Bilkent University

CS-464 Introduction to Machine Learning

Project Report

Skin Cancer Prediction
Using Deep Learning Algorithms

Instructor: Ayşegül Dündar Boral

Group Members:

Mehmet Bayık – 21802166 – EE – mehmet.bayik@ug.bilkent.edu.tr
Bekir Burak Çelik – 21901642 – EE – bekir.celik@ug.bilkent.edu.tr
Bilgehan Yılmaz Sandıkcı – 21902354 – CS – yilmaz.sandikci@ug.bilkent.edu.tr
Piotr Wojslawski – 22301137 – CS – pw79871@student.sgh.waw.pl
Fadilah Zakaria – 22301091 – CTIS – fadilah.zakaria@ug.bilkent.edu.tr

December 25, 2023

Introduction

Skin cancer is a major global public health concern that requires immediate attention, especially for early detection and treatment. This is particularly valid for aggressive types such as melanoma. The manuscript details the development of our project, which aims to detect skin cancer using deep learning techniques. This field has enormous promise for advancements in medical imaging and diagnostics. Our research examines various types of skin cancer, including Basal Cell Carcinoma (BCC), Actinic keratoses and intraepithelial carcinoma (AKIEC), and Melanoma. Melanoma is extremely aggressive and the primary cause of deaths from skin cancer. The most prevalent type, AKIEC is a little less common than AKIEC, but if left untreated, it has a greater chance of spreading. Slightly more common, BCC, is less likely to spread and advances slowly, but if left untreated, it can seriously harm the skin.

The core of our approach is the use of advanced deep learning models to accurately classify and predict these types of cancer. We primarily utilize Convolutional Neural Networks (CNNs), essential in image recognition, to process pixel data and distinguish between cancerous and non-cancerous lesions. We also incorporate Residual Neural Networks (ResNet), which include skip connections that help in training deeper network structures by addressing the vanishing gradient problem. This architecture improves the model's ability to recognize complex patterns in dermatological images. Additionally, our project uses transfer learning, specifically using the VGG-19 model pre-trained on extensive datasets. This approach allows our model to benefit from existing knowledge from similar tasks, enhancing its performance, especially in scenarios with limited training data. Our project also explores a hybrid method combining CNNs with eXtreme Gradient Boosting (XGBoost). This approach uses CNNs for feature extraction and XGBoost for classification, blending the strengths of both deep learning and ensemble learning methods.

The report also covers our dataset, which includes numerous high-resolution images of skin lesions, and details preprocessing steps like image augmentation and normalization. We discuss the use of dimensionality reduction techniques, such as Principal Component Analysis (PCA), to improve computational efficiency.

Dataset Description and Preprocessing

The dataset we are using is the HAM10000 dataset provided by Harward, and it has another version in the Harvard Dataverse website containing both a training set and a test set. The dataset is also uploaded to kaggle but it was not containing the test set thus we decided to use the updated version in the Harward website [2]. The original dataset contains 10015 training images and there are 327 ‘akiec’, 514 ‘bcc’, 1099 ‘bkl’, 115 ‘df’, 1113 ‘mel’, 6705 ‘nv’, and 142 ‘vasc’ images according to different categories. The test set has 1511 images and there are 43 ‘akiec’, 93 ‘bcc’, 217 ‘bkl’, 44 ‘df’, 171 ‘mel’, 909 ‘nv’, and 35 ‘vasc’ images according to categories.

The categories ‘df’ and ‘vasc’ contain the smallest number of images in our dataset, which could be contributing to a decrease in accuracy when they are augmented. Also, these types of skin lesions are the less important lesions that are not causing any harm. Thus, detecting those lesions might not be crucial in terms of health application as well. We decided to remove these categories

entirely from both training and test dataset. Now, we will only consider five main classes for our final evaluation.

We first separated the unique images in our dataset into two groups: a training set and a validation set, using stratified sampling with a 4:1 ratio. Then, we enriched the training set by adding all duplicate images from the original dataset. This process resulted in a training set with 8675 images and a validation set consisting of 1083 images.

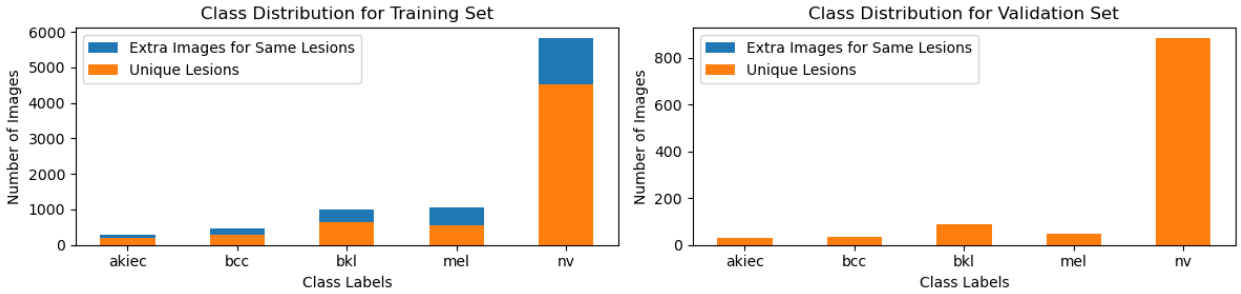

Figure 1: Class Distribution of Training and Validation Sets Before Augmentation

Since our dataset is highly imbalanced, we applied data augmentation to balance the different classes. First, we applied different data augmentation techniques such as horizontal flip, brightness change, contrast change, rotations and gaussian blur. And since there are a lot of original images from the category nv, we dropped the extra images and limited it. To correctly evaluate our models, the data augmentation applied after splitting the data to training and validation sets. This is done because if augmentation is applied before splitting, it might cause the augmented images to appear both in the training and validation sets and the model might be giving wrong accuracy results. The following diagram shows the augmented dataset distribution.

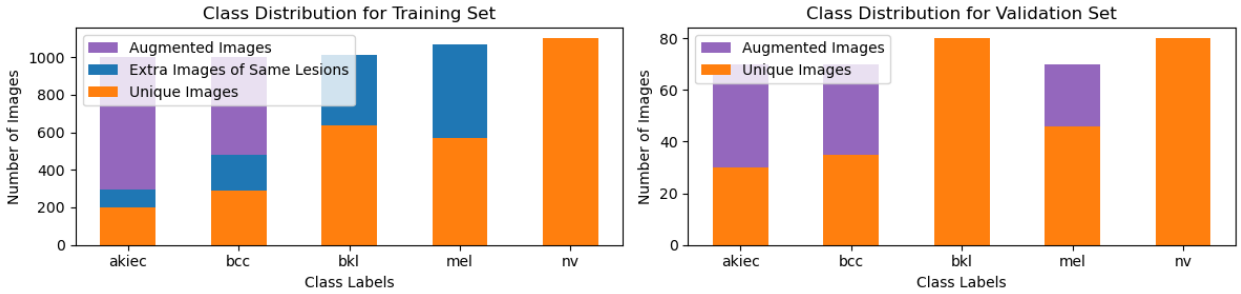

Figure 2: Class Distribution of Training and Validation Sets After Augmentation

For the training dataset, we limited classes like 'nv' with over 1100 images to exactly 1100. This was done by first removing duplicate images and then randomly selecting unique images to reach the 1100 limit. Classes ‘bkl’ and ‘mel’, which had between 1000 and 1100 images, were left unchanged. However, for classes ‘akiec’ and ‘bcc’, which had fewer than 1000 images, we

increased their count to 1000. This increase was achieved by applying random augmentation to the unique images in these classes.
In the validation dataset, classes ‘nv’ and ‘bkl’ with more than 80 images were reduced to 80 in a similar manner, by randomly choosing images. For classes ‘akiec’, ‘bcc’, and ‘mel’, which had fewer than 70 images, we increased their number to 70 using random augmentation on images.
After splitting, augmentation and data cropping from nv category images, the final distribution of the training dataset included 1000 images each from ‘akiec’ and ‘bcc’, 1011 from ‘bkl’, 1067 from ‘mel’, and 1100 from ‘nv’. The validation dataset ended up with 70 images each from ‘akiec’, ‘bcc’, and ‘mel’, and 80 images each from ‘bkl’ and ‘nv’.

