High Risk Project

Image-Based Dermatological Diagnosis Using Deep Learning

Hugo Alberto Miranda Burnes
UT EID: hm27834
AI in Healthcare
Master of Science in Artificial Intelligence
University of Texas at Austin

Table of Contents

Introduction

In dermatology, skin diseases can range from mild rashes to dangerous cancers. Detecting them early is essential for effective treatment. Recently, Artificial Intelligence tools have been developed to classify skin images into disease categories, helping with early diagnosis and timely care.

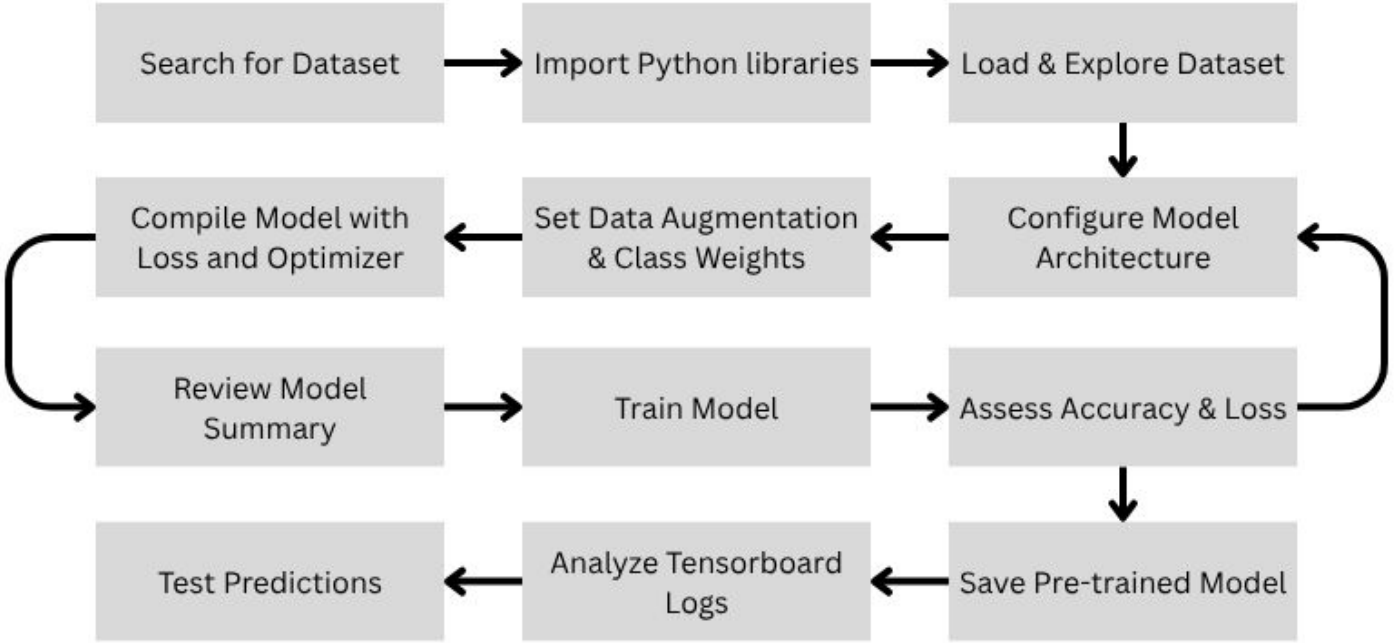
This presentation explains how a Convolutional Neural Network (CNN) was built and trained from scratch using a dataset of labeled skin condition images. The goal was to test different techniques to improve accuracy and gain a deeper understanding of how deep learning can be applied to healthcare.

According to the American Academy of Dermatology:

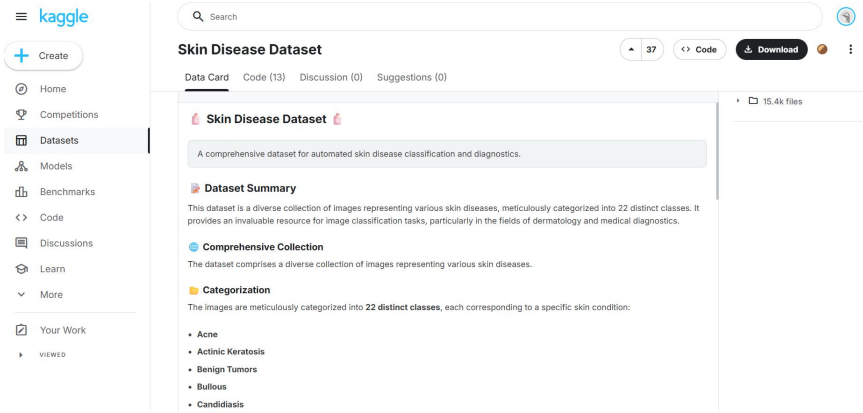
- Acne is the most common skin condition in the United States ; up to **50 million** people are affected by it every year[1]
- Skin cancer is the most common type of cancer in USA; around **20%** of the population develop skin cancer in their lifetime with **9,500** people in the U.S. are diagnosed with skin cancer every day. [2]

