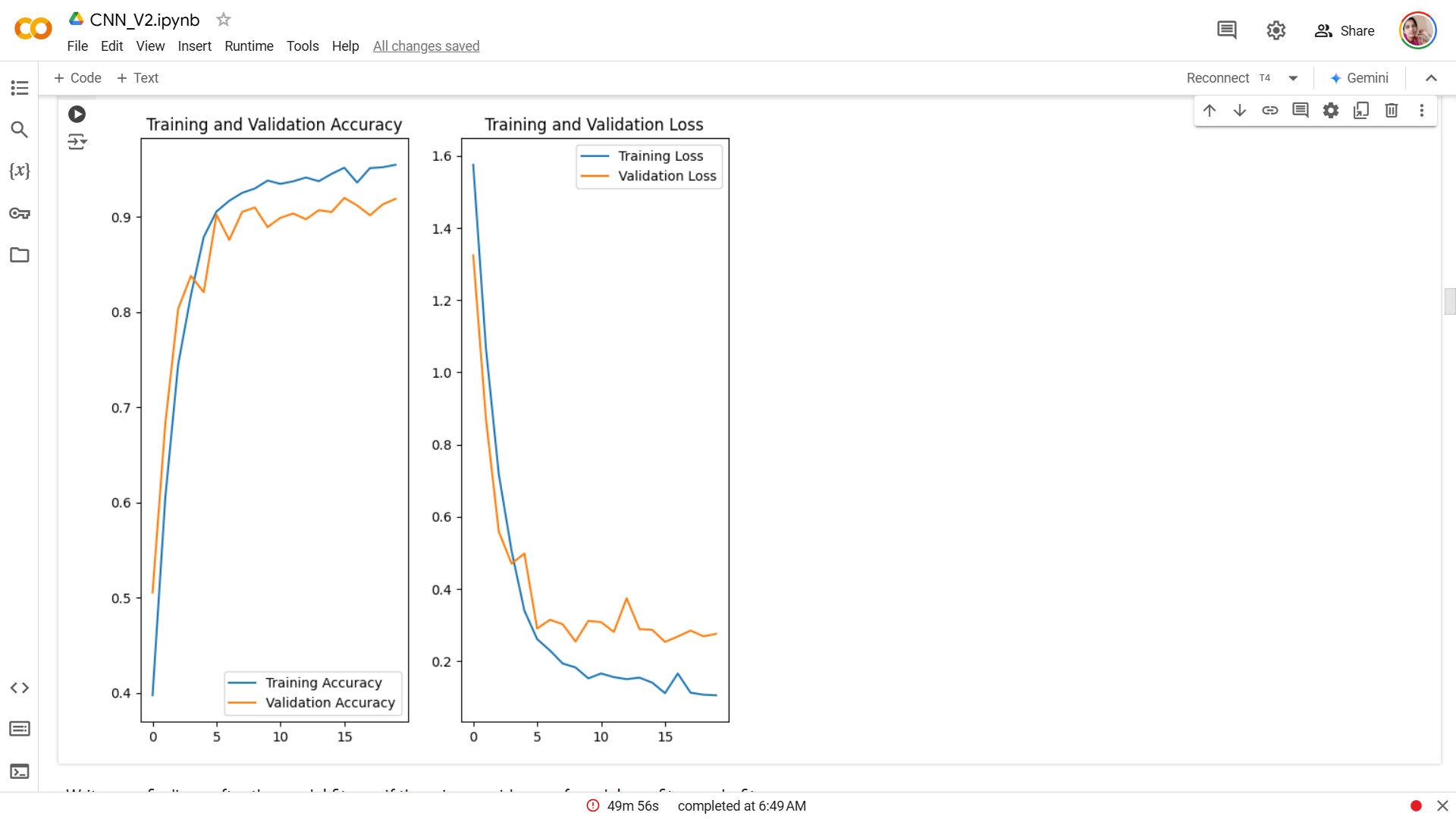
Created a CNN model, which can accurately detect 9 classes present in the dataset. Used keras.preprocessing.Rescaling to normalize pixel values between (0,1). The RGB channel values are in the [0, 255] range. This is not ideal for a neural network. Here, it is good to standardize values to be in the [0, 1]