Principal Component Analysis

In order to increase the computational speed, we used principal component analysis (PCA), a statistical method for reducing dimensionality, which is very useful for handling high-dimensional data, such as the photos we’re using to classify skin cancer. Fundamentally, PCA converts the original data into a new collection of orthogonal variables called principal components, making sure that the first few preserve most of the variance found in the original dataset.
PCA’s main benefit is its ability to decrease machine learning models’ computational complexity without sacrificing a significant amount of information. We can reduce the number of variables we need to process by eliminating lower-order components that contribute less to data variance by representing the original data in terms of these principal components. This decrease is particularly helpful in the processing of images, as the large number of features can result in significant computational demands.
Using PCA on our skin cancer classification project will make our data less dimensional, which will cut down on the amount of time needed to train our ResNet model computationally. However, using PCA might reduce the accuracy of our model because skin lesions can have tiny variances and by applying PCA these differences might become unrecognizable. We are going to train our model both with PCA applied images and original images and compare the results to determine whether it is useful to use PCA on image classification tasks.
The figure below demonstrates the PCA process. As you can see the PCA result looks like the original image. Since PCA removes the lower-order components the images do not lose a lot of information.

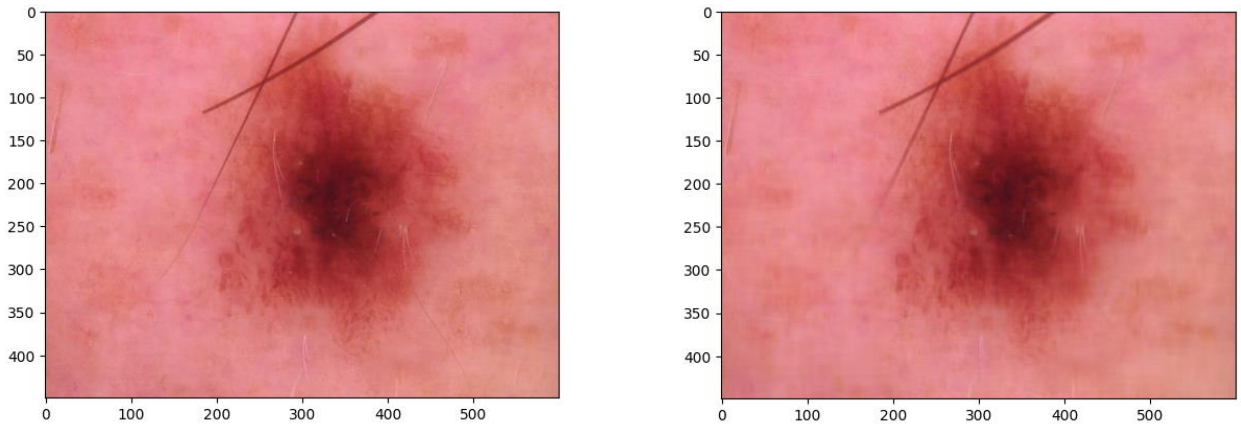

Figure 3: Original Image (Left) and PCA Applied Image (Right)

Description of Selected Algorithms

Convolution Neural Networks (CNNs)

We are using Convolutional Neural Networks (CNNs) as they work well with a visual dataset, such as images. CNNs mainly contain three types of layers. Firstly, convolutional layer followed by pooling layer and lastly, fully connected (FC) layer. With each layer, the CNN increases in its complexity identifying greater portions of the image until it identifies the intended object [3].
At its core, it has a convolutional layer, which contains a series of filters which are matrices that are used on a subset of the input pixel values, the same size as the kernel. Each pixel is multiplied by the corresponding value in the kernel, then the result is summed up for a single value for simplicity representing a grid cell, like a pixel, in the output channel/feature map. The kernel strides over the input matrix of numbers moving horizontally column by column, sliding/scanning over the first rows in the matrix containing the image's pixel values. Then the kernel strides down vertically to subsequent rows. This can be seen in Fig 4[4].

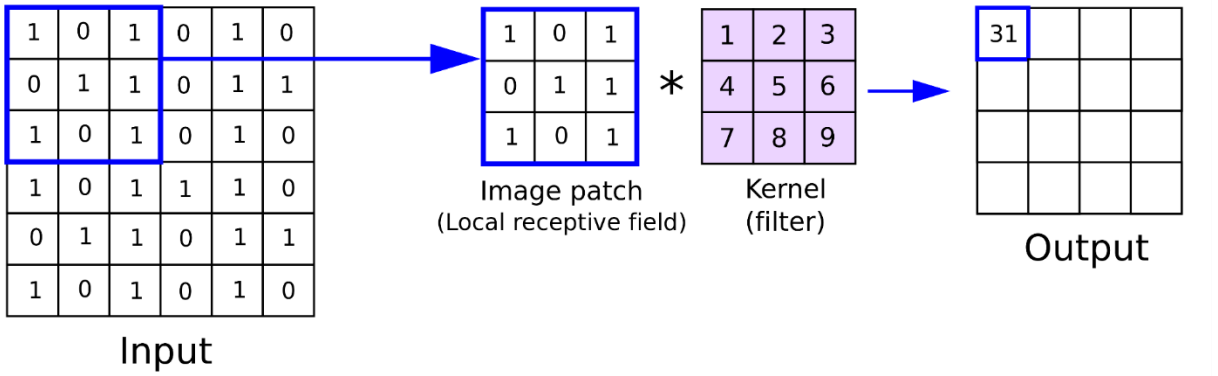

Figure 4: Convolutional Layer of CNNs

The pooling layer aids in reducing computational load, memory usage and the number of parameters in the network. There are no trainable weights in this layer as it is an empty sliding window. Generally, max, or mean pooling is used.
Lastly, in the FC layer, each node in the output layer connects directly to a node in the previous layer [3]. Generally, softmax activation function is used in FC layers to classify inputs producing a probability from 0 to 1.

Residual Networks (ResNet):
The results of our CNN model were compared with our Residual Network (ResNet) model. ResNets, an innovative development in the field of deep learning, specifically in the context of CNNs, was presented by Kaiming He in 2015. They were developed to address the challenge of training extremely deep networks. Adding extra layers to deep networks often resulted in the vanishing gradient problem, which made the gradient too small for the previous layers to learn effectively. ResNets solved this problem by introducing Residual Connections.

