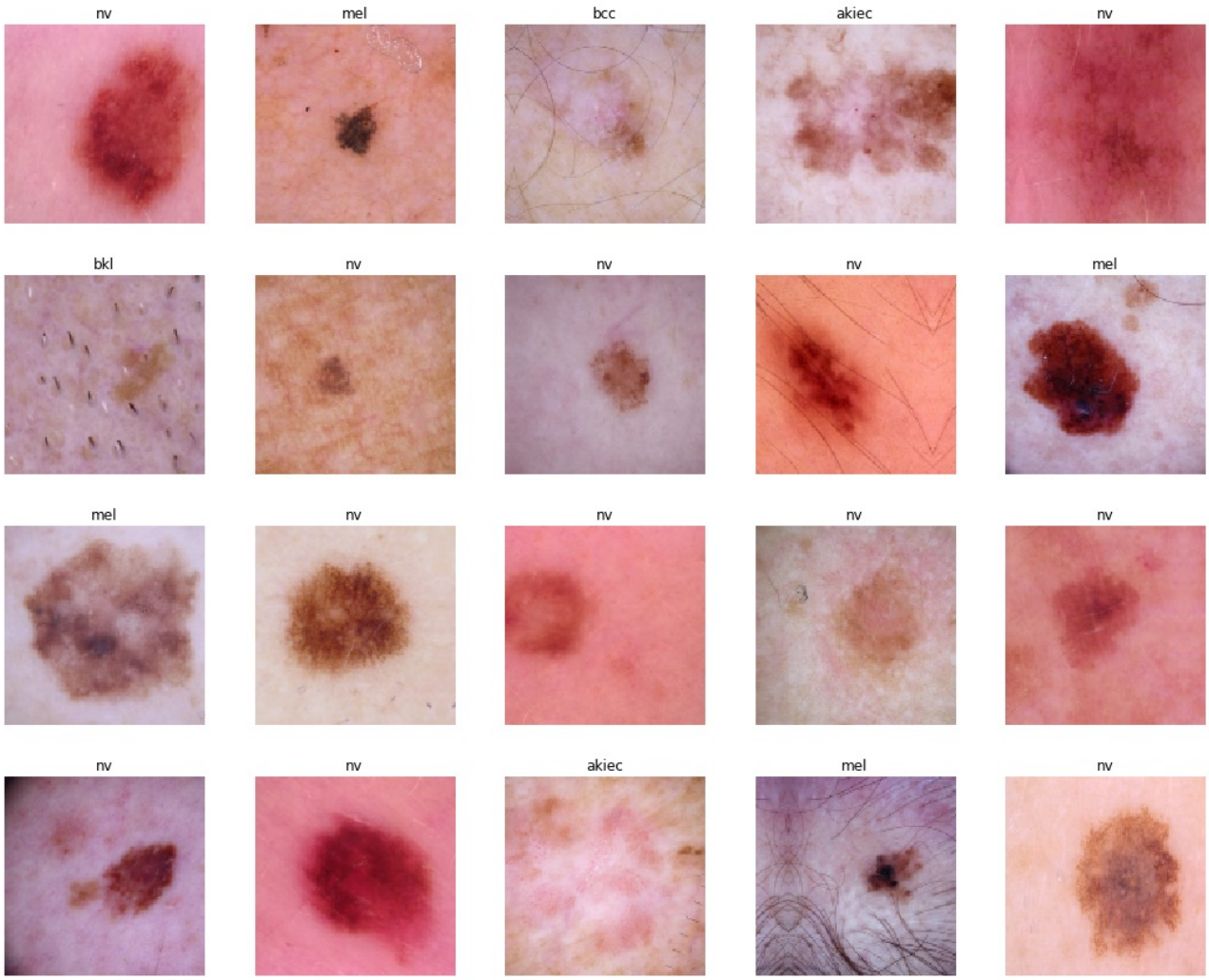
Skin Lesion Classification and Segmentation

Using HAM10000

Yousef Alaa El-Din — Shahd El-Refai - 10 December 2024



Overview

Introduction

Skin cancer is a major health concern worldwide, and catching it early can make all the difference in treatment success. With advancements in technology, especially in machine learning and computer vision, we now have tools to help improve how quickly and accurately skin cancer is diagnosed. This project focuses on using the HAM10000 dataset—a collection of diverse, carefully labeled skin images—to classify different types of skin lesions and pinpoint their exact locations in the images. By automating this process, the hope is to create a system that supports doctors in diagnosing skin cancer more efficiently and accurately.

Objective

The goal of this project is to be able to make:

1. **Classification:** To classify dermatoscopic images into one of seven categories of skin lesions.
2. **Segmentation:** To identify and delineate the region of interest (lesion) in dermatoscopic images to aid in more precise diagnosis and further analysis.

The Dataset

The HAM10000 ("Human Against Machine with 10000 training images") dataset contains 10,015 labeled dermatoscopic images, with seven categories of lesions:

1. Actinic keratoses (AKIEC)
2. Basal cell carcinoma (BCC)
3. Benign keratosis-like lesions (BKL)
4. Dermatofibroma (DF)
5. Melanoma (MEL)
6. Melanocytic nevi (NV)
7. Vascular lesions (VASC)

All the images beloging to all different classes are located in one directory. A csv file named *GroundTruth.csv* specifies the class of which each image belongs to.

image	MEL	NV	BCC	AKIEC	BKL	DF	VASC
ISIC_0024306	0.0	1.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024307	0.0	1.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024308	0.0	1.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024309	0.0	1.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024310	1.0	0.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024311	0.0	1.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024312	0.0	0.0	0.0	0.0	1.0	0.0	0.0
ISIC_0024313	1.0	0.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024314	0.0	1.0	0.0	0.0	0.0	0.0	0.0
ISIC_0024315	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Project Methodology

1. **Data Preprocessing:**
    - Normalization and resizing of images.
    - Data augmentation techniques to handle class imbalance and improve model generalization.
2. **Model Selection:**
    - Classification: ResNet18, ResNet50
    - Segmentation: U-Net
3. **Evaluation Metrics:**
    - Classification: Accuracy, precision, recall, F1-score, and AUC-ROC.
    - Segmentation: Dice coefficient, Intersection over Union (IoU).

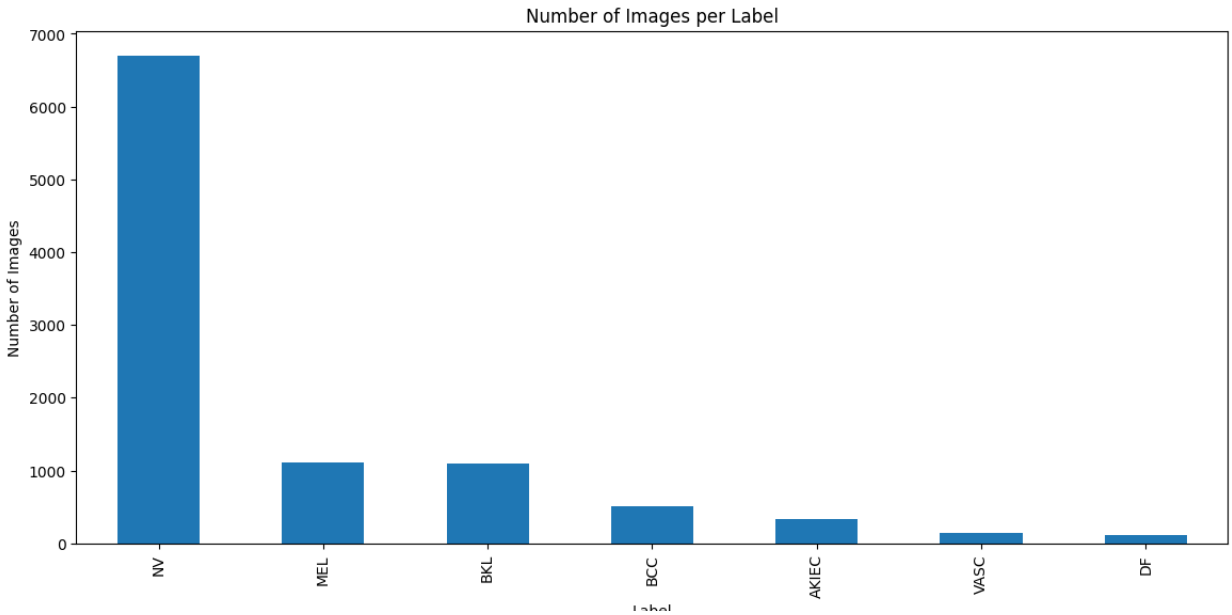
The Dataset Dataframe

The dataframe containg all the dataset is ***skin_df*** starting by reading *GroundTruth.csv* file using *pandas* libray.

We then add to columns:

1. **Label:** the abbreviated class name corresponding to each image.
2. **Path:** the memory path to the corresponding image.

Displaying the class distribution of *skin_df*



Label	Number of Images
NV	6705
MEL	1113
BKL	1099
BCC	514
AKIEC	327
VASC	142
DF	115

There is a huge imbalance in our dataset. We'll address this issue with resampling and during data augmentation. But first, we need to split our dataset to be able to train and test our model.

Splitting the Dataset

We will start by spliting the *skin_df* into:

1. **train_df:** 80% of skin_df.
2. **val_df:** 10% of skin_df.
3. **test_df:** 10% of skin_df.

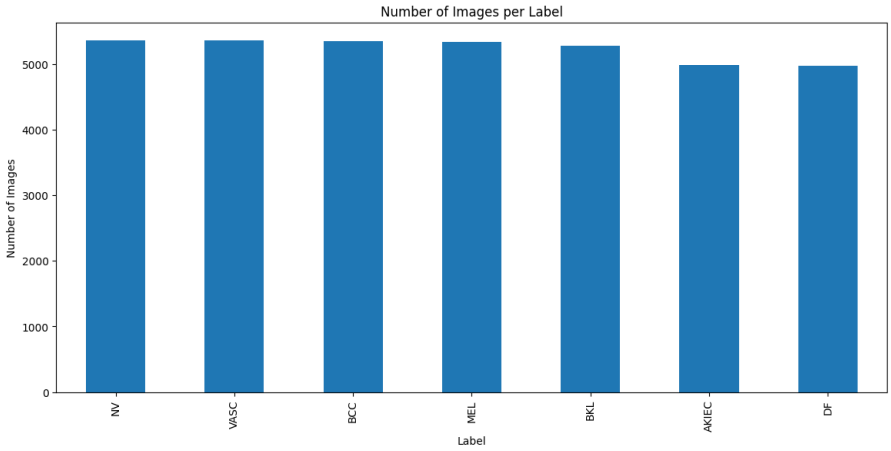
Using ***train_test_split*** from sklearn library, making sure the each class is split fairly.

label	Train	Validation	Test
NV	5364	670	671
MEL	890	111	112
BKL	879	110	110
BCC	411	51	52
AKIEC	262	33	32
VASC	114	14	14
DF	92	12	11

Class Imbalance Issue

An attempt was made to address this issue without directly resampling the classes, unfortunately the model was good predicting some classes but couldn't predict some (sometimes at all).

Resampling the data was done using pandas dataframe ***resample*** method on ***train_df***, providing each class with the factor it needs to be close to the rest.



class	Number of Images
NV	5364
VASC	5358
BCC	5343
MEL	5340
BKL	5274
AKIEC	4978
DF	4968

An attempt to address the class imbalance issue using weighted classes in the loss function. This assigns higher weights to underrepresented classes to ensure they contribute more significantly to the loss during training. Unfortunately, it didn't yield to better performance with resnet18 model, so we discontinued its use.

Normalization and Resizing of Images

In order to train our model the images should be normalized and resized to achieve consistency.

Best practice of normalizing the data is done by calculating the mean and standard deviation of the training dataset. After calculating we found:

- **Mean:** [0.76272242, 0.54567698, 0.56987753]
- **Std:** [0.14188903, 0.1527812, 0.17025911]

Resizing the images to **(224,224)** captures good amount of detail and not too big to hurdle the computation and is compatible with ResNet.

Data Augmentation

As promised, we will address our classes imbalance with data augmentation too. Our approach will be augmenting only the classes that needs augmentation, which are basically all classes except **NV** stored in an array **NO_AUGMENT_CLASSES**.

To achieve this we first initialized the training dataset without applying any transformation. Then looping through each **image, label** in the dataset, we check the label if it's found in **NO_AUGMENT_CLASSES** we dont augment else we augment.

Augmentation transforms include:

- Random Horizontal and Vertical Flips
- Random rotation - 20°
- Color Jitter: brightness=0.1, contrast=0.1, hue=0.1

Applying augmentation on NV class too was also tried, but it yielded worse results (will be provided later on).

Custom Dataset Class

Two issues arrised from our approaches on resampling and data augmentation.

- Resampling issue: Our training dataset increased from 8,012 to 36,625. It's unrealistic to save them in a folder to create a dataset.
- Data augmentation issue: It's a conditioned augmentation, we can't pass the transformation with the dataset anymore.

What we need:

- Access to data through our dataframe.
- Choose which transformation to be applied according to the class of the image.

Solution:
A **CustomImageDataset** class which is child of **Dataset** class found in *torch.utils.data* library and overrides

- **__len__( ) :** returning length of the dataframe
- **__getitem__( ) :** fetches the class and image path both saved in the dataframe, applies the appropriate transformation on the image based on its class.

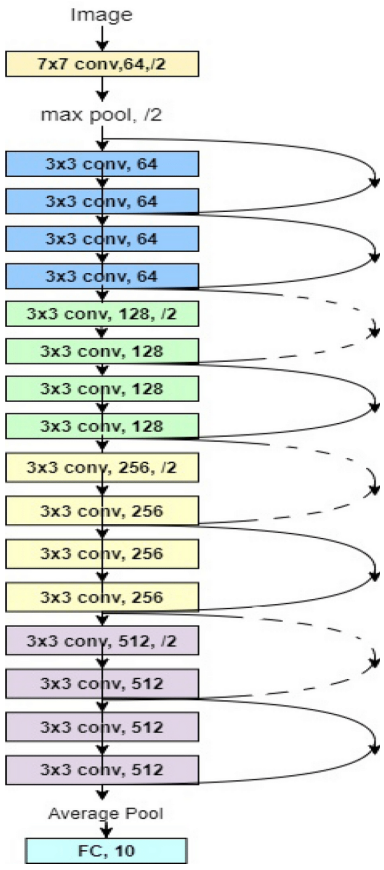
Now using our custom dataset we can easily obtain ***train_dataset***, and use it also to get ***val_dataset*** and ***test_dataset***. Passing these datasets to DataLoader, we obtain train_loader, val_loader and test_loader, which facilitates calling during training and testing by working with batches.

Classification Models

Now on to the interesting but painful part. I never thought training NNs can be this time and energy (for me but specifically my laptop) consuming.

ResNet18

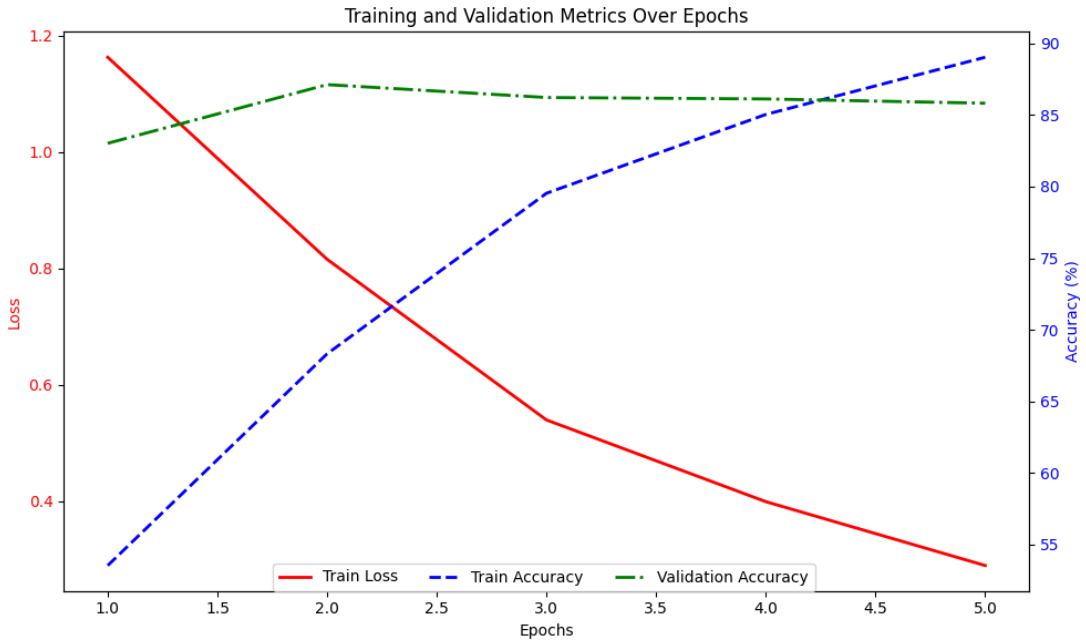
What makes ResNets special? It's their skip connections.NNs become deeper, a vanishing gradient problem arrises. where gradients calculated through backpropagation become very small as they are propagated backward through the layers. This causes the weights in earlier layers to update very slowly, leading to slow or stalled learning.



A a skip connection, or a sequence of skip connections in a deep network, creates a direct path from the deeper parameters to the loss. This makes their contribution to the gradient of the loss more direct, as partial derivatives of the loss with respect to those parameters have a chance not to be multiplied by a long chain of other operations.

Using pytorch resnet18 model, untrained, and altering the fully connected layer to have an output of 7 neurons (input is the same as we resized to 224), batch size of 32, lr=1e-3 with 0.1 factor (min mode on validation loss), Adam optimizer and Cross Entropy Loss.

Total number of model parameters: 11180103

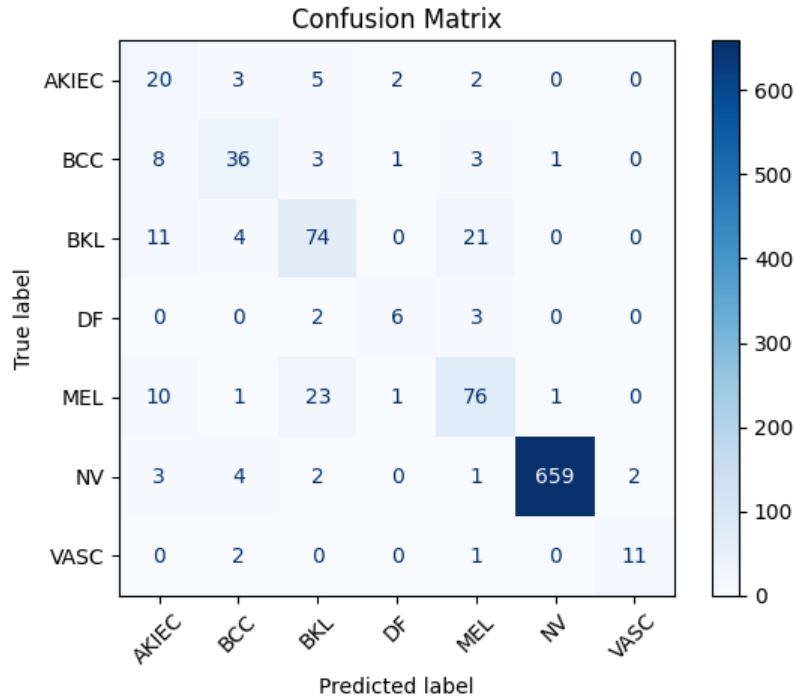


The highest validation score:

```
Epoch [2/10], Train Loss: 0.8156, Train Accuracy: 68.32%,
              Validation Accuracy: 87.11%
```

Then it began decreasing while training accuracy increased, this means it started overfitting.

The model best state was saved and loaded for testing.



```
Total Loss: 0.3779
Overall Accuracy: 88.02%
Precision (macro-avg): 0.71
Recall (macro-avg): 0.71
F1 Score (macro-avg): 0.70

+---------+-------------+----------+------------+-----------+---------+
| Class   |   Precision |   Recall |   F1 Score |   Correct |   Total |
+=========+=============+==========+============+===========+=========+
| AKIEC   |    0.384615 | 0.625    |   0.47619  |        20 |      32 |
+---------+-------------+----------+------------+-----------+---------+
| BCC     |    0.72     | 0.692308 |   0.705882 |        36 |      52 |
+---------+-------------+----------+------------+-----------+---------+
| BKL     |    0.678899 | 0.672727 |   0.675799 |        74 |     110 |
+---------+-------------+----------+------------+-----------+---------+
| DF      |    0.6      | 0.545455 |   0.571429 |         6 |      11 |
+---------+-------------+----------+------------+-----------+---------+
| MEL     |    0.71028  | 0.678571 |   0.694064 |        76 |     112 |
+---------+-------------+----------+------------+-----------+---------+
| NV      |    0.996974 | 0.982116 |   0.989489 |       659 |     671 |
+---------+-------------+----------+------------+-----------+---------+
| VASC    |    0.846154 | 0.785714 |   0.814815 |        11 |      14 |
+---------+-------------+----------+------------+-----------+---------+
```

Because we have a case of imbalanced class,
let's remember once again the classes' unique data.

label	Train	Validation	Test
NV	5364	670	671
MEL	890	111	112
BKL	879	110	110
BCC	411	51	52
AKIEC	262	33	32
VASC	114	14	14
DF	92	12	11

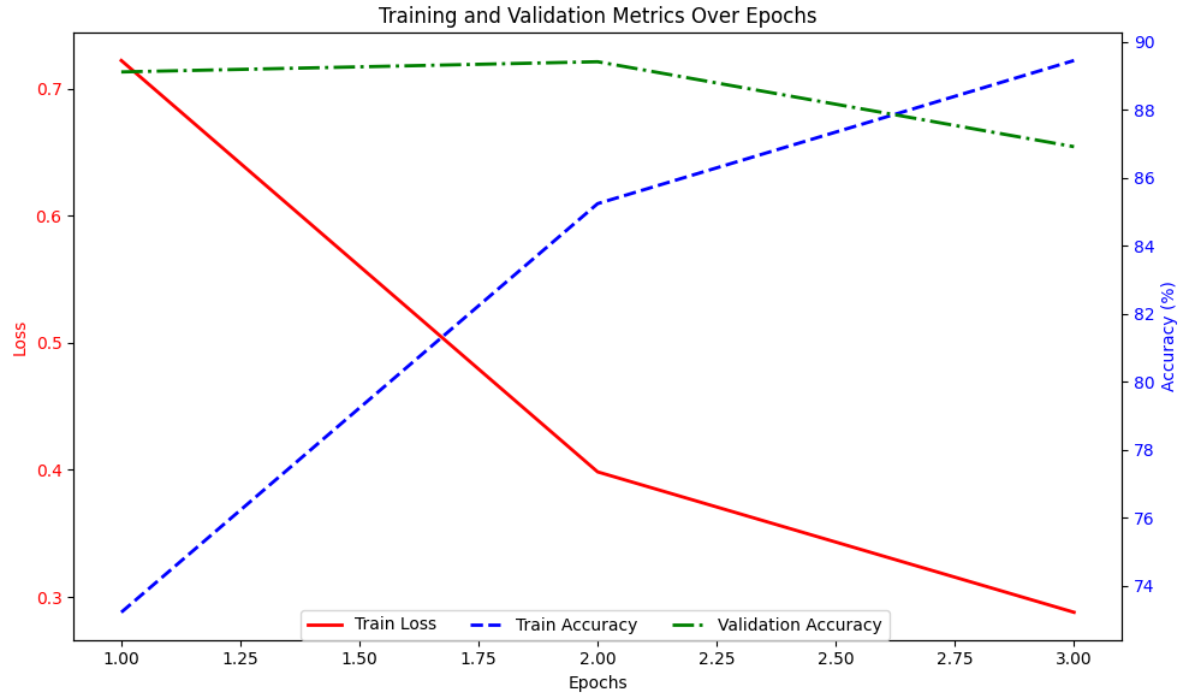
Surprisingly, VASC, being the second least class, it very high scores. While AKIEC, having more data, it scored the least in all of them. DF, being the least class did fairly compared to its challenges.

ResNet50

It has 50 convolution layers compared to ResNet18 having 18 layer. Capturing more features and yielding better results but in a much longer Tim nearly the double.

Same everything as ResNet18 to build the model.

Total number of model parameters: 23522375



The highest validation score:

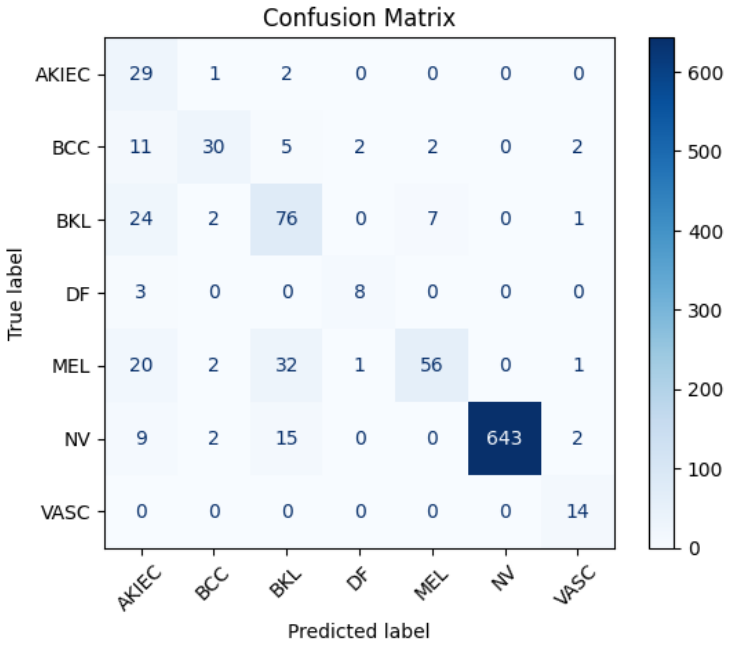
```
Epoch [2/10], Train Loss: 0.3985, Train Accuracy: 85.24%,
              Validation Accuracy: 89.41%
```

Then it began decreasing while training accuracy increased, this means it started overfitting.

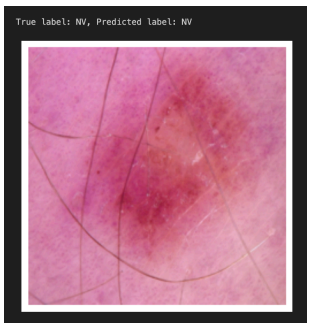
The model best state was saved and loaded for testing.

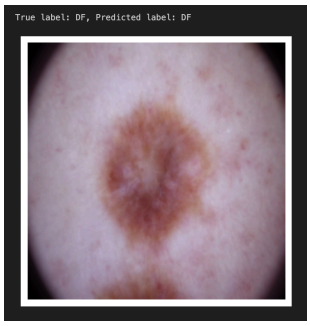
```
Total Loss: 0.4688
Overall Accuracy: 85.43%
Precision (macro-avg): 0.71
Recall (macro-avg): 0.77
F1 Score (macro-avg): 0.70

+---------+-------------+----------+------------+-----------+---------+
| Class   |   Precision |   Recall |   F1 Score |   Correct |   Total |
+=========+=============+==========+============+===========+=========+
| AKIEC   |    0.302083 | 0.90625  |   0.453125 |        29 |      32 |
+---------+-------------+----------+------------+-----------+---------+
| BCC     |    0.810811 | 0.576923 |   0.674157 |        30 |      52 |
+---------+-------------+----------+------------+-----------+---------+
| BKL     |    0.584615 | 0.690909 |   0.633333 |        76 |     110 |
+---------+-------------+----------+------------+-----------+---------+
| DF      |    0.727273 | 0.727273 |   0.727273 |         8 |      11 |
+---------+-------------+----------+------------+-----------+---------+
| MEL     |    0.861538 | 0.5      |   0.632768 |        56 |     112 |
+---------+-------------+----------+------------+-----------+---------+
| NV      |    1        | 0.958271 |   0.978691 |       643 |     671 |
+---------+-------------+----------+------------+-----------+---------+
| VASC    |    0.7      | 1        |   0.823529 |        14 |      14 |
+---------+-------------+----------+------------+-----------+---------+
```

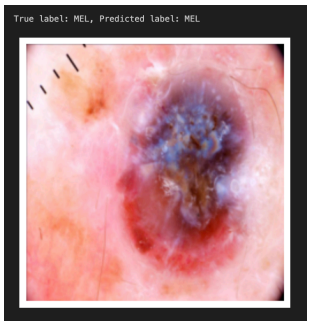
DF did much better than before, AKIEK improved too, same with VASC. Unfortunately the rest gotten worse. The model was good at learning features, many of them, so it improved the performance with classes of lower data, while it overfit with classes that had a lot of data.



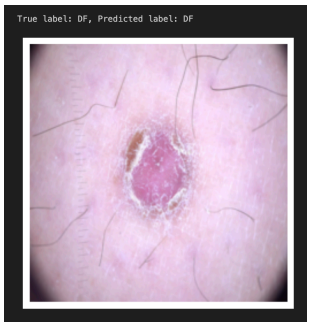
ResNet18 correct predictions

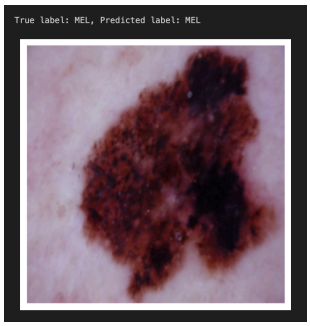


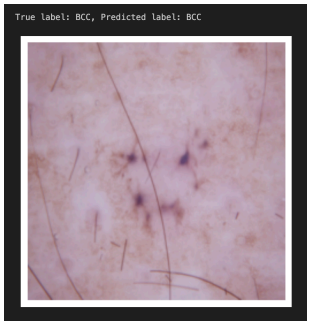




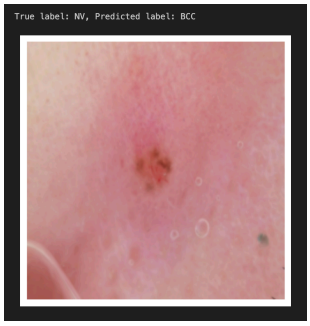
ResNet50 correct predictions

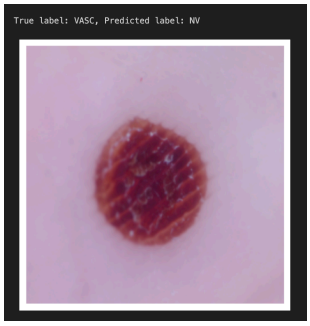


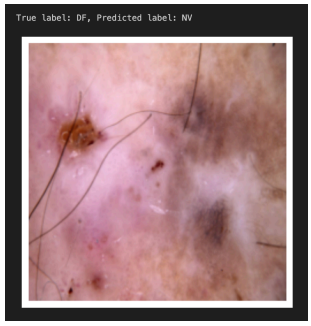




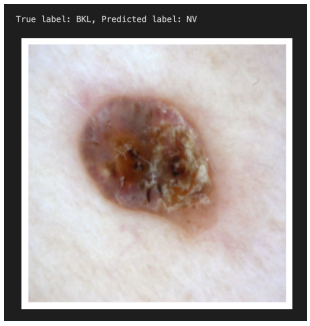
ResNet18 wrong predictions

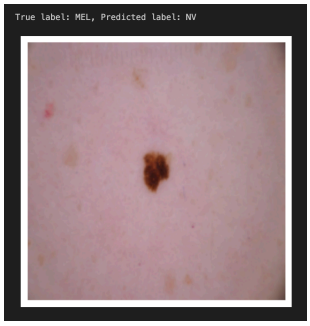


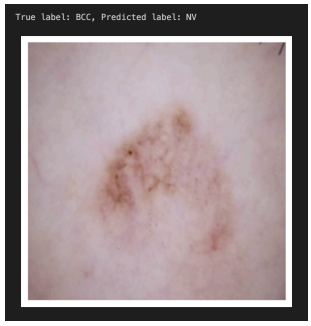


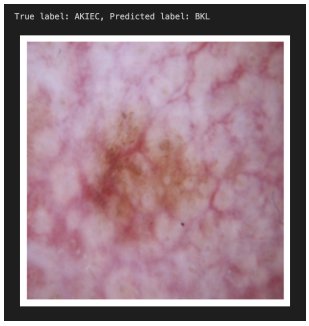


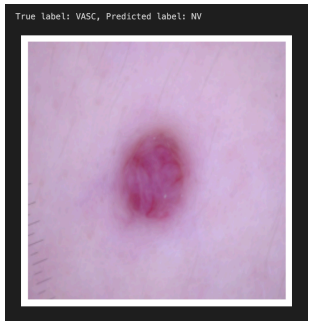
ResNet50 wrong predictions





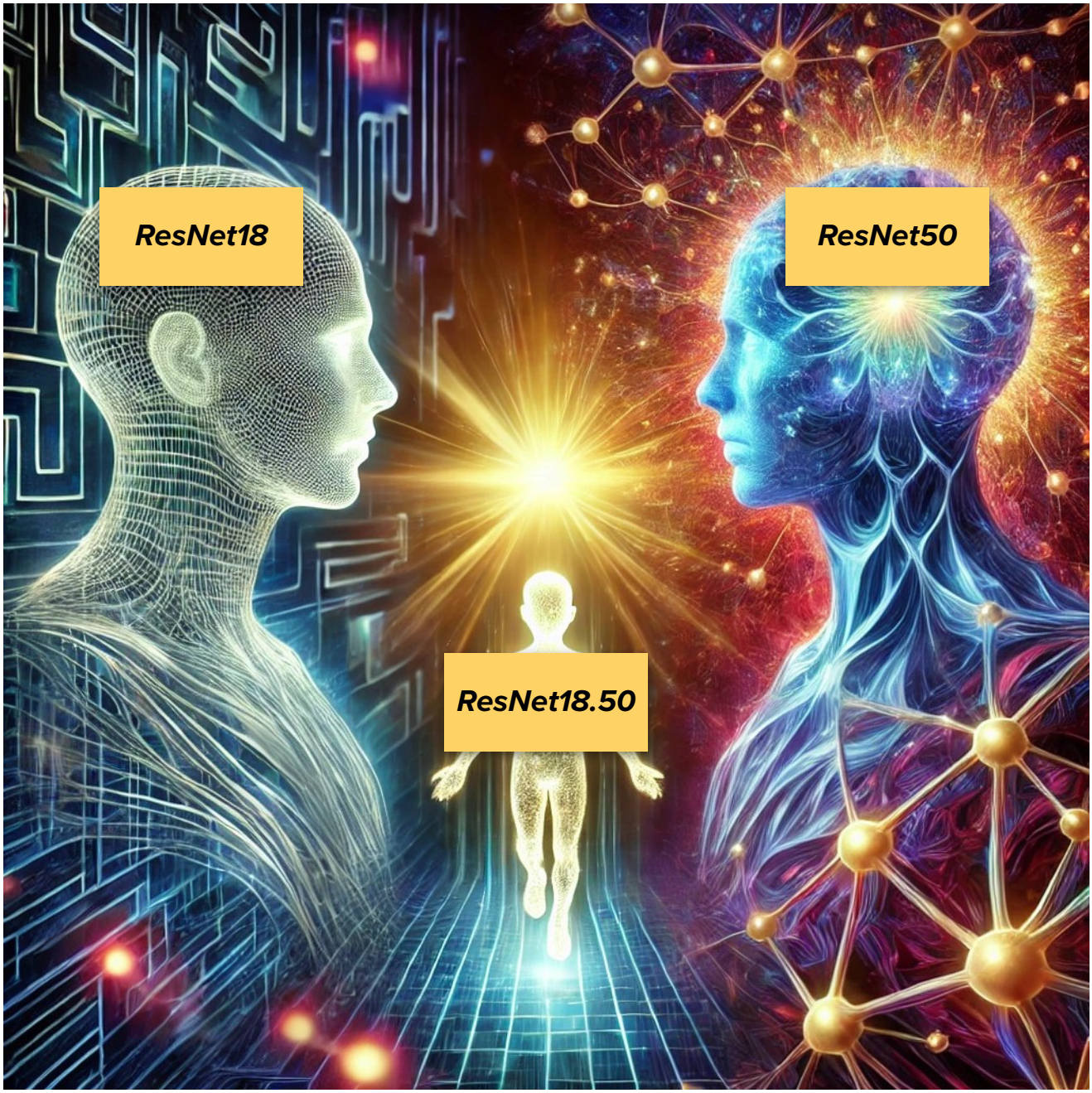






Classification Conclusion

A model that mixes the ResNet18 on large data classes and ResNet50 on lower data classes is needed.



Segmentation Model

U-Net

The U-Net model is chosen for being good in segmentation task which will be explained in this section.

How the Unet model works:

The U-Net model is a fully convolutional neural network designed for image segmentation tasks, particularly in biomedical applications. It has a U-shaped architecture with two main paths: an encoder (contracting path) and a decoder (expanding path). The encoder reduces the spatial dimensions of the input image while extracting features through successive convolutional layers with ReLU activation and max pooling. This compresses the image into a lower-resolution representation rich in features. The decoder then gradually up samples this representation using transposed convolutions and combines it with high-resolution features from the encoder through skip connections. These skip connections preserve spatial details and improve segmentation accuracy by bridging the encoder and decoder layers. The model concludes with a final convolutional layer that outputs a segmentation mask. U-Net's design balances feature extraction and localization, making it highly effective for tasks requiring precise segmentation while addressing issues of redundancy and computational inefficiency found in earlier methods.

Implementation:

- **DoubleConv Module:** Performs two convolutional layers with batch normalization and ReLU activation.
- **DownBlock:** Combines DoubleConv with MaxPooling for downsampling.
- **UpBlock:** Uses transposed convolutions or bilinear upsampling for upsampling.
- **Final Convolution:** Outputs a single-channel mask.
- **Forward Pass:** Downsampling is performed to extract features, followed by upsampling with skip connections.

Model Training and Validation

The training loop involves backpropagation and weight updates using Adam optimizer.

Implementation:

- **Loss Function:** Binary Cross-Entropy Loss with logits is used.
- **Training:** The model is trained for multiple epochs with loss tracking.
- **Validation:** At the end of each epoch, the model is evaluated on the validation set.
- **Dice Coefficient and IoU:** Calculated for comparing predictions against ground truth masks.
- **Visualization:** Random samples of input images, ground truth masks, and predicted masks are visualized

Dice Coefficient:

The Dice Coefficient measures the overlap between the predicted mask and the ground truth mask. It is defined as:

$$\text{Dice Coefficient} = 2 \times |A \cap B| / |A| + |B|$$

Where:

- A is the set of pixels in the predicted mask.
- B is the set of pixels in the ground truth mask.
- $|A \cap B|$ is the number of overlapping pixels between the predicted and ground truth masks.

The Dice Coefficient ranges from 0 to 1, where 1 indicates perfect overlap and 0 indicates no overlap. It is particularly sensitive to imbalances in the size of the overlapping and non-overlapping areas, making it a common choice for segmentation tasks.

IOU coeff: (intersection over union)

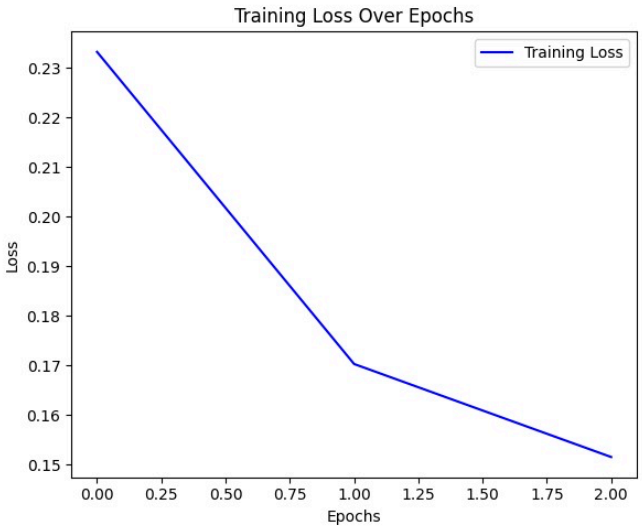
IoU, also known as the Jaccard Index, measures the ratio of the intersection to the union of the predicted and ground truth masks. It is defined as:

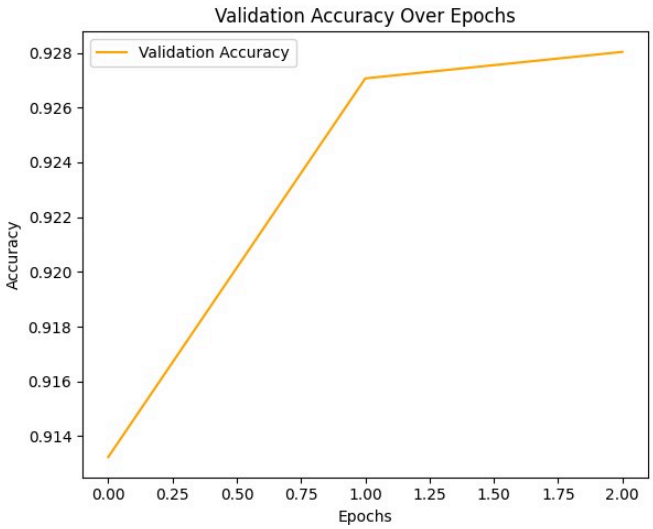
$$\text{IoU} = |A \cap B| \ / \ |A \cup B|$$

Where:

- $|A \cap B|$ is the area of overlap between the predicted and ground truth masks.
- $|A \cup B|$ is the total area covered by both masks.

IoU also ranges from 0 to 1, with 1 indicating perfect alignment. Unlike the Dice Coefficient, IoU penalizes predictions that have large areas outside the ground truth region.





Dice Coefficient: 0.8581, IoU: 0.7515