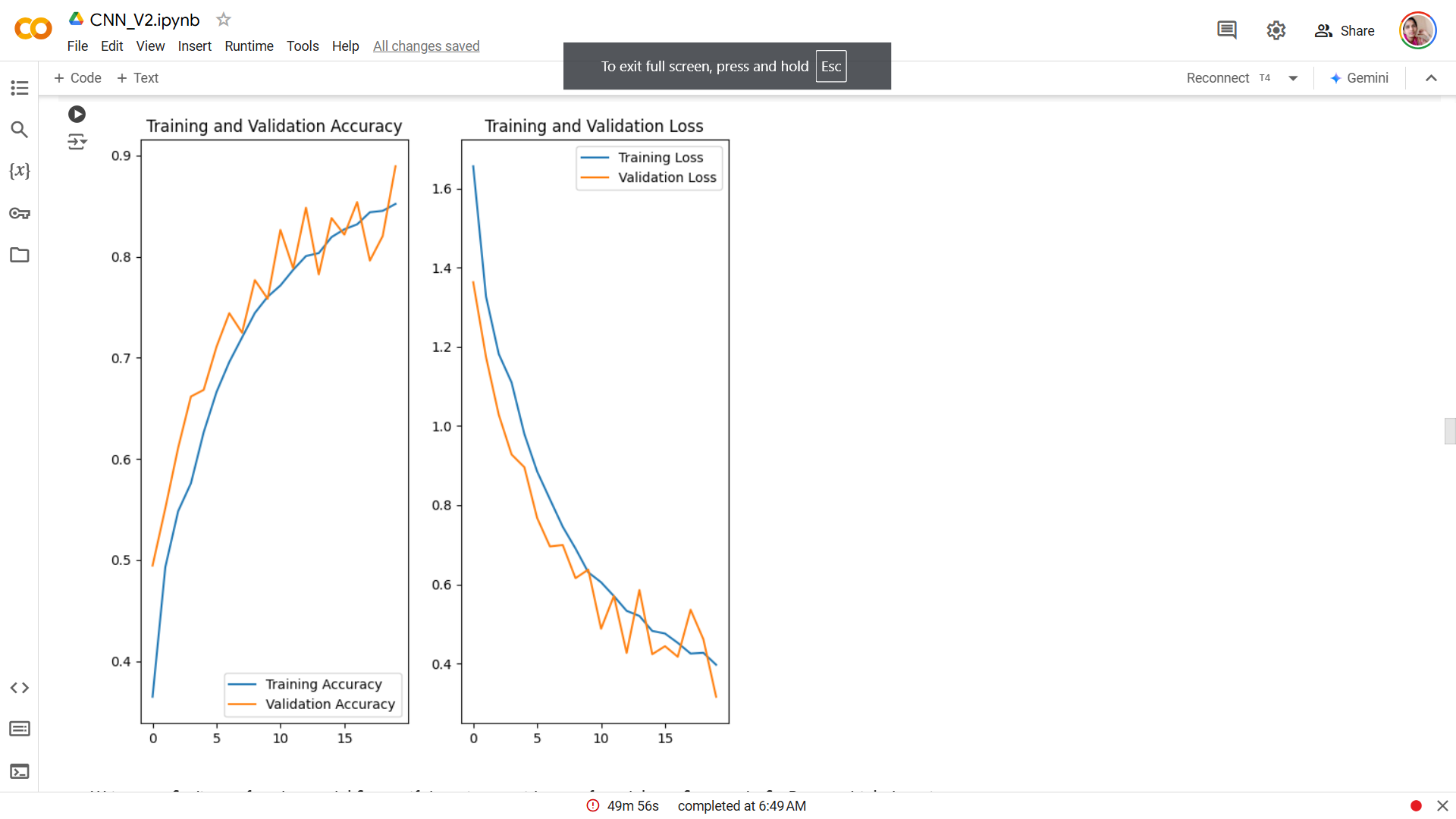
## **Model Evaluation**

1. The model's training accuracy shows a steady increase of upto 90%, while validation accuracy remains consistently around 55%. A high training accuracy suggests the model has effectively captured the noise within the data. However, its poor performance on validation data indicates a lack of generalizability.