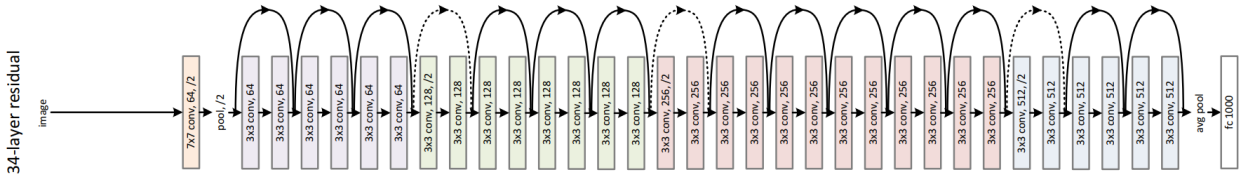

Figure 5: ResNet34 Architecture

The architectural design of ResNets, which includes *skip connections* or *shortcuts* that enable the addition of input to a layer's output at a deeper layer in the network, is one of its primary characteristics. Because of this design, gradients can propagate directly between the layers, making it easier to train deeper networks.
ResNet models are available at different depths; the numbers represent the number of layers in the network. Common models are ResNet-34, ResNet-50, and ResNet-101. ResNets have demonstrated amazing efficacy for image classification tasks, such as the skin cancer classification we are working on. Their proficiency in feature extraction is essential for distinguishing tiny variations in skin lesions related to various forms of skin cancer.
We used ResNets for our project's classification task. We also took advantage of transfer learning from large, diverse datasets such as VGG-19 by using a pre-trained ResNet model. Transfer learning is a strategy that was especially useful for medical imaging applications when there may not have been a lot of data available. To precisely classify the seven forms of skin cancer in our dataset, we fine-tuned the ResNet model according to F1-scores we obtained during our tests. We expected that our model would be able to capture the complex patterns and different characteristics of various skin cancer types, thus aiding in accurate classification.

Coding environment summary

To implement our algorithms, we used a deep learning library named keras from tensorflow. Since our data labels are given as csv files, we first extracted this data. Splitted for training and validation sets and applied some preprocessing. The test set also had a separate csv file. We shuffled these csv files and created image data using flow_from_dataframe() attribute of ImageDataGenerator class from Keras library. To get reproducible results, we used the same random seed and random state values while splitting our data.

```
akiec  (294, 33, 43)
bcc    (463, 51, 93)
bkl    (989, 110, 217)
mel    (1002, 111, 171)
nv     (6034, 671, 908)

((8782, 6), (976, 6), (1432, 6))
```

Figure 6: Output of Dataset Split (Train - Val - Test)

Implementation and Results

CNN Model

We implemented the CNN algorithm using keras from scratch. Training took 8 seconds per epoch using batch size 64 and picture size (32x32x3) thanks to PCA application. Many different models are trained with different parameters such as image size, batch size, conv2d layers, pooling layers, dropout layers and learning rate. [4] Also, ReduceLROnPlateau(), ModelCheckpoint() and EarlyStopping() functions are used as callbacks. This way, the model can improve validation accuracy even after reaching a plateau, and the model with best validation accuracy will be saved, and won't waste our time if validation accuracy is not increasing for some number of epochs. For comparison, 3 different data was used. One is grayscale images, we discovered that color is pretty important for skin lesions and switched to rgb images to increase our accuracy. Also there are two models one of is base data without any augmentation and other one is augmented data. We can see that while total accuracy is decreasing due to undersampling for classes with high number of samples, accuracies for little classes are increasing, as we will observe for each model: recal of class 'mel' which is most dangerous skin lesion that is a direct cause of skin cancer. The best models and their outputs are shared below.

Layer, Filters, Kernel_size	Batch Normalization	Activation	MaxPooling2d	Dropout
Conv2d(32, (3,3))	yes	(‘relu’)	pool_size=(3,3))	(0.25)
Conv2d(64, (5,5))	yes	(‘relu’)	pool_size=(3,3)	(0.25)
Flatten()				
Dense(64)	-	(‘relu’)		(0.25)
Dense(32)	-	(‘relu’)		(0.25)
Dense(7)	-	(‘softmax’)		-

Table 1: CNN Model Summary

Model	**Data**	**Val. Acc.**	**Val. F1 Score**	**Test Acc.**	**Test F1 Score**	**Recall of “mel”**
CNN	Base	77.05%	74.72%	73.46%	72.10%	43.27%
CNN	Augmented	66.49%	66.51%	55.45%	59.40%	47.37%
CNN	Grayscale	58.38%	58.52%	49.65%	53.79%	33.92%

Table 2: Model Evaluation

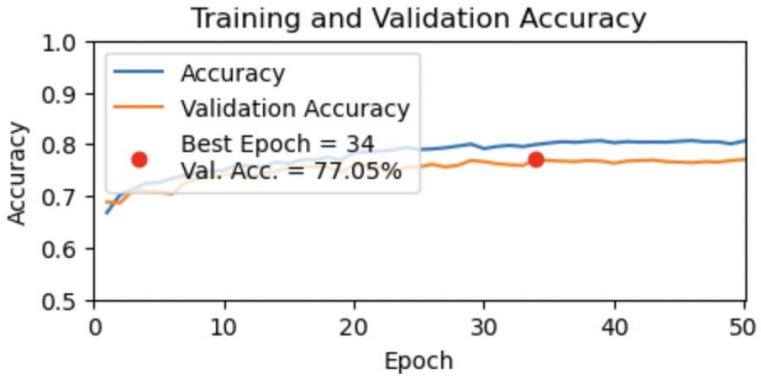

Figure 7: Accuracy Plot During Training using Base Data

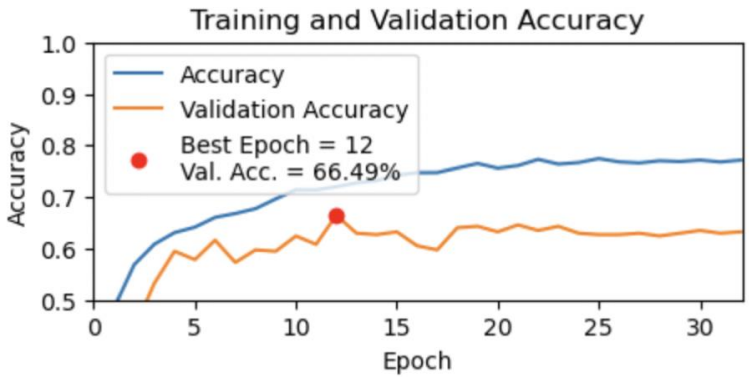

Figure 8: Accuracy Plot During Training using Augmented Data

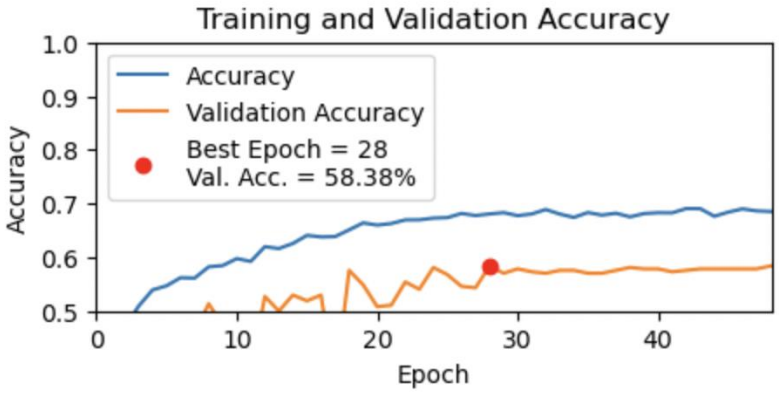

Figure 9: Accuracy Plot During Training using Grayscale Data

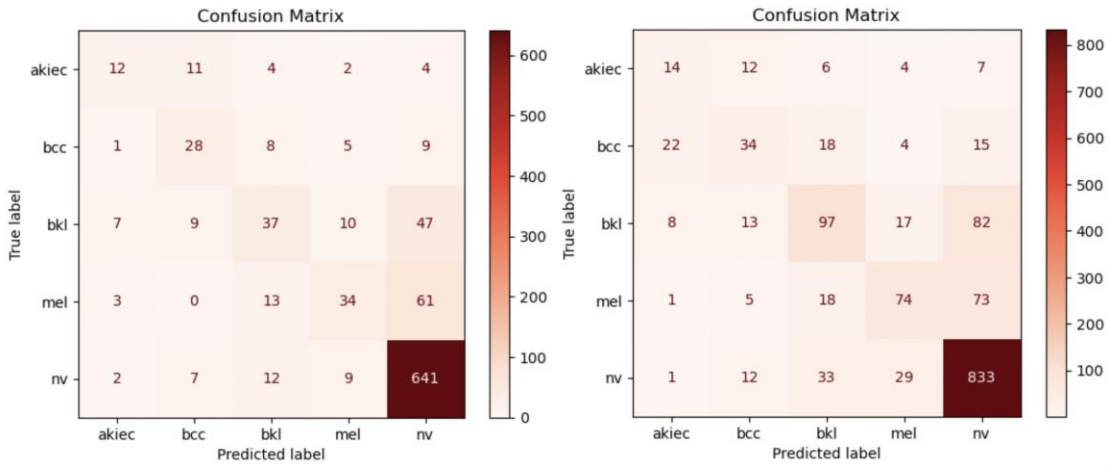

Figure 10: Confusion Matrices of Validation and Test Set Using Base Data

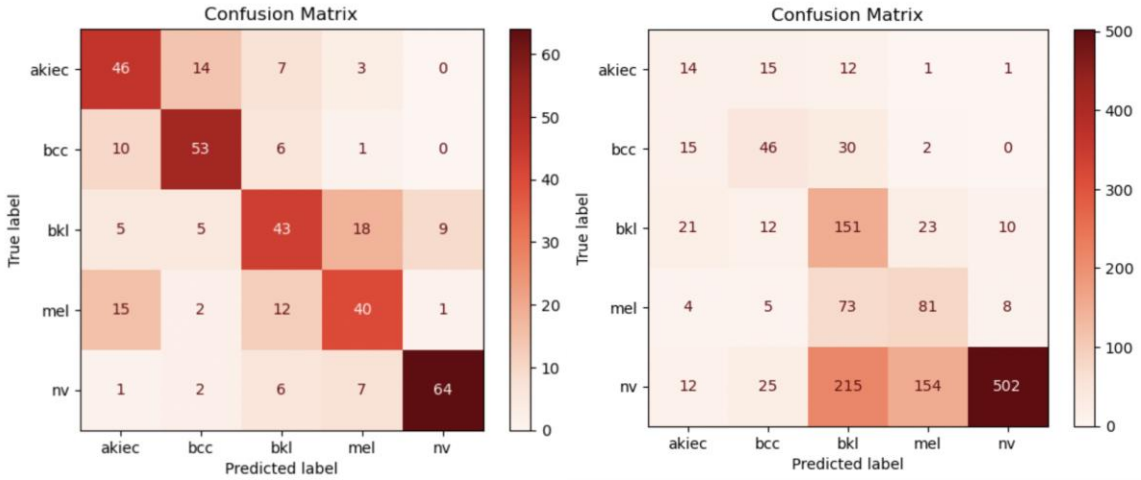

Figure 11: Confusion Matrices of Validation and Test Set Using Augmented Data

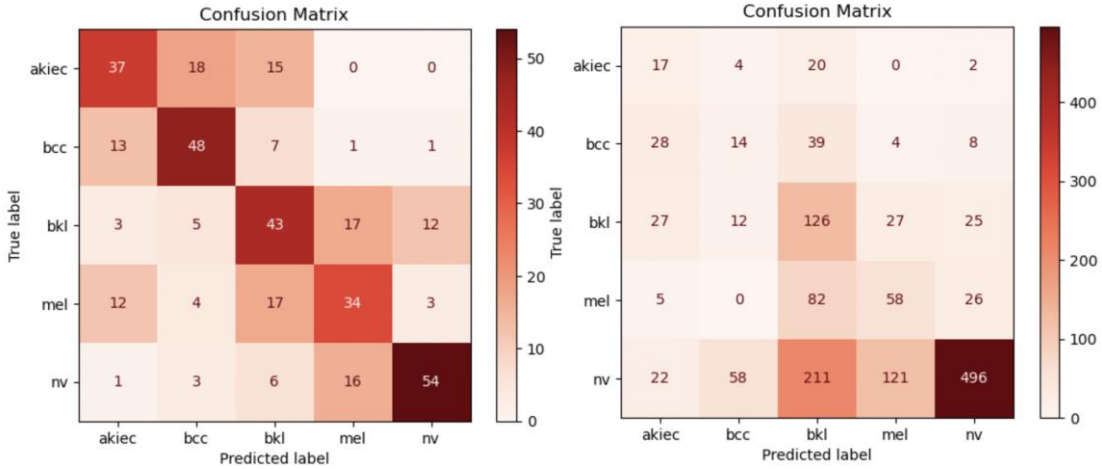

Figure 12: Confusion Matrices of Validation and Test Set Using Grayscale Data

	precision	recall	f1-score	support
akiec	30.43 %	32.56 %	31.46 %	43
bcc	44.74 %	36.56 %	40.24 %	93
bkl	56.40 %	44.70 %	49.87 %	217
mel	57.81 %	43.27 %	49.50 %	171
nv	82.48 %	91.74 %	86.86 %	908
accuracy	73.46 %	73.46 %	73.46 %	1432
macro avg	54.37 %	49.77 %	51.59 %	1432
weighted avg			72.10 %	1432

Figure 13: Classification Report of Test Data with Base Data

	precision	recall	f1-score	support
akiec	21.21 %	32.56 %	25.69 %	43
bcc	44.66 %	49.46 %	46.94 %	93
bkl	31.39 %	69.59 %	43.27 %	217
mel	31.03 %	47.37 %	37.50 %	171
nv	96.35 %	55.29 %	70.26 %	908
accuracy	55.45 %	55.45 %	55.45 %	1432
macro avg	44.93 %	50.85 %	44.73 %	1432
weighted avg			59.40 %	1432

Figure 14: Classification Report of Test Data Using Augmented Data

	precision	recall	f1-score	support
akiec	17.17 %	39.53 %	23.94 %	43
bcc	15.91 %	15.05 %	15.47 %	93
bkl	26.36 %	58.06 %	36.26 %	217
mel	27.62 %	33.92 %	30.45 %	171
nv	89.05 %	54.63 %	67.71 %	908
accuracy	49.65 %	49.65 %	49.65 %	1432
macro avg	35.22 %	40.24 %	34.77 %	1432
weighted avg			53.79 %	1432

Figure 15: Classification Report of Test Data Using Augmented Grayscale Data

ResNet Model

We used the ResNet50 model in our project, which is a Residual Neural Networks (ResNet) variant that is well-known for its effectiveness in deep learning tasks. Because of their special architecture that makes it possible to train extremely deep networks, ResNet models—particularly ResNet50—are essential to computer vision and deep learning. This is mostly accomplished by using "skip connections" or "residual connections," which aid in resolving the "vanishing gradient problem,"

a prevalent problem in deep networks where gradients get too small to produce meaningful backpropagation updates.

Our project's ResNet50 model was specifically adapted for skin cancer prediction. Implemented using the PyTorch framework, the model was trained from scratch without the use of pre-trained weights. We tailored the top layers of our ResNet50 model to suit the task at hand, including the integration of a dropout layer at a rate of 20% to prevent overfitting—a critical factor when working with medical imagery where accuracy is paramount.

We added a dense layer with 128 units and 'relu' activation after the dropout layer. Because of its full connectivity, this layer enables the model to learn non-linear combinations of the high-level features that the ResNet50 base extracted. Following, a second dropout layer was added, this time at the same 20% rate to reinforce the model's resistance to overfitting.

Our model's last layer is dense, with units in it that correspond to the number of classes in our dataset (found by measuring the labels array's length). This layer is appropriate for multi-class classification—in our case, different forms of skin cancer—because it makes use of the 'softmax' activation function. The probability distribution over the classes produced by the 'softmax' function enables an easily understood classification.

In our project, the ResNet50 model was trained twice using the same dataset with no augmentation and with augmentation to observe the effect of augmentation. The first training utilized original images with PCA. PCA helped reduce the dimensionality of our data, enhancing computational efficiency without significant loss of information. However, two classes with minimal image counts were excluded from this dataset to improve model accuracy, as their scarcity negatively impacted performance.

For the second training, we employed data augmentation techniques. This approach was vital in addressing class imbalance, a common challenge in medical image datasets. By augmenting the images, we created a more balanced dataset with approximately 1000 images per class. This not only helped in mitigating overfitting but also ensured that the model was exposed to a variety of transformations, mimicking real-world variations in skin cancer presentations.

Model	Data	Val. Acc.	Val. F1 Score	Test Acc.	Test F1 Score	Recall of “mel”
ResNet	Base	78.07%	77.11%	71.50%	71.08%	44.44%
ResNet	Augmented	65.6%	64.76%	59.07%	62.58%	46.78%

Table 3: ResNet Model Evaluation

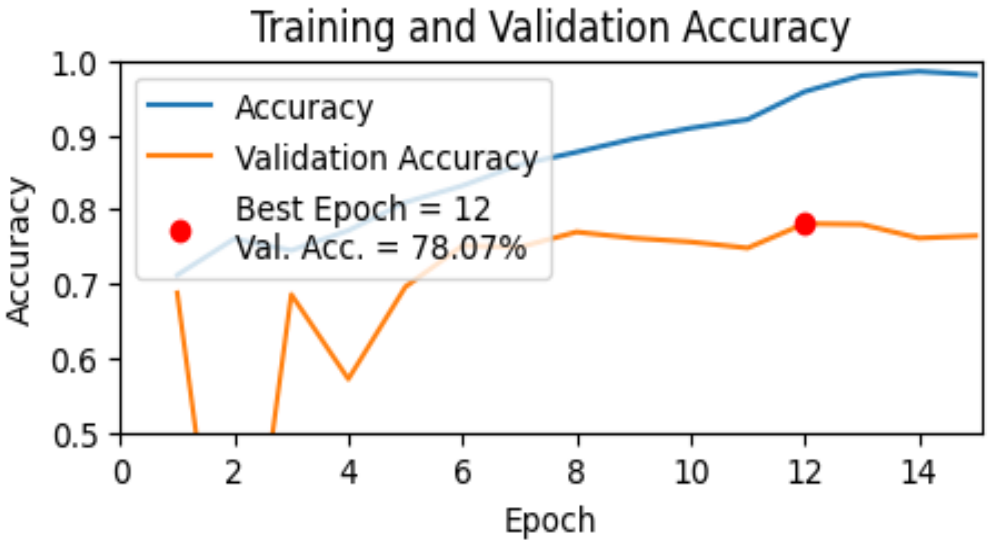

Figure 16: Accuracy Plot During Training with Base Data

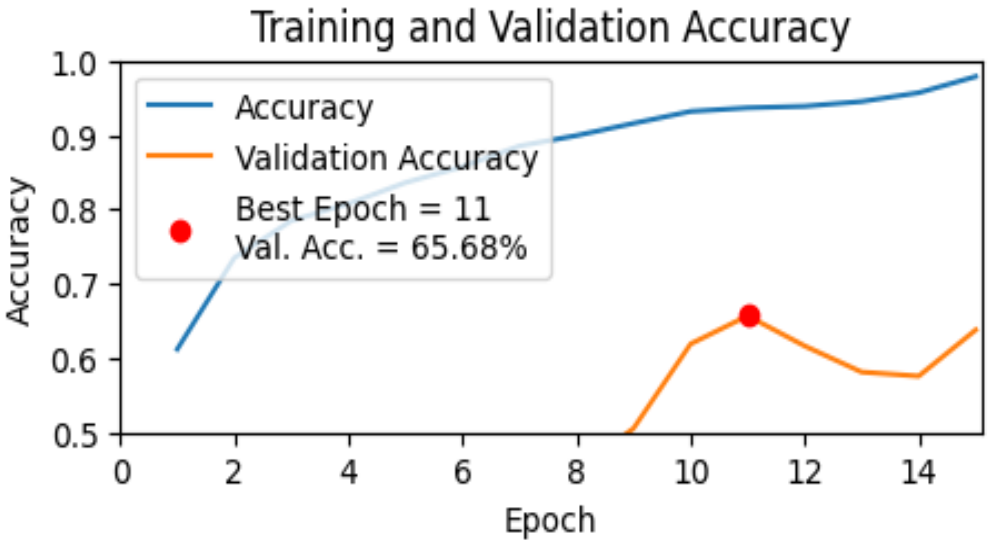

Figure 17: Accuracy Plot During Training with Augmented Data

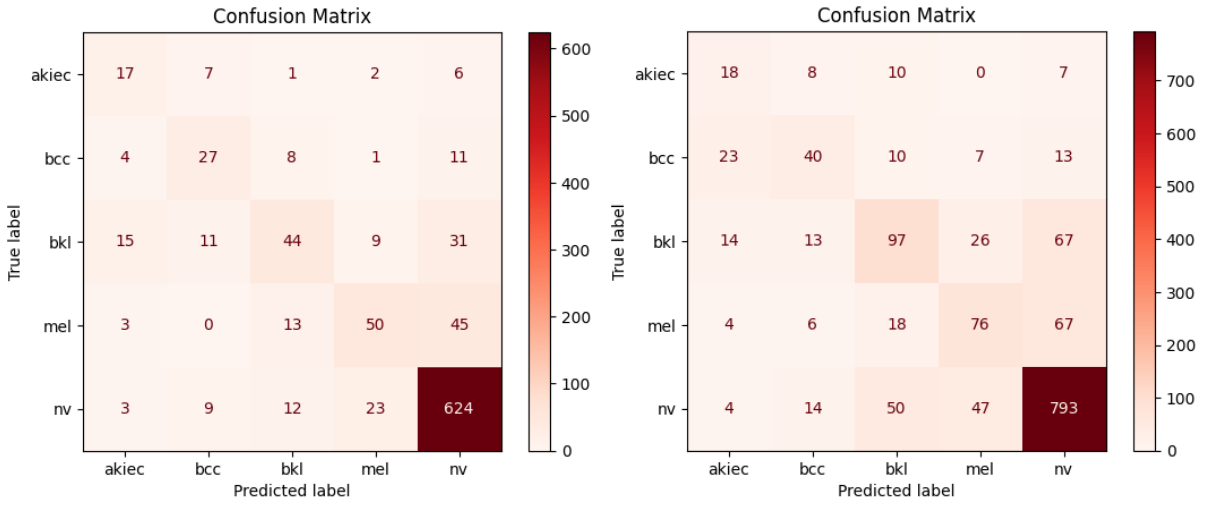

Figure 18: Confusion Matrices of Validation and Test Set Using Base Data

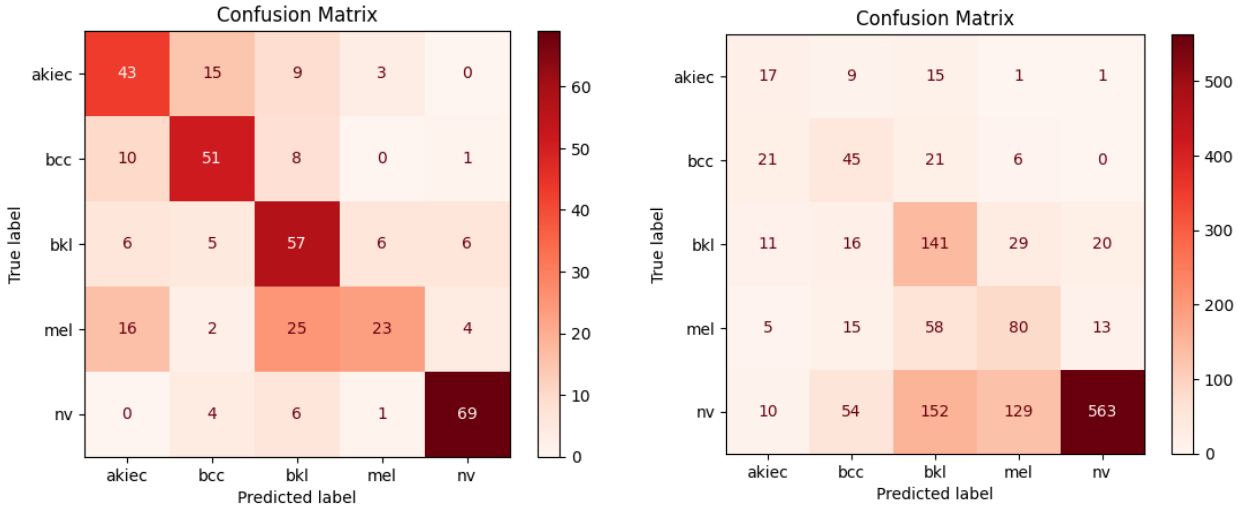

Figure 19: Confusion Matrices of Validation and Test Set Using Augmented Data

	precision	recall	f1-score	support
akiec	28.57 %	41.86 %	33.96 %	43
bcc	49.38 %	43.01 %	45.98 %	93
bkl	52.43 %	44.70 %	48.26 %	217
mel	48.72 %	44.44 %	46.48 %	171
nv	83.74 %	87.33 %	85.50 %	908
accuracy	71.51 %	71.51 %	71.51 %	1432
macro avg	52.57 %	52.27 %	52.04 %	1432
weighted avg			71.08 %	1432

Figure 20: Classification Report of Test Data with Base Data

	precision	recall	f1-score	support
akiec	26.56 %	39.53 %	31.78 %	43
bcc	32.37 %	48.39 %	38.79 %	93
bkl	36.43 %	64.98 %	46.69 %	217
mel	32.65 %	46.78 %	38.46 %	171
nv	94.30 %	62.00 %	74.82 %	908
accuracy	59.08 %	59.08 %	59.08 %	1432
macro avg	44.47 %	52.34 %	46.11 %	1432
weighted avg			62.58 %	1432

Figure 21: Classification Report of Test Data with Augmented Data

Transfer Learning

We implemented VGG-19 as our transfer learning model. VGG-19 is a deep convolutional neural network consisting of 19 layers (16 convolutional and 3 fully connected), as seen in Figure 18 [6]. Traditionally, it employs mean subtraction for preprocessing and 3x3 kernels with stride 1, preserving spatial resolution via spatial padding. It utilized max pooling (2x2 windows, stride 2) for downsampling, ReLU for non-linearity, and featured three fully connected layers, with the last layer comprising 1000 channels for 1000-way ILSVRC classification using a softmax function for predictions [7]. VGG-19 is effective in transfer learning as its uniform structure with 3x3 convolutional layers and max pooling makes it versatile for various image-related tasks. Additionally, pre-trained VGG-19 models on large datasets (like ImageNet) offer a strong starting point for transfer learning by providing rich feature representations that can be fine-tuned for specific tasks with smaller datasets. This transferability of learned features across diverse visual recognition tasks makes VGG-19 a strong candidate for transfer learning applications.

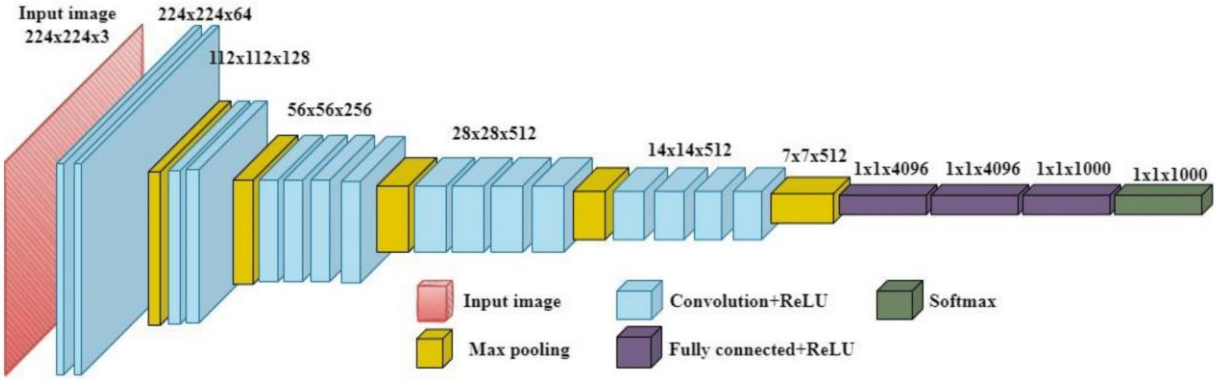

Figure 22: VGG-19 architecture

We used Keras's VGG model, with include_top = False and weights = 'imagenet' as the base model. We added 3 dense layers (512, 128, 32 respectively) with uniform kernel initializers and ReLu activation. Lastly, we had a final softmax layer for classification (5 classes). Training took 460 seconds per epoch using batch size 64 and picture size (32x32x3). Also, ReduceLROnPlateau() and ModelCheckpoint() functions are used as callbacks. This way, the model can improve validation accuracy even after reaching a plateau, and the model with best validation accuracy will be saved. The best models and their outputs are shared below.

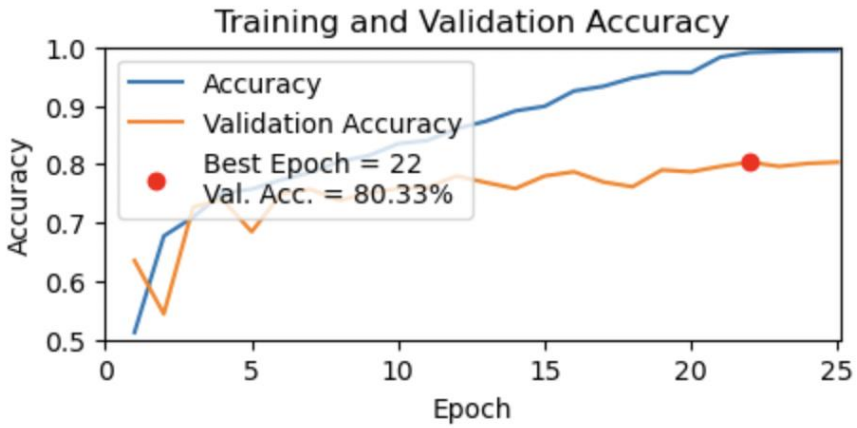

Figure 23: Accuracy Plot During Training of VGG-19 Model on Base Data

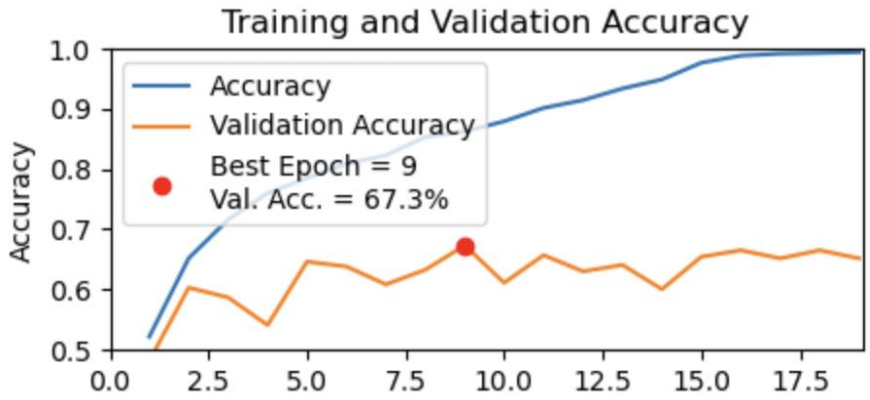

Figure 24: Accuracy Plot During Training of VGG-19 Model on Augmented Data

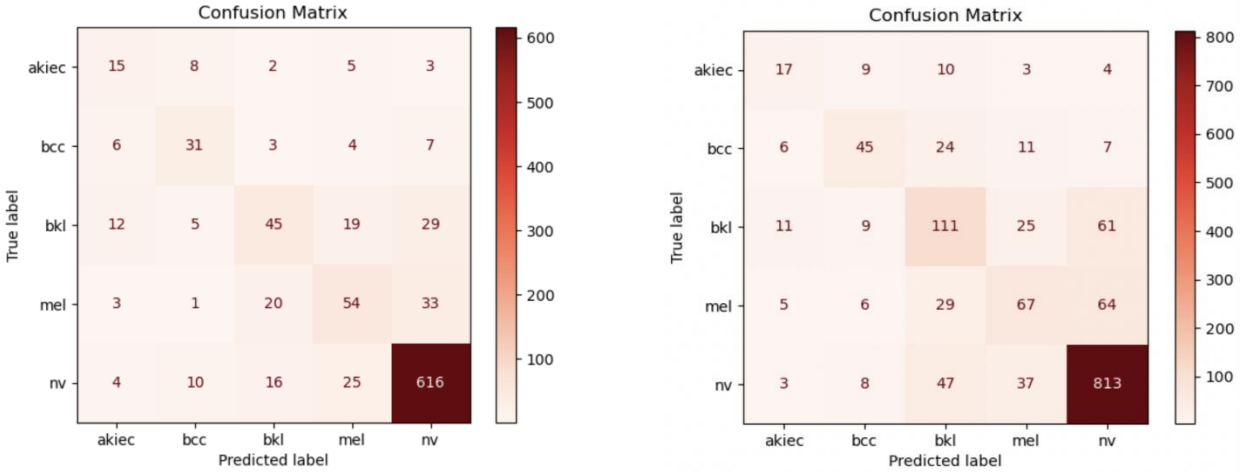

Figure 25: Confusion Matrices of Validation and Test Set Using Base Data

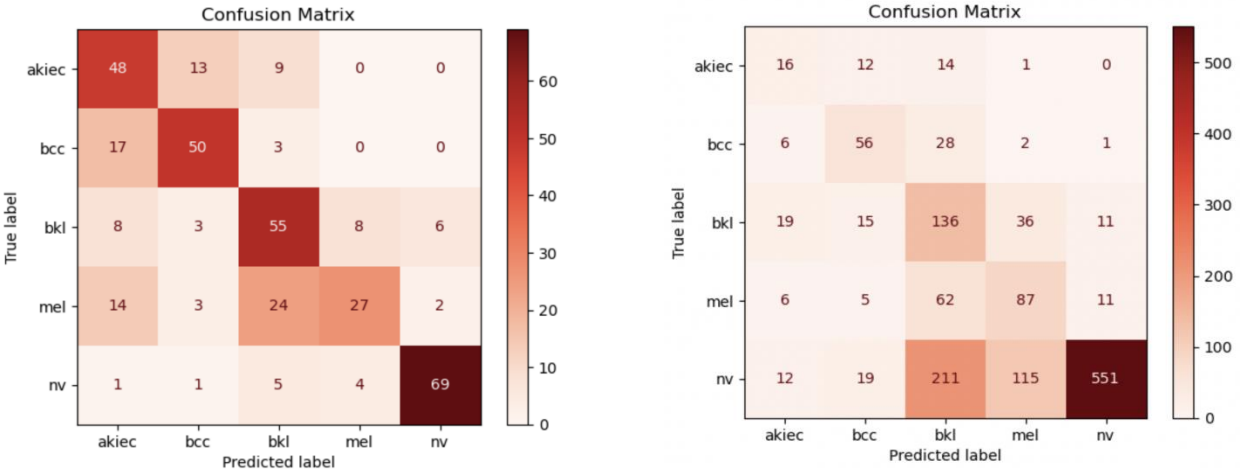

Figure 26: Confusion Matrices of Validation and Test Set Using Augmented Data for VGG19

	precision	recall	f1-score	support
akiec	40.48 %	39.53 %	40.00 %	43
bcc	58.44 %	48.39 %	52.94 %	93
bkl	50.23 %	51.15 %	50.68 %	217
mel	46.85 %	39.18 %	42.68 %	171
nv	85.67 %	89.54 %	87.56 %	908
accuracy	73.53 %	73.53 %	73.53 %	1432
macro avg	56.33 %	53.56 %	54.77 %	1432
weighted avg			72.94 %	1432

Figure 27: Classification Report of Test Data with Base Data for VGG19

	precision	recall	f1-score	support
akiec	27.12 %	37.21 %	31.37 %	43
bcc	52.34 %	60.22 %	56.00 %	93
bkl	30.16 %	62.67 %	40.72 %	217
mel	36.10 %	50.88 %	42.23 %	171
nv	95.99 %	60.68 %	74.36 %	908
accuracy	59.08 %	59.08 %	59.08 %	1432
macro avg	48.34 %	54.33 %	48.94 %	1432
weighted avg			62.94 %	1432

Figure 28: Classification Report of Test Data with Augmented Data for VGG19

Model	Data	Val. Acc.	Val. F1 Score	Test Acc.	Test F1 Score	Recall of "mel"
VGG-19	Base	80.33%	77.58%	73.53%	72.94%	39.18%
VGG-19	Augmented	67.3%	66.9%	59.08%	62.94%	50.88%

Table 4: Transfer Learning Model Evaluation

Hybrid Approach using XGBoost

After implementing all models, we were searching for different ways to get better results. We realized that we have a csv file containing patient demographics data and also clinical data such as lesion's location on body. Since it is not possible to use such data in convolutional neural networks, we decided to use convolutional neural networks for just feature extraction. After extracting features, we can merge features we got from convolutional neural network models and csv demographics data. Then using this merged data to create another model, ensemble of decision trees, XGBoost. By using this approach, we are able to get best results for the most dangerous skin lesion 'mel' and also best results overall.

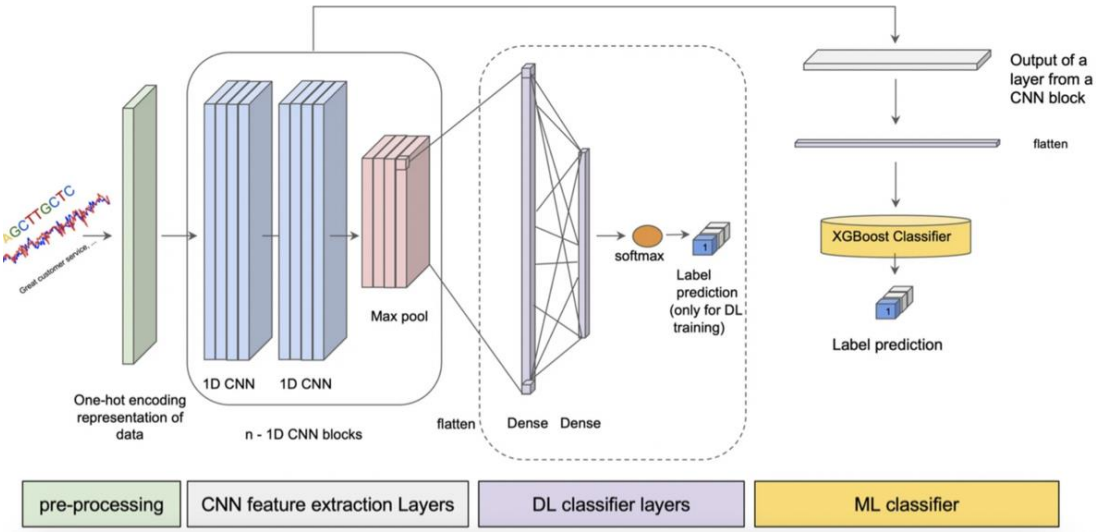

Figure 29: Hybrid Approach Summary [8]

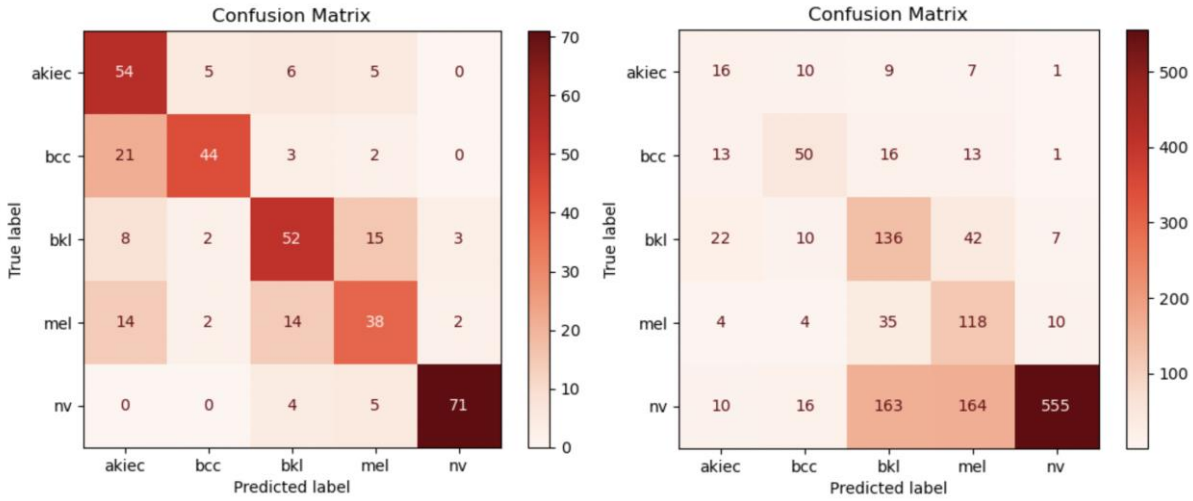

Figure 30: Confusion Matrices of Validation and Test Set Using Augmented Data for VGG19+XGBoost

	precision	recall	f1-score	support
akiec	24.62 %	37.21 %	29.63 %	43
bcc	55.56 %	53.76 %	54.64 %	93
bkl	37.88 %	62.67 %	47.22 %	217
mel	34.30 %	69.01 %	45.83 %	171
nv	96.69 %	61.12 %	74.90 %	908
accuracy	61.10 %	61.10 %	61.10 %	1432
macro avg	49.81 %	56.75 %	50.44 %	1432
weighted avg			64.56 %	1432

Figure 31: Classification Report of Test Data with Augmented Data for VGG19+XGBoost

Discussion on Results

We started to train our models using grayscale images. But we discovered that color is a very important aspect for skin lesions. Because of that, we used rgb color to train all our models. One grayscale trained model can be seen below for comparison, all other models use rgb colored images.

The plain CNN model, despite its simplicity, gave pretty decent results compared to huge transfer learning models. Results with base data and augmented data can be seen below. After the plain CNN model, we wanted to train a more complex model: ResNet. Due to more connections between layers, ResNet gave better results. But these results weren't enough for the skