Skin Cancer Detection

CSCE 485 Computer and Machine Vision

Christopher Yang

November 18, 2024

Introduction:

So the topic I chose for this project was skin cancer detection. The reason why I chose this topic is because skin cancer often goes undetected by many people, and healthcare professionals sometimes miss early signs, leading to delayed treatment and, in some cases, preventable death. Thus, I wanted to try to help make skin cancer more detectable in early stages so as to prevent preventable deaths.

Dataset:

The dataset I chose to work with was the HAM10000 dataset. This dataset has approximately 2.9 GB of 10,015 images of various types of skin cancer. The dataset also contained information on the location where the skin cancer was found, and the gender of each individual.

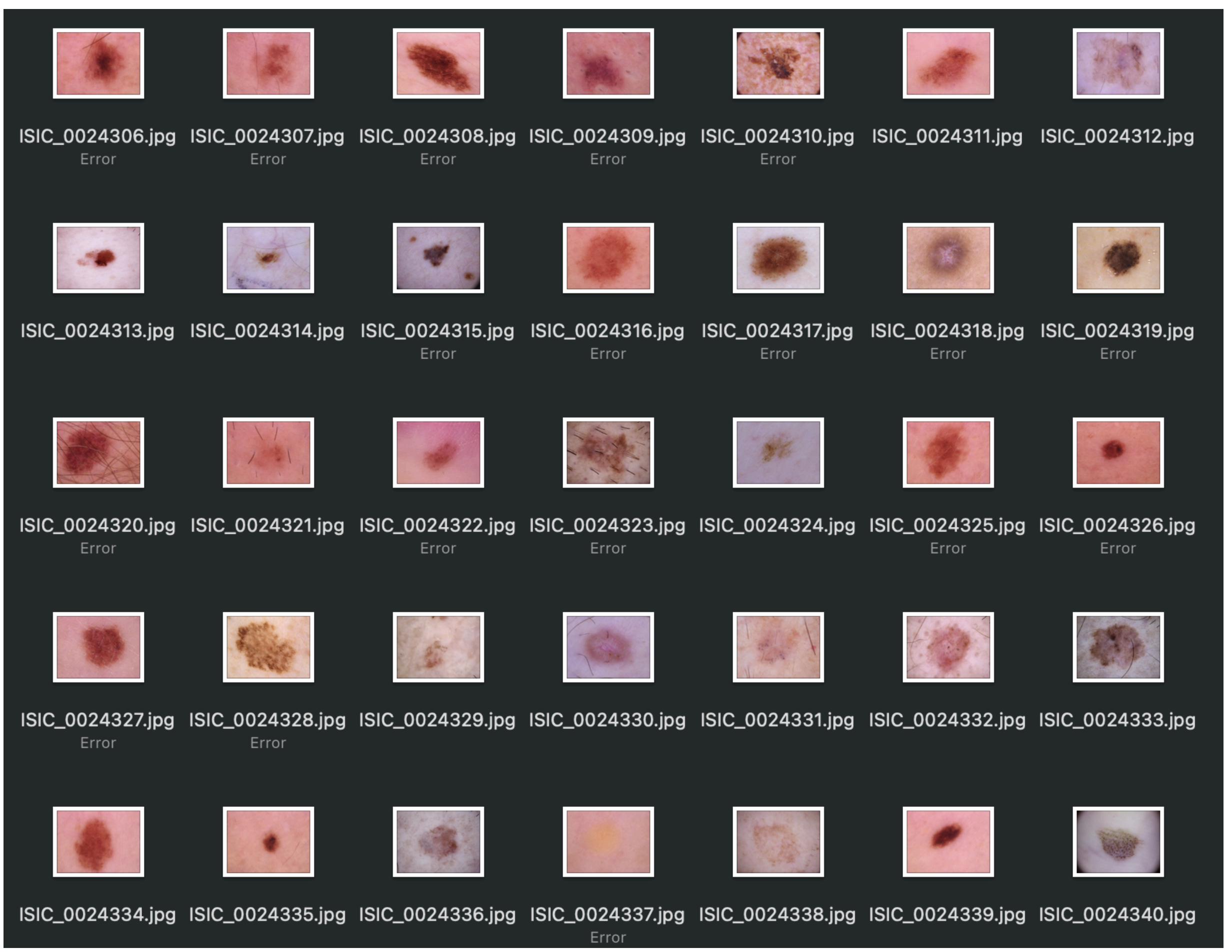


Fig. 1 : Sample dataset of skin cancer

HAM10000_metadata

lesion_id	image_id	dx	dx_type	age	sex	localization	dataset
HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	vidir_modern
HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	vidir_modern
HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	vidir_modern
HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	vidir_modern
HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	vidir_modern
HAM_0001466	ISIC_0027850	bkl	histo	75.0	male	ear	vidir_modern
HAM_0002761	ISIC_0029176	bkl	histo	60.0	male	face	vidir_modern
HAM_0002761	ISIC_0029068	bkl	histo	60.0	male	face	vidir_modern
HAM_0005132	ISIC_0025837	bkl	histo	70.0	female	back	vidir_modern
HAM_0005132	ISIC_0025209	bkl	histo	70.0	female	back	vidir_modern
HAM_0001396	ISIC_0025276	bkl	histo	55.0	female	trunk	vidir_modern
HAM_0004234	ISIC_0029396	bkl	histo	85.0	female	chest	vidir_modern
HAM_0004234	ISIC_0025984	bkl	histo	85.0	female	chest	vidir_modern
HAM_0001949	ISIC_0025767	bkl	histo	70.0	male	trunk	vidir_modern
HAM_0001949	ISIC_0032417	bkl	histo	70.0	male	trunk	vidir_modern
HAM_0007207	ISIC_0031326	bkl	histo	65.0	male	back	vidir_modern
HAM_0001601	ISIC_0025915	bkl	histo	75.0	male	upper extremity	vidir_modern