[1] Anon. Skin conditions by the numbers. Retrieved August 10, 2025 from https://www.aad.org/media/stats-numbers
[2] Anon. Skin cancer. Retrieved August 10, 2025 from https://www.aad.org/media/stats-skin-cancer

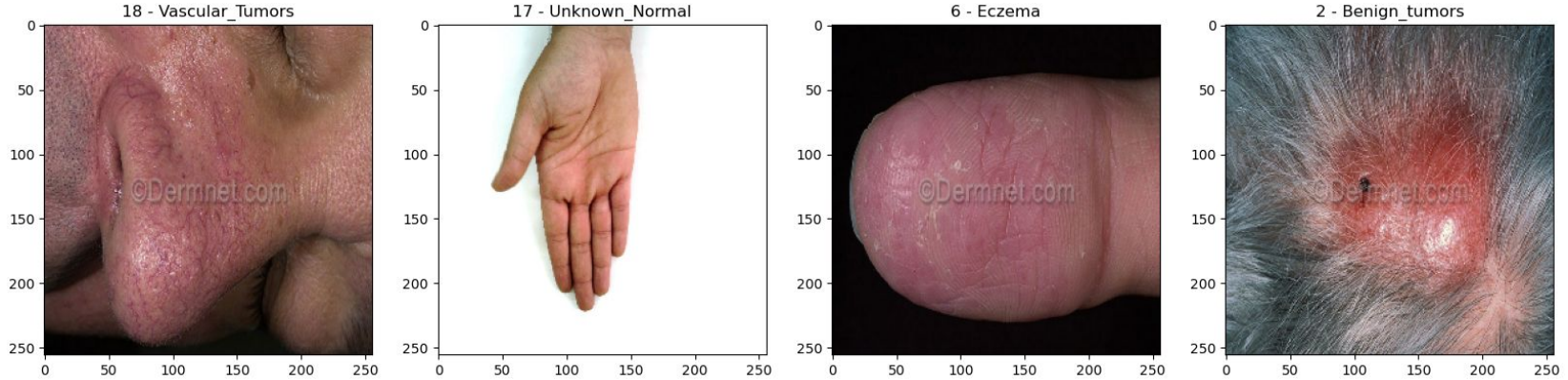
Methodology - Workflow



Methodology - Dataset and Python Libraries




```python
import tensorflow as tf
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Activation
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.utils import class_weight
from sklearn.utils.class_weight import compute_class_weight
```



Methodology - Model Architecture, Loss and Optimizer

```
model = Sequential()

model.add(Conv2D(16, (3, 3), input_shape=(256, 256, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D())

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D())
```

```
model.add(Conv2D(256, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D())

model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D())

model.add(GlobalAveragePooling2D())

model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.25))
model.add(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.25))
model.add(Dense(22, activation='softmax'))
```

```
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
num_epochs = 200
```