These observations confirm the presence of overfitting. To address this issue, augmentation techniques will be employed. Given the limited training data available, new samples will be generated by making slight modifications to existing data (such as horizontal/vertical image flipping, minor image rotations, etc.), which will then be used for training the model.

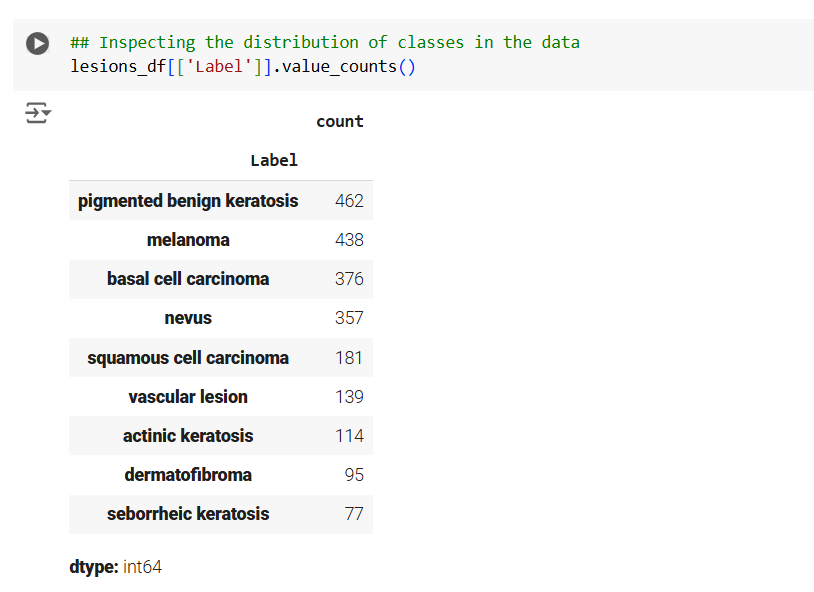


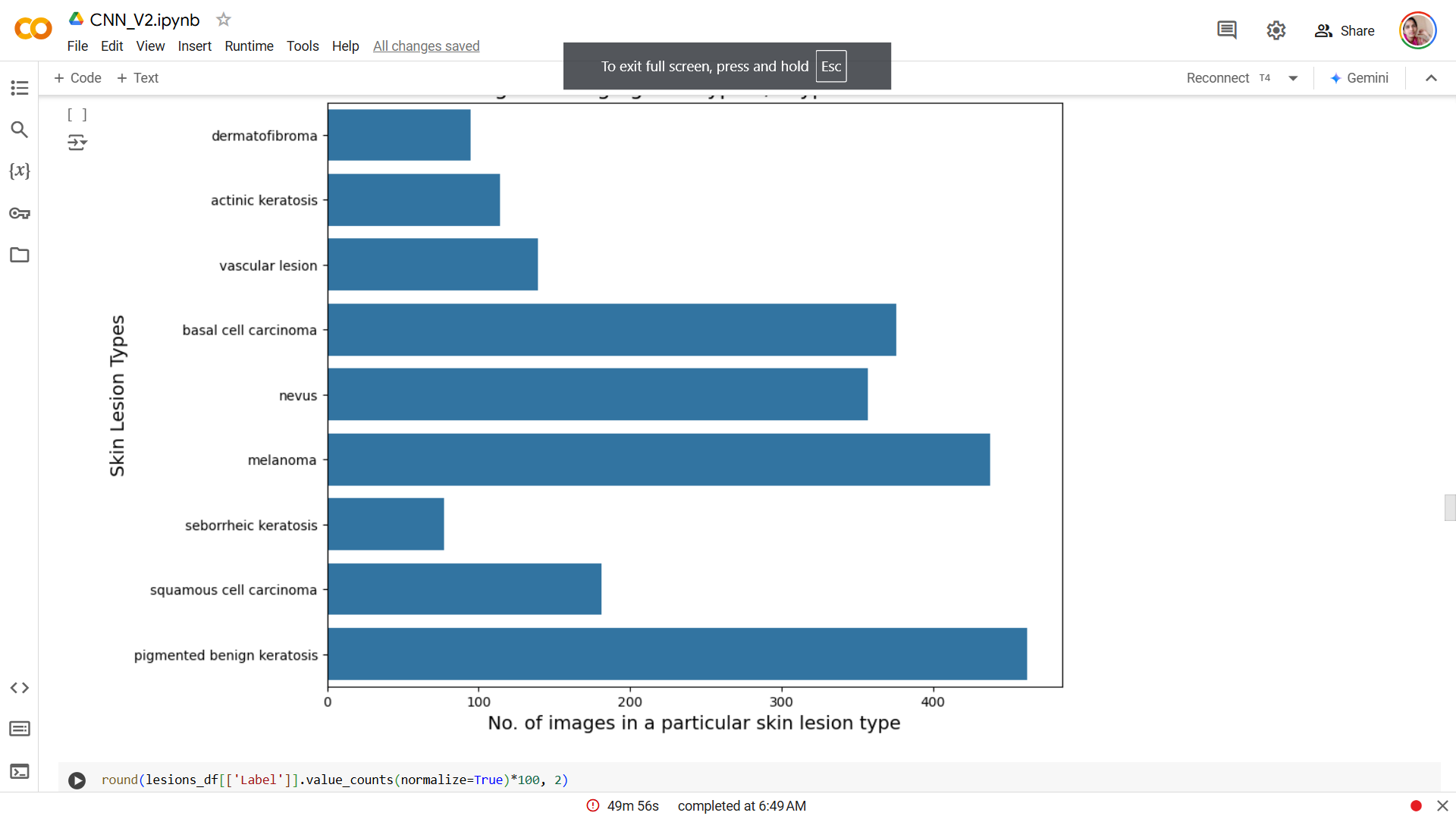
By leveraging augmented data, overfitting has been effectively addressed.The training and validation accuracy of the aforementioned model fall within a similar range. Both the training and validation data are showing poor performance, indicating that the model is now exhibiting underfitting.



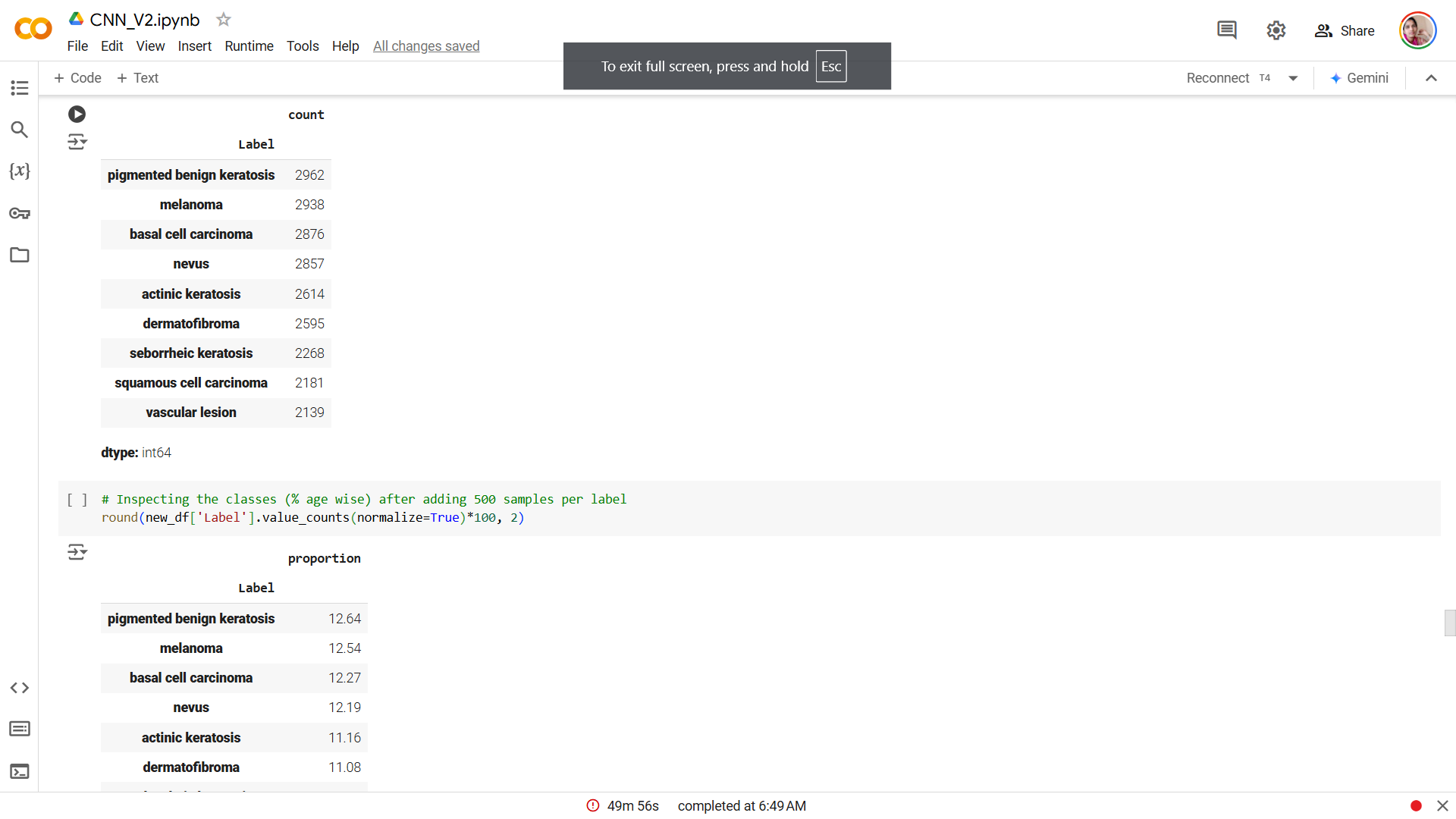
By leveraging augmented data, overfitting has been effectively addressed.The training and validation accuracy of the aforementioned model fall within a similar range. Both the training and validation data are showing poor performance, indicating that the model is now exhibiting underfitting.