Fig. 2 : CSV - HAM10000_metadata

pixel0000	pixel0001	pixel0002	pixel0003	pixel0004	pixel0005	pixel0006	pixel0007	pixel0008	pixel0009	pixel0010	pixel0011	pixel0012	pixel0013	pixel001
192	153	193	195	155	192	197	154	185	202	162	192	208	165	20
25	14	30	68	48	75	123	93	126	158	128	158	172	136	16
192	138	153	200	145	163	201	142	160	206	149	165	207	149	15
38	19	30	95	59	72	143	103	119	171	125	134	177	118	12
158	113	139	194	144	174	215	162	191	225	179	214	232	189	22
8	1	3	19	5	10	26	8	13	34	13	24	100	69	7
194	147	137	197	148	139	197	148	132	200	154	142	202	156	15
161	121	105	169	128	119	172	129	116	176	134	125	181	134	12
125	84	85	165	114	118	181	120	125	188	133	142	189	132	14
228	179	194	227	174	191	226	165	182	215	157	175	206	149	16
23	13	16	22	12	14	29	20	24	76	63	65	122	107	10
159	129	127	166	136	135	170	140	142	170	137	128	173	139	13
37	22	22	100	74	61	145	119	111	155	128	126	159	133	13
155	131	142	160	135	146	165	139	157	168	142	162	172	147	16
193	164	192	194	162	190	195	166	193	195	168	191	197	172	19
197	158	179	201	160	182	201	160	182	202	163	181	203	167	18
171	133	142	176	141	153	180	144	153	183	140	149	193	154	16
212	160	173	214	157	173	219	169	185	220	168	183	220	159	17
166	131	138	174	139	145	185	155	163	187	154	162	199	162	18
217	179	200	219	183	203	219	183	205	220	186	212	220	185	21
4	0	2	13	2	10	18	8	17	17	8	18	22	9	2
0	0	3	8	4	14	63	48	62	144	120	125	189	166	16
216	191	193	216	187	192	220	191	200	222	195	202	220	194	19
181	163	187	180	163	186	180	162	182	180	161	175	181	162	17

Fig. 3 : CSV - hmnist_28_28_RGB

They also provide .csv files on the labeling and the pixel sizes of each sample. For figure 2, that CSV contains information on their patients, such as: age, gender, type of skin cancer, and location of the skin cancer. In figure 3, each image's data is stored as a series of numbers, where each number represents the color value (Red, Green, and Blue) for each pixel in the image.

Data Augmentation:

For the data, I applied several preprocessing steps for model training and to improve its generalization ability. First, I resized all the images to a uniform size of 28x28 pixels, ensuring consistency in the input data. Next, I normalized the images by scaling the pixel values between 0 and 1, which helps the model learn more efficiently by reducing the impact of large differences in feature scales. After normalization, I flattened the images from their original 2D structure into 1D vectors, which is necessary for the fully connected layers of the neural network. The reason why I did this is to help the model generalize better and avoid overfitting.

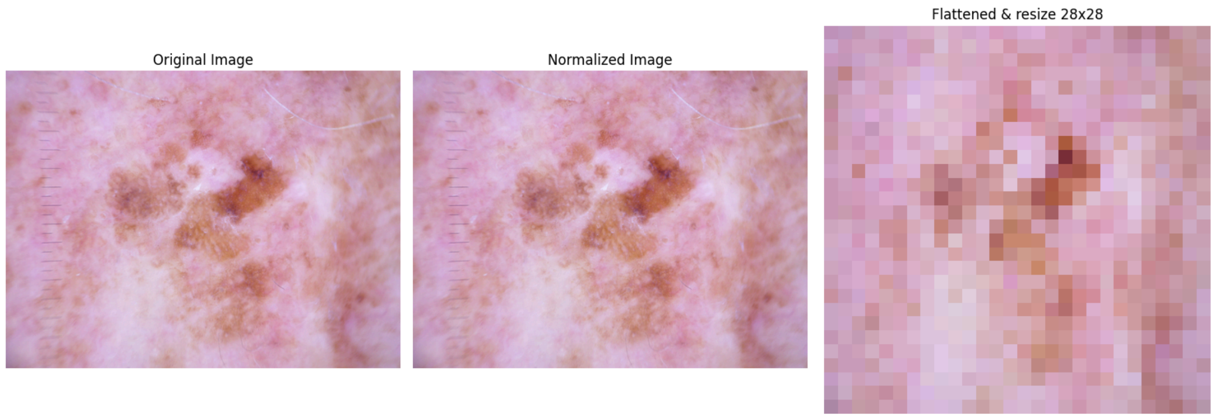


Fig. 0 : Data Augmentation

Methodology:

This project uses the HAM10000 dataset, consisting of 10,015 images of skin lesions classified into 7 different types of skin cancer. The images are prepared for the model by using the hmnist_28_28_RGB.csv file, which contains the pixel values of images that have been resized to 28x28 pixels. These pixel values are used in the model. The images are then normalized by scaling the pixel values to the range of 0 to 1, which helps improve training efficiency and stability.

The model architecture has a fully connected neural network (MLP) designed to classify the images. The model starts with an InputLayer, where the input shape matches the 28x28 pixel images. After flattening the image data into a 1D vector, the model passes the data through a Dense layer with 128 units and ReLU activation. Finally, the model ends with a Dense layer of 7 units, to correspond for the seven class, and a softmax activation function.

I then used the categorical crossentropy loss function, used for multi-class classification, and the Adam optimizer, for efficiency in training neural networks. The model's performance is evaluated using a confusion matrix, which helps to visualize how well the model is distinguishing between different types of skin cancer. And I will visualize eight randomly selected images alongside their true labels and predicted labels in a grid. To visually evaluate how well the model is performing.

Initial Results:

Before running the actual model, I normalize the dataset from figure 4 (right graph) so as to get an even distribution of my data and keep it in the same range. As in figure 4 (left graph), I’m dealing with a negatively skewed graph, which can lead to inaccurate predictions. So to solve that issue, I normalized the dataset to be a bell shape, as it makes the data more balanced, which could help with better predictions.

```
x = (x - np.mean(x)) / np.std(x)
```

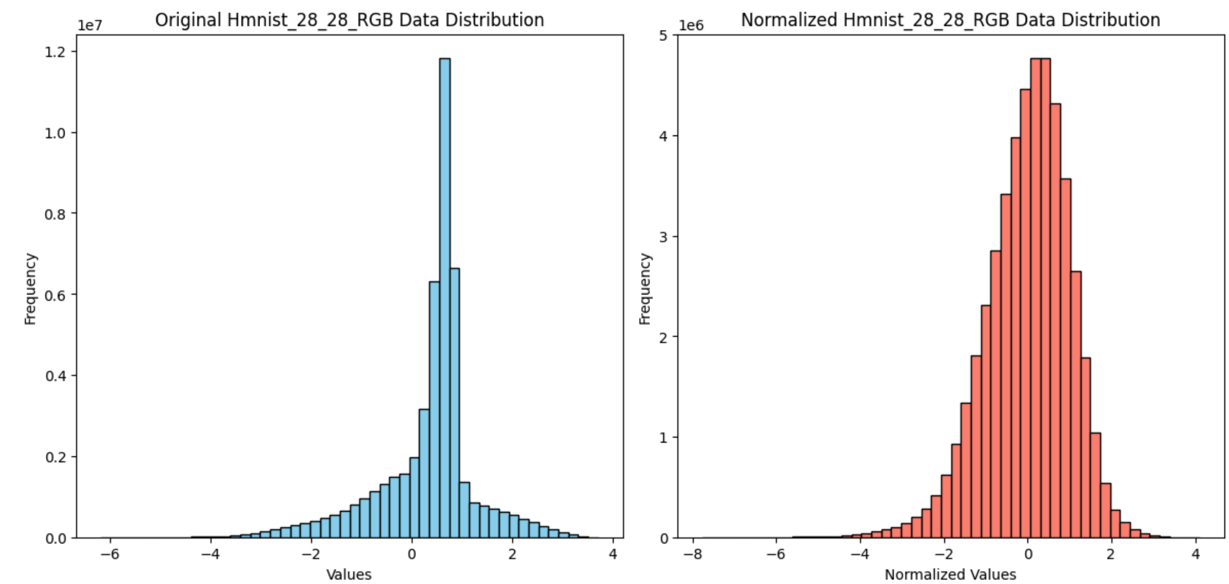


Fig. 4 : Normalized Data for hmnist_28_28_RGB.csv

Afterwards, once I test and train the model with the dataset. I got the model for the accuracy and loss of the model here:

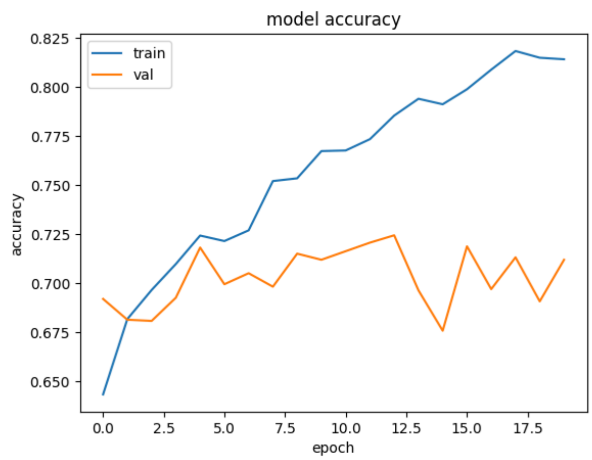


Fig. 4 : Model accuracy - train & validation

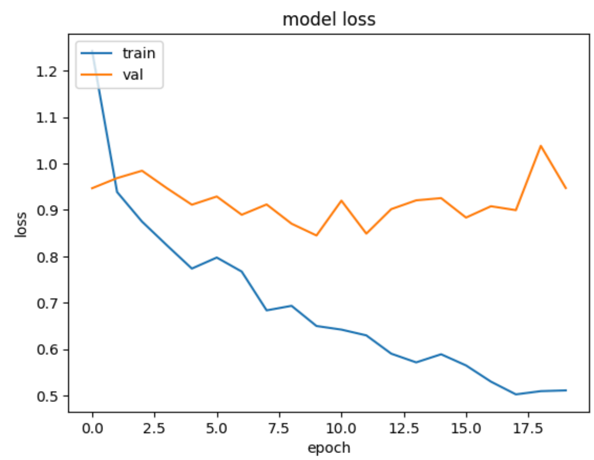


Fig. 5 : Model loss - train train & validation

During the training and validation test, from the model, there was an accuracy of 67.89% and a loss of 90.27%. In figures 6 and 7, it appears that overfitting is occurring because the training accuracy remains significantly higher than the validation accuracy, and the validation loss increases as the model continues training. This indicates that the model is performing well on the training data but is failing to generalize to unseen data.