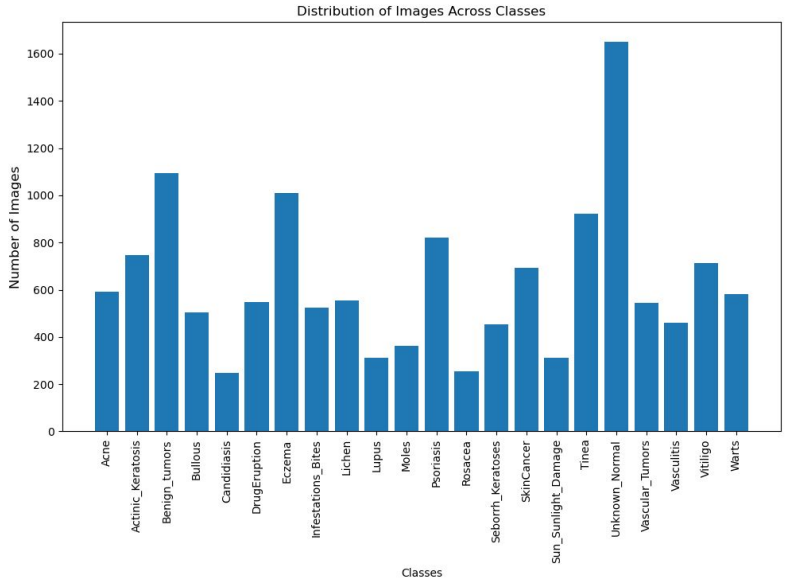
```
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

Methodology - Data Augmentation and Class Weights

The training process revealed that limited image variation caused overfitting, where the model memorized training patterns but performed poorly on validation data. This issue was resolved through data augmentation using *ImageDataGenerator* from Keras, which applied random transformations like rotation, shifting, flipping, brightness adjustment, and zooming to create a more diverse dataset and improve model generalization.

Another challenge was class imbalance, where some skin disease categories had far more images than others. This was addressed using Keras's *compute_class_weight*, which assigned weights to classes so the model treated each category fairly, reducing bias toward overrepresented classes and improving overall prediction balance.

```python
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)
```



Methodology - Model Summary

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 16)	448
batch_normalization (BatchNormalization)	(None, 256, 256, 16)	64
activation (Activation)	(None, 256, 256, 16)	0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4,640
batch_normalization_1 (BatchNormalization)	(None, 128, 128, 32)	128
activation_1 (Activation)	(None, 128, 128, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 64, 64, 64)	256
activation_2 (Activation)	(None, 64, 64, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73,856
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 128)	512
activation_3 (Activation)	(None, 32, 32, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 16, 16, 256)	295,168
batch_normalization_4 (BatchNormalization)	(None, 16, 16, 256)	1,024
activation_4 (Activation)	(None, 16, 16, 256)	0
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 256)	0
conv2d_5 (Conv2D)	(None, 8, 8, 512)	1,180,160
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 512)	2,048
activation_5 (Activation)	(None, 8, 8, 512)	0
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 512)	262,656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 22)	5,654

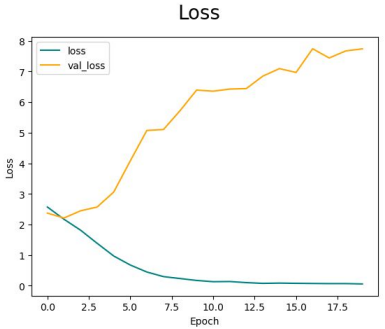
```
Total params: 1,976,438 (7.54 MB)

Trainable params: 1,974,422 (7.53 MB)

Non-trainable params: 2,016 (7.88 KB)
```

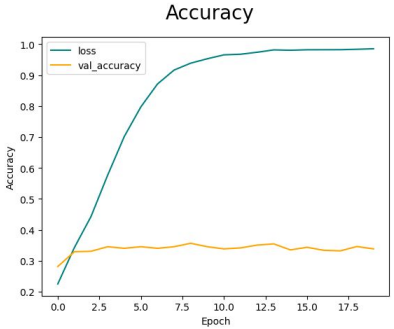
Methodology - Training Model and Assessing Accuracy and Loss

Attempt #1

Results: Overfitting the data with nearly 99% accuracy for training data but stuck at around 30% accuracy for validation.

- 3 - 2D Convolution Networks using ReLU and Max Pooling
- Flatten followed by 1 Dense layer with ReLU
- Output Dense layer for 22 channels with Softmax Activation
- 20 Epoch training

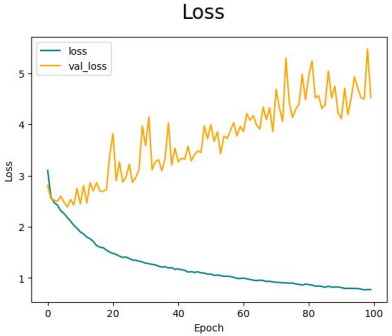


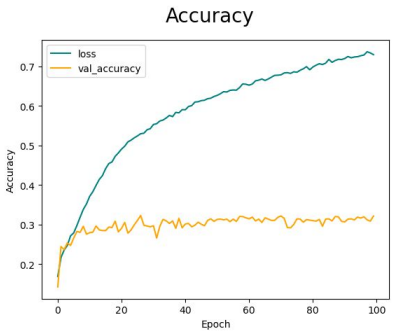


Attempt #2

Results: The model was still overfitting but the accuracy for training data decreased at around 70%.

- Batch Normalization
- L2 Regularizer at 0.001
- One extra Dense layer
- Dropout at 0.3 for both hidden Dense layers
- 100 Epoch training



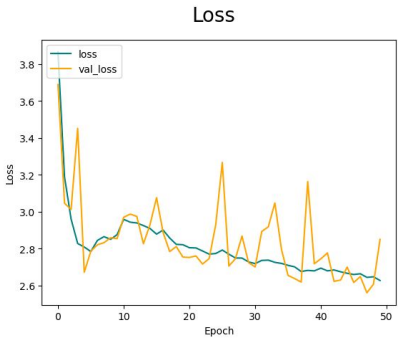


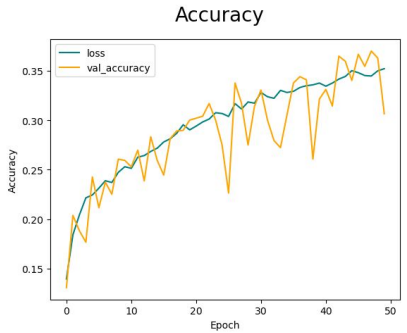
Methodology - Training Model and Assessing Accuracy and Loss

Attempt #3

Results: Overfitting was not present anymore since the validation accuracy and loss were now following roughly the same trends for the training accuracy, but it was overall very low at around 35%.

- Data Augmentation
- 50 Epoch training

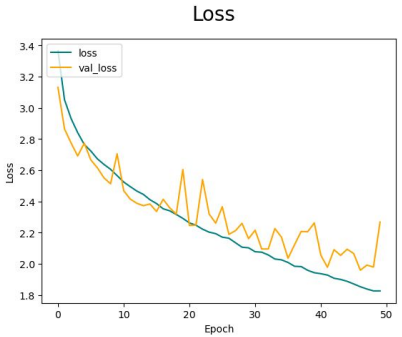


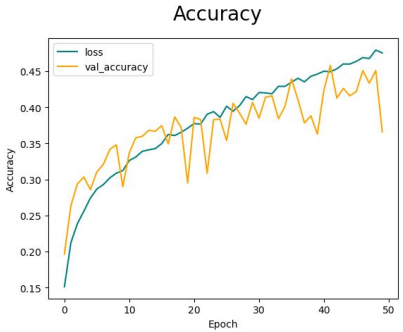


Attempt #4

Results: This configuration got similar results to the last attempt, but the training data accuracy increased to around 45%, while the validation accuracy followed close to 40%.

- 5 - 2D Convolution Networks using ReLU, Batch Normalization and Max Pooling
- Adam optimizer learning rate changed from 0.01 to 0.0001
- 50 Epoch training



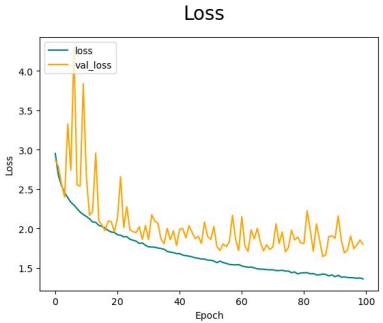


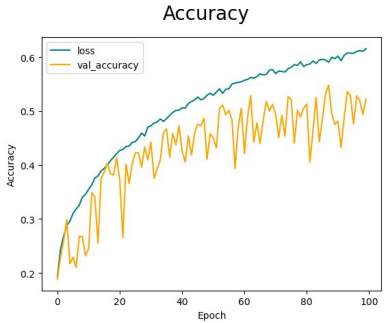
Methodology - Training Model and Assessing Accuracy and Loss

Attempt #5

Results: Training data accuracy improved to around 60% but the validation accuracy trailed behind at around 50%.

- 6 - 2D Convolution Networks using ReLU, Batch Normalization and Max Pooling
- Adam optimizer learning rate changed from 0.0001 to 0.001
- 100 Epoch training





Attempt #6

Results: This attempt was interrupted at 50 epochs due to showing very similar results to the last attempt at this point.

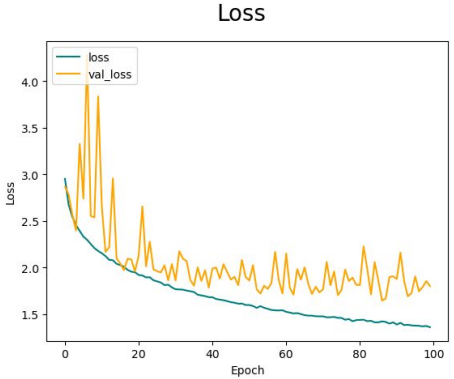
- Changed architecture order to Conv2d => BatchNorm => ReLU => MaxPool
- Dense layer at 512 channels
- Removed second Dense layer
- Dropout reduced to 0.15
- 100 Epoch training

Methodology - Training Model and Assessing Accuracy and Loss

Attempt #7

Results: The training data accuracy increased slightly at around 63% but the validation data accuracy seemed to be worse than before at the 45-50% range.

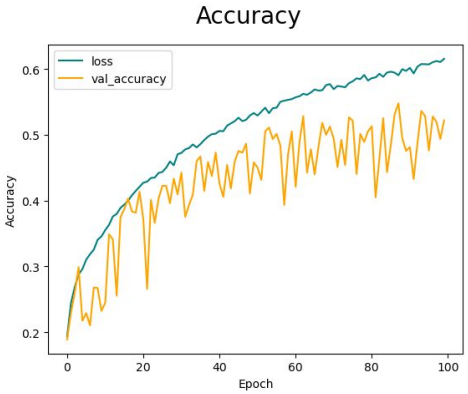
- Switched to using less initial layers (4 - Conv2d => BatchNorm => ReLU => MaxPool) but with increasing number of filters after each layer (32, 64, 128, 256)
- Switched Flatten for GlobalAveragePooling2D
- Tested with Dropout at 0.5
- Used Class Weights for the unbalanced data
- Adam optimizer learning rate changed back from 0.001 to 0.0001
- 100 Epoch training



Attempt #8

Results: This attempt was interrupted at 25 epochs due to showing a trend of worse results to the last attempt at this point, stopping at 30% training data accuracy.

- Removed Dropout
- Adam optimizer learning rate changed from 0.0001 to 0.01
- Used default L2 Regularizer
- 100 Epoch training

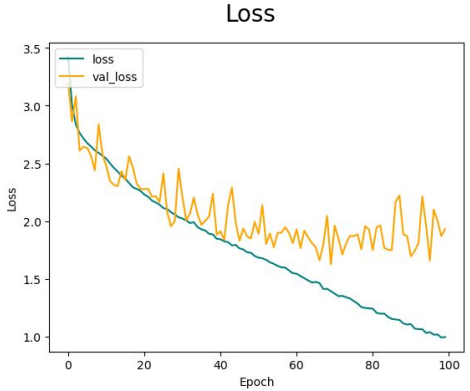


Methodology - Training Model and Assessing Accuracy and Loss

Attempt #9

Results: The training data accuracy went back up to around 70% while the validation data accuracy remained closer to 50%, but both trends showed projections of growth at greater epochs.

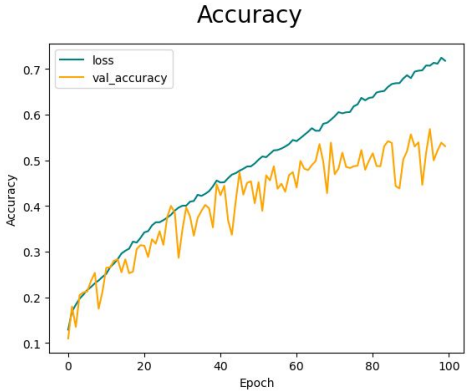
- 6 - CNN Layers increasing from 16 to 512 filters
- L2 Regularizer at 0.001
- Added padding to the CNNs
- Added back Dropout at 0.25
- Added one more Dense layer with 256 channels
- 100 Epoch training



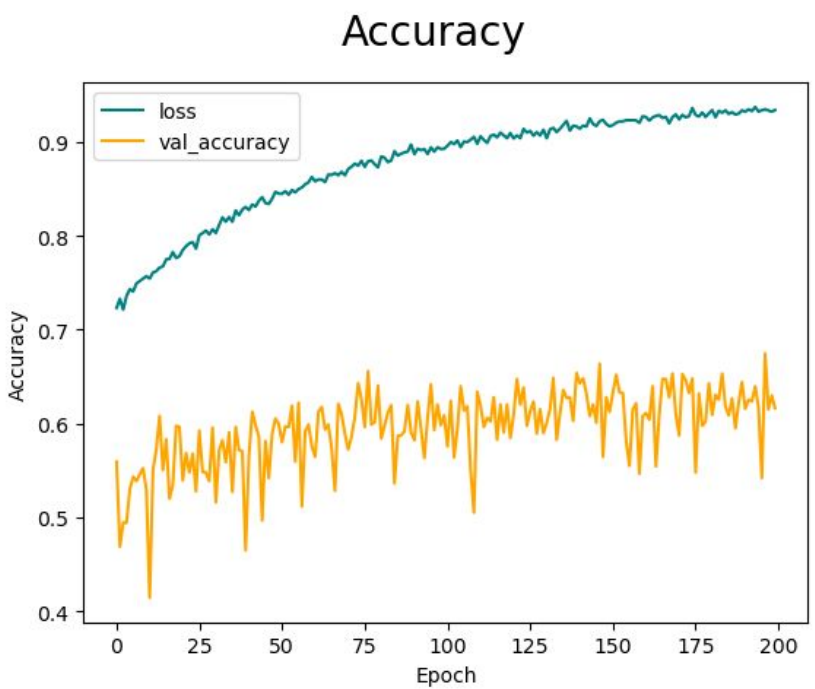
Attempt #10

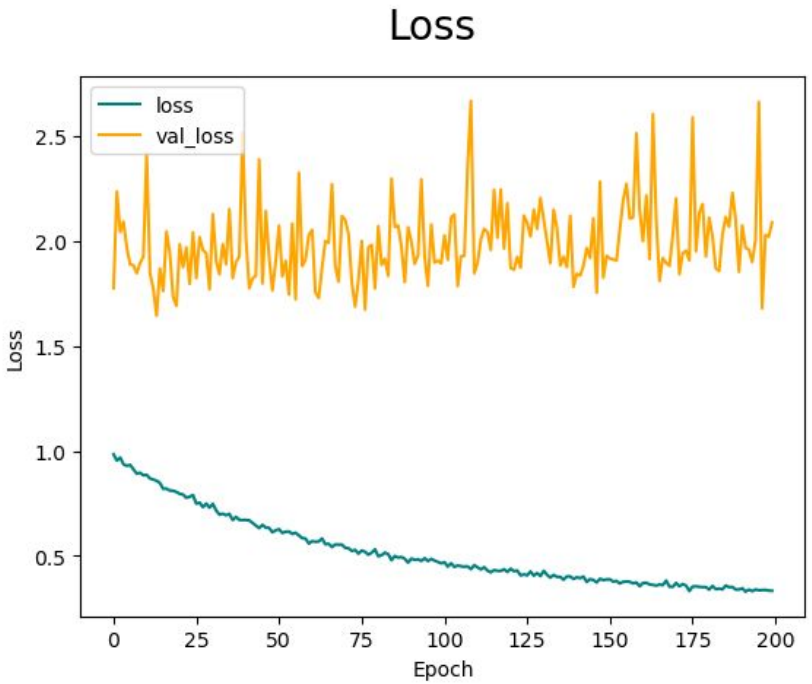
Results: Training data accuracy finalized above 90%, while the validation data accuracy deviated even more from the training data accuracy than the previous attempt, oscillating around 60%, which ended up being the highest validation accuracy obtained for this assignment.

- Training started using the same pre-trained model as the previous attempt but simply added 200 more epochs to review results for prolonged training with the same conditions for a total of 300 epochs.

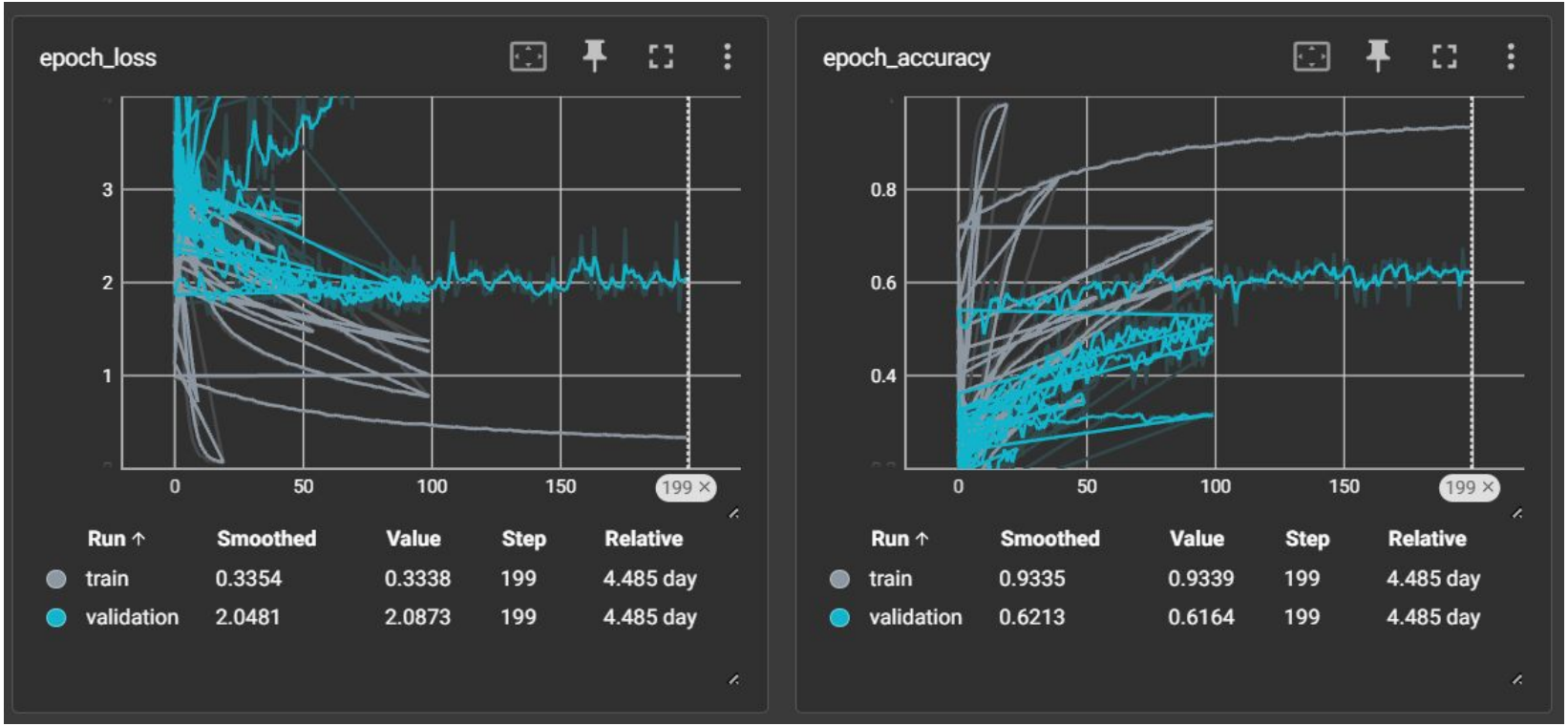


Results - Final Model Accuracy and Loss

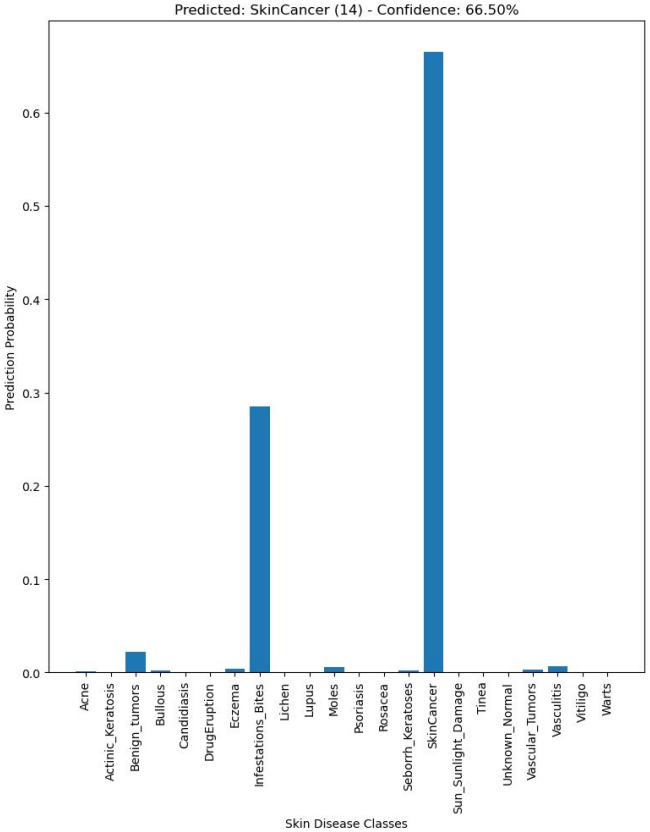




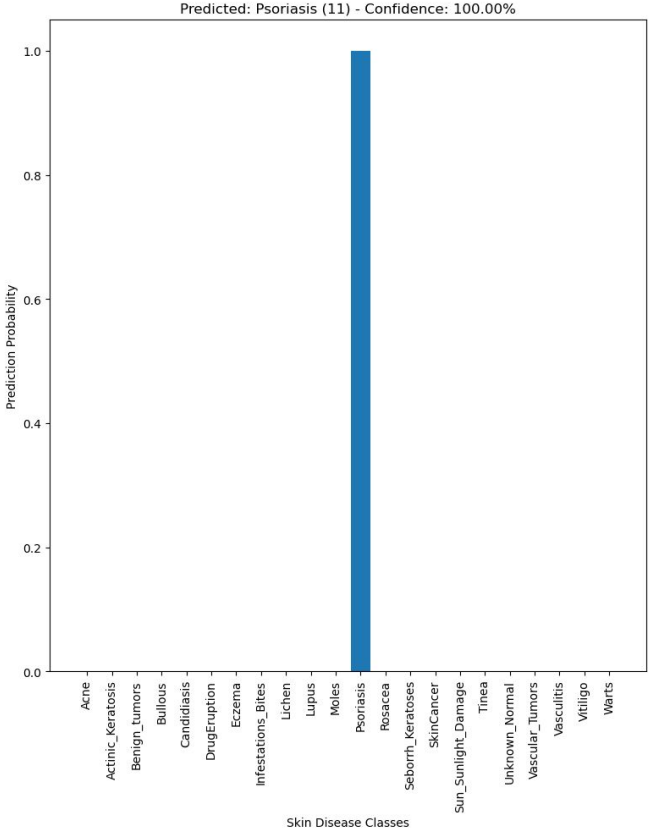
Results - Analyzing Tensorboard Logs



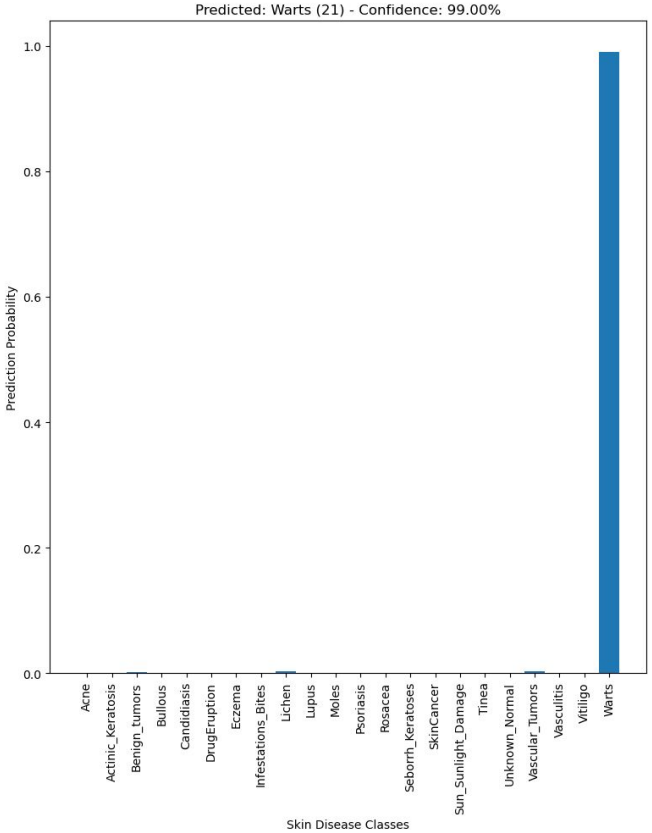
Results - Testing Predictions (Mixed Results)



Results - Testing Predictions (Correct Result)



Results - Testing Predictions (Incorrect Result)



Conclusion & Future Directions

For medical image analysis it could be said that for a prediction accuracy percentage to be decent or acceptable it should range somewhere 70 to 80%, depending on the exact task at hand. With this in mind, it is correct to imply that the model trained for this assignment failed the mark at ~60%.

The main reason for this could be the amount of data available per class, meaning trying to categorize this amount of images into 22 unique classes using Deep Learning algorithms presents a bigger challenge and perhaps it was too ambitious, however the goal for this assignment, as it name implies, was to present a high risk.

Future Directions:

- Using a larger Dataset (10x)
- Grouping skin diseases/conditions to lower the total amount of classes
- Use pre-trained models such as ResNet or EfficientNet