#### ****Context:**** Many times real life datasets can have class imbalance, one class can have proportionately higher number of samples compared to the others. Class imbalance can have a detrimental effect on the final model quality. Hence as a sanity check it becomes important to check what is the distribution of classes in the data.





After using augmentor class imbalance is solved



1. Implementing batch normalization in the model yielded a slight uptick in both training and validation accuracy.
2. Formulating an alternative model without batch normalization aims to investigate prospective improvements in accuracy.
3. The ultimate model showcases well-balanced performance, displaying no signs of underfitting or overfitting.
4. The implementation of class rebalancing has notably enhanced the model's performance across both training and validation datasets.
5. Following 37 epochs, the final model attains an accuracy of 84% on the training set and approximately 79% on the validation set.
6. The narrow divergence between training and validation accuracies underscores the robust generalization capability of the final CNN model.
7. The addition of batch normalization failed to enhance both training and validation accuracy.

**Observations:**

1. The ultimate model showcases well-balanced performance, displaying no signs of underfitting or overfitting.
2. The implementation of class rebalancing has notably enhanced the model's performance across both training and validation datasets.
3. Following 37 epochs, the final model attains an accuracy of 84% on the training set and approximately 79% on the validation set.
4. The narrow divergence between training and validation accuracies underscores the robust generalization capability of the final CNN model.
5. The addition of batch normalization failed to enhance both training and validation accuracy.

## Technologies Used

- [Python](https://www.python.org/) - version 3.11.4

- [Matplotlib](https://matplotlib.org/) - version 3.7.1

- [Numpy](https://numpy.org/) - version 1.24.3

- [Pandas](https://pandas.pydata.org/) - version 1.5.3

- [Seaborn](https://seaborn.pydata.org/) - version 0.12.2

- [Tensorflow](https://www.tensorflow.org/) - version 2.15.0

<!-- As the libraries versions keep on changing, it is recommended to mention the version of library used in this project -->

## Acknowledgements

- UpGrad tutorials on Convolution Neural Networks (CNNs) on the learning platform

- [Melanoma Skin Cancer](https://www.cancer.org/cancer/melanoma-skin-cancer/about/what-is-melanoma.html)

- [Introduction to CNN](https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/)

- [Efficient way to build CNN architecture](https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7)

## Collaborators

Created by [@radhika\_g](https://github.com/radhikasridhar90/Melanoma\_Skin\_Cancer\_Detection)