Confusion matrix:

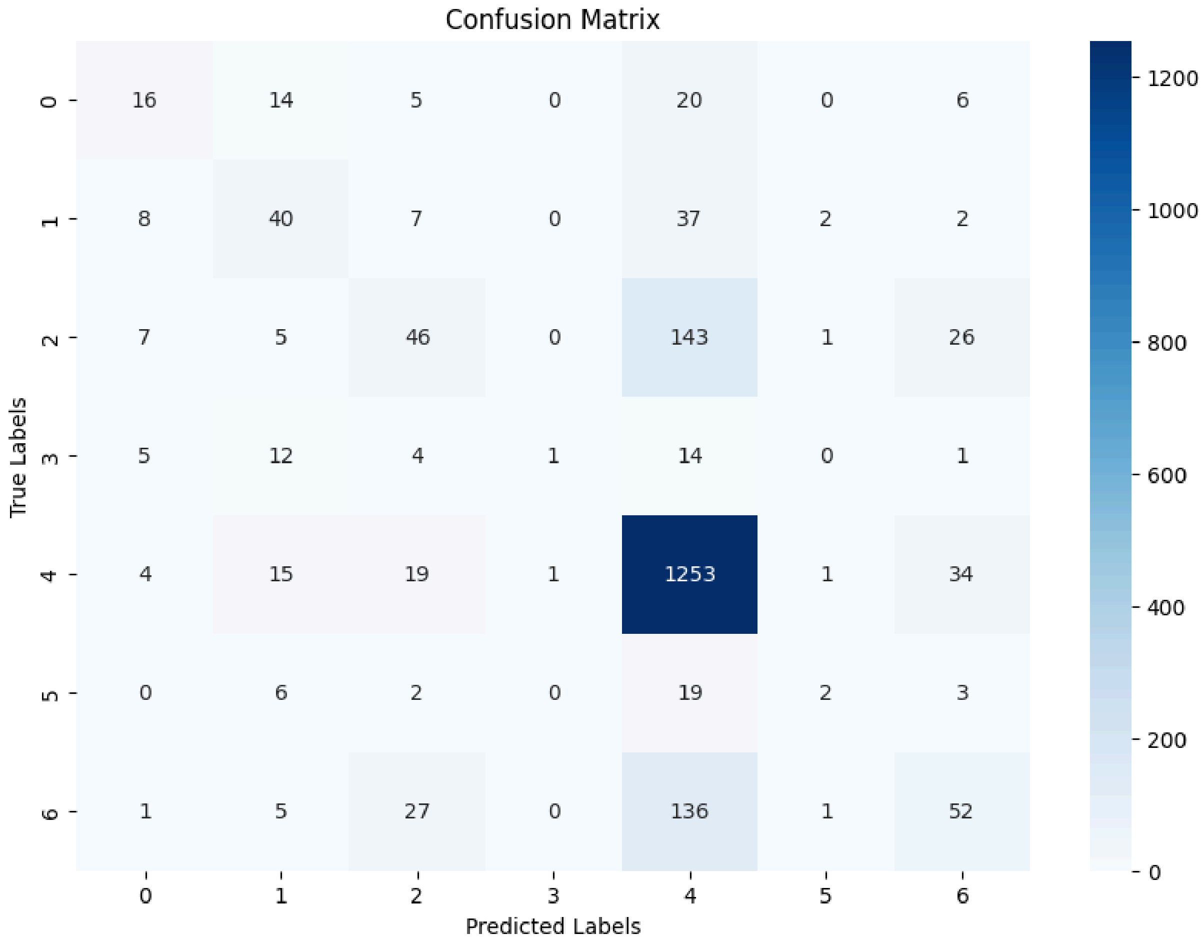


Fig. 8 : Confusion Matrix (Accuracy 67.89%, *Loss 90.27%)*

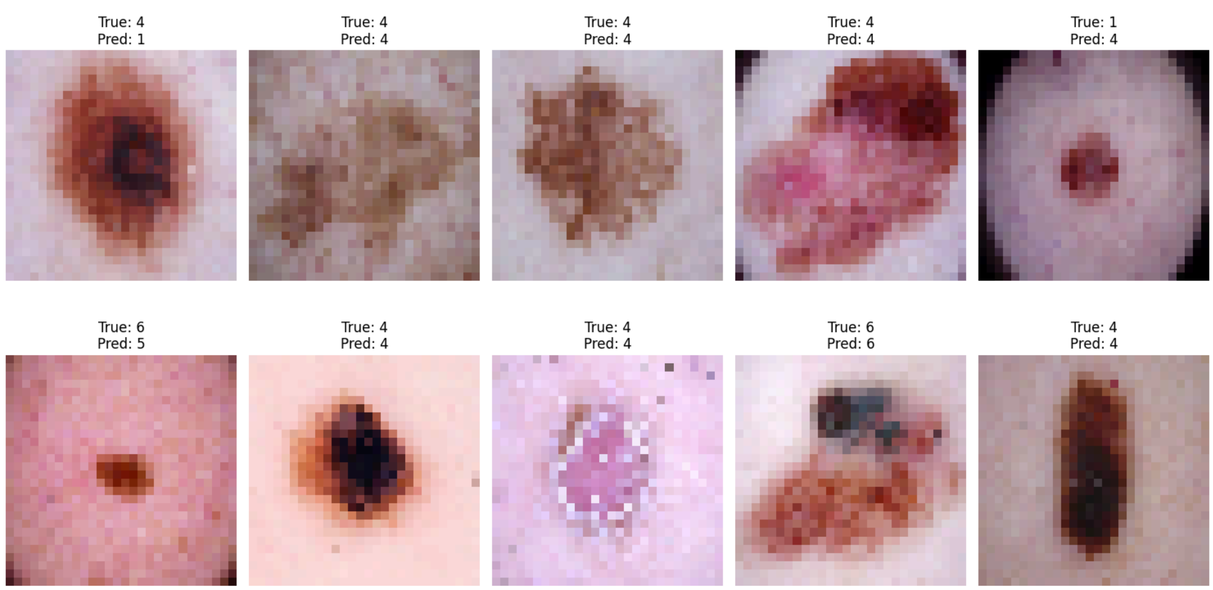


Fig. 9 : Predicted Visualize Results

Labels/Classes:

```
0: ('akiec', 'Actinic keratoses and intraepithelial carcinomae'),
1: ('bcc', 'basal cell carcinoma'),
2: ('bkl', 'benign keratosis-like lesions'),
3: ('df', 'dermatofibroma'),
4: ('nv', 'melanocytic nevi'),
5: ('vasc', 'pyogenic granulomas and hemorrhage'),
6: ('mel', 'melanoma')
```

Fig. 10 : Table of context

The first thing I noticed was that the prediction and true values in figure 9, were often predicted correctly. But for some of the cases when I reran the cell to get a different sample prediction, it would always mistake the sample as 4 (nv), which explains the reason why I got a loss of 90.27%. But when I made a confusion matrix, figure 8, I realized that my dataset is very imbalanced. As almost all of the skin cancer samples are being classified as melanocytic nevi (nv). So to make sure, I was dealing with an imbalancing problem. I used MATLAB and plotted the distribution of my dataset to see if I was indeed dealing with a balancing issue.

Attempts for Improvement (methods used are **bolded**):

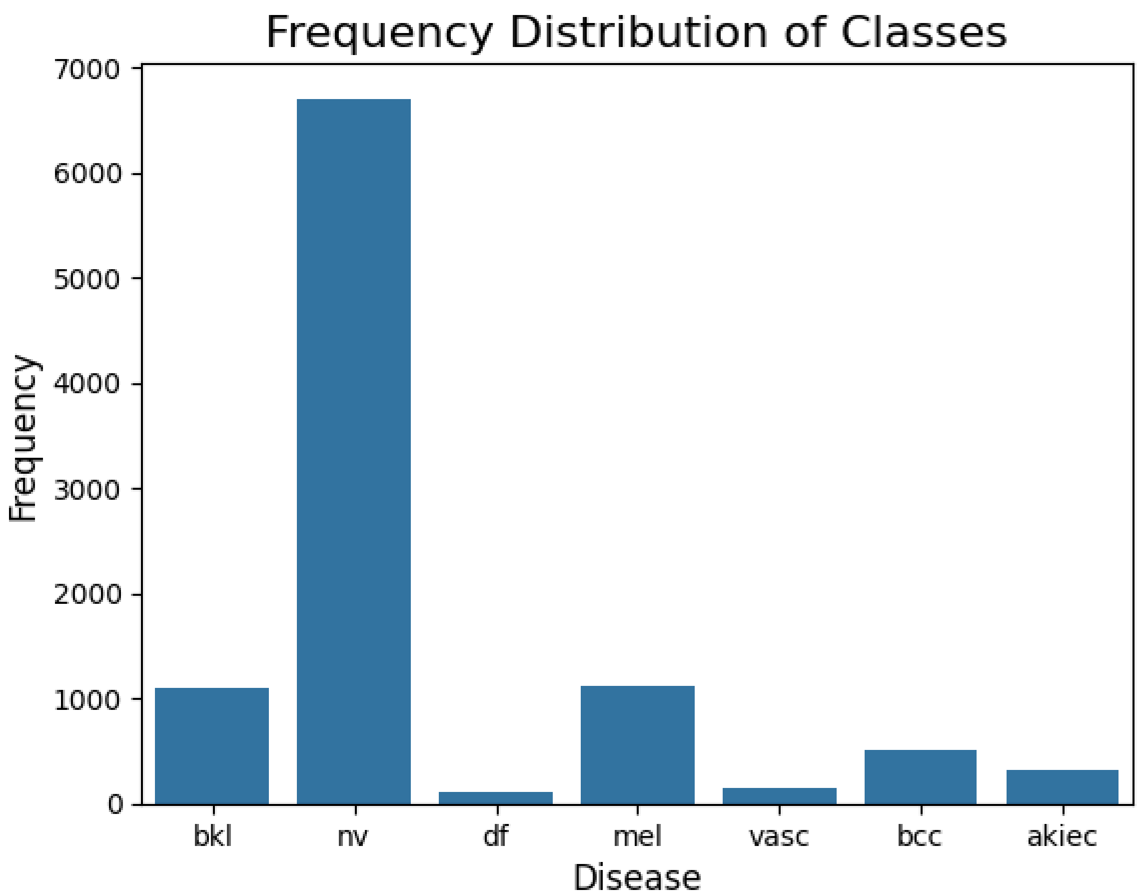


Fig. 10 : Distribution Graph

From looking at the confusion matrix and the distribution graph, it made it pretty obvious I was dealing with an imbalancing issue with my dataset. Because I was getting an accuracy of about 67.89%, I thought my model did well. But because my class distribution is so skewed. As there are more samples of nv then the rest, it explains the reasoning why I was getting 67.89%.

So from my understanding and inference, since the majority of the dataset belongs to the highest class distribution, such as the nv class, the model has a higher chance of accurately predicting the nv class. This is because nv samples make up approximately 75% of the dataset, which I believe explains why I was getting a high accuracy but an even higher loss.

To find a way to address the issue of class imbalance in my dataset, I did some research on techniques that could help. Initially, I was considering using the **class-weight** method like in class, which adjusts the weight of each class during training to make the model more sensitive to the minority class. However, after further consideration, I decided against using this method due to the fact that it didn't improve performance significantly in my tests. Whenever I added weights to the minority class, I would still have issues with overfitting and a very high loss in the 90%.

Instead, I decided to use **SMOTE** (Synthetic Minority Over-sampling Technique), a more advanced method of handling imbalanced datasets, it takes the minority class instances and generates synthetic data samples by interpolation, making the dataset more balanced without needing to collect new data.

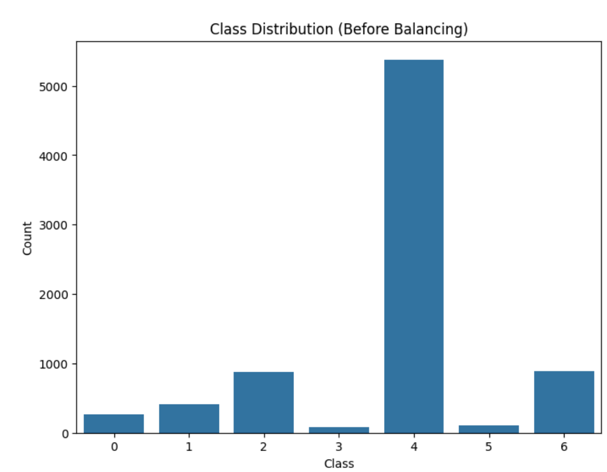


Fig. 12 : Original Class Distribution

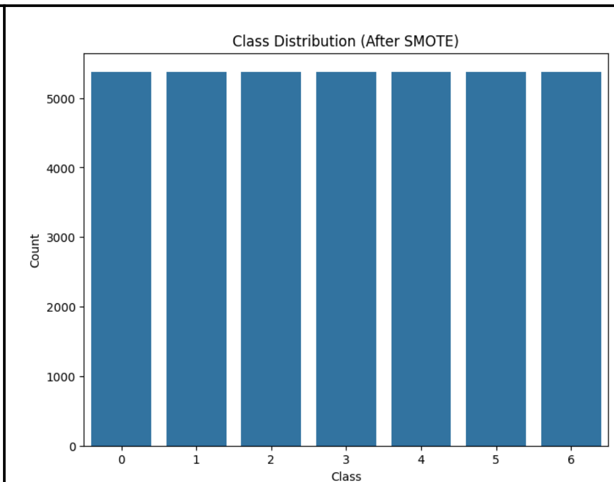
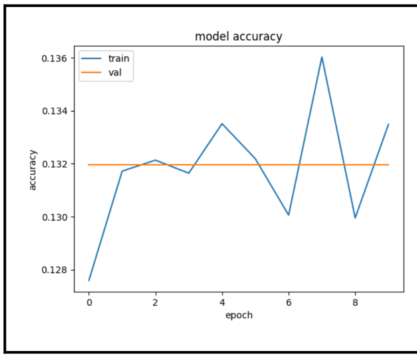
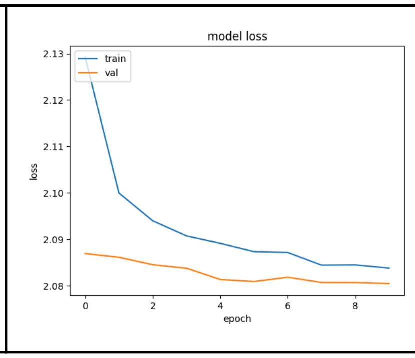


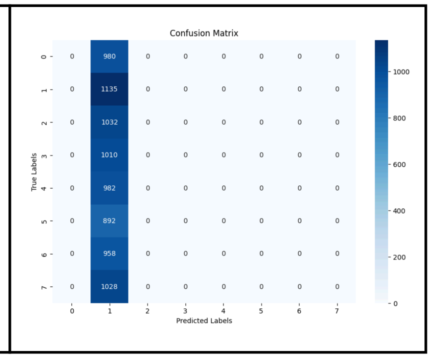
Fig. 13 : Class Dist. after using SMOTE

To address that issue, I had with figure 4 and 5, I tried to reduce the **learning rate to 0.0001** and add a **dropout layer** in my model, with a **0.5 rate**, to help prevent overfitting. As a result there was a slight improvement in both models. As the accuracy became 69.90% and the loss was 90.16%, which isn't the best, but it was a slight improvement in the accuracy, but I was still dealing with a high loss of 90.16%. So to fix this issue with the high loss, I decided to add in the **early stop method** (results in "Final Results" section.)

For the models, I tried to implement another one, a basic **AlexNet model**, which had 5 convolution layers, 3 max-pooling layers, 2 Normalized layers, 2 fully connected layers and 1 SoftMax layer.







So, from the results above. I chose to not use that model for this project, because the accuracy was as low as 15%, which I think a factor to this model not performing well is because of the class imbalancing I was facing. I **re-augmented the data, to 227x227x3 images**, and increased the **epoch to about 30**, but I was still facing a lower accuracy of 12% and a high loss of about 110%.

Finally, the last method, I used was a library: `from imblearn.over_sampling import RandomOverSampler`

I tried using this library to solve the imbalancing issue; the library is used for oversampling the minority class in imbalanced datasets. So I thought this library would work really well for my model, but the only thing that it was able to work well on was only in solving the imbalancing issue I was having. The loss and accuracy models were very bad; there was a lot of overfitting, which I think is because of the library, because it randomly duplicates the samples in different classes, which I think increased the likelihood of overfitting since it just duplicates existing data rather than creating new synthetic examples. Because of that, I choose to stick with SMOTE which creates synthetic examples through interpolation. Note, this method did not improve the results for the AlexNet model also.

Final Results:

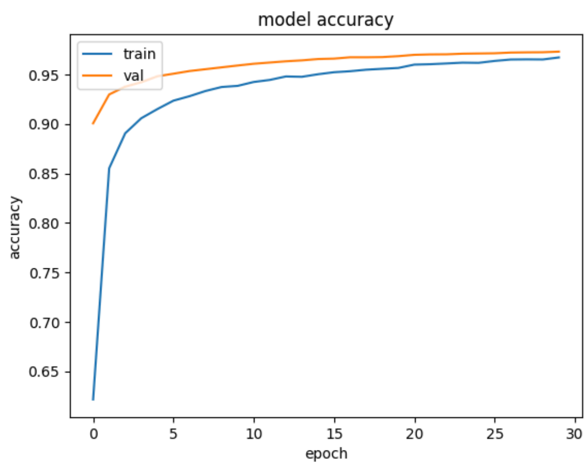


Fig. 14 : Final Model Accuracy

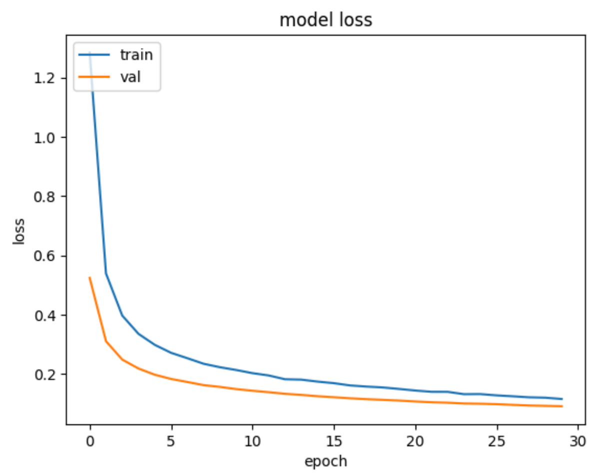


Fig. 15 : Final Model Loss

So for my final results, I stuck with my original model that I described in the methodology. I used SMOTE to solve the issue, where I had an imbalancing issue and for the overfitting issue, I used the early stop method. So when testing with the model, I tested different epochs, ranging from 10-50, but 10 performed much better. As I was getting an accuracy of 80 - 96% and a loss of 10-21 % which was much better than my original result. Notably, the confusion matrix was now predicting the other classes about 80 - 90% of the time from what I inferred in figure 16 and 17.

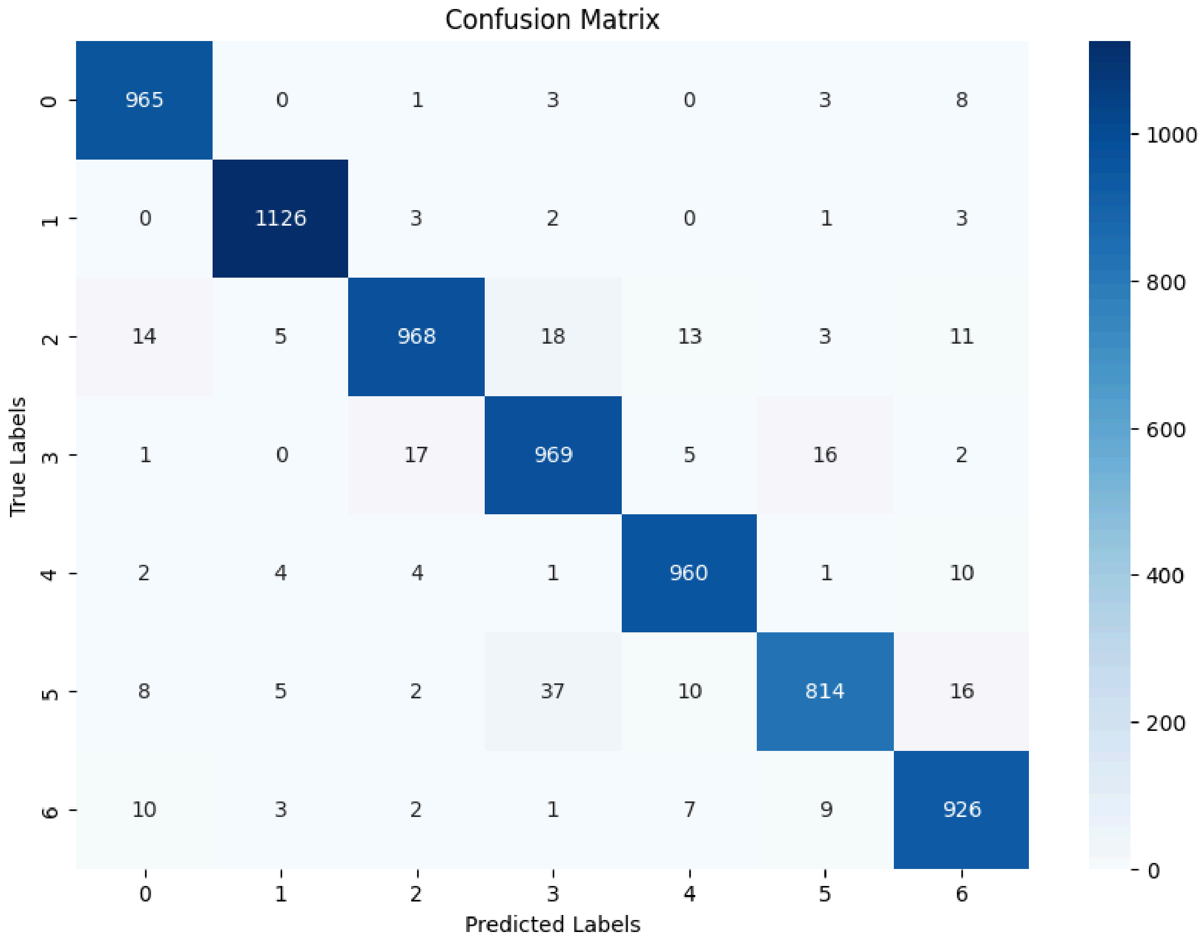
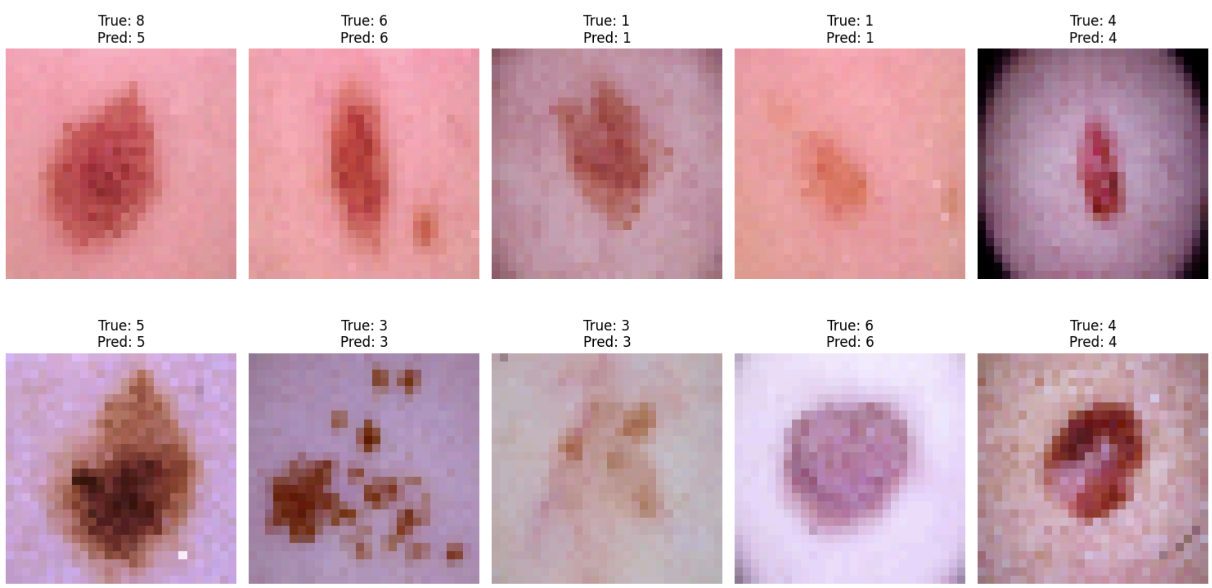


Fig. 16 : Final Confusion Matrix (Accuracy 94.43%, Loss 19.73%)



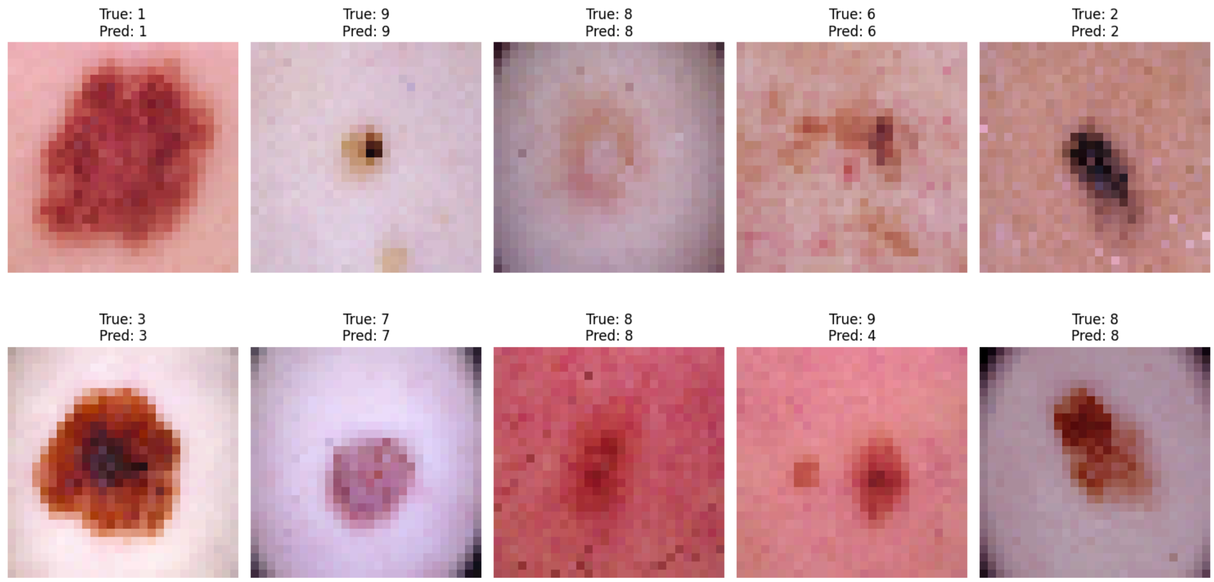


Fig. 17 : Predicted Visualize Results

What Would I Do Differently:

What I would do differently, would be trying to get the AlexNet model to work. I spent too long trying to solve the imbalancing issue and overfitting. I just ran out of time to figure out what was going on with the AlexNet model. So, maybe next time, I would work with this model first, because it's known to work well with big datasets.

Conclusion:

Overall, I learned a lot about how to preprocess, analyze, and work with imbalanced datasets. I learned how crucial it is to properly prepare data for machine learning models by scaling images to a consistent dimension and normalizing datasets. Also, of the difficulties in dealing with deep learning architectures, and how important data balancing is for better model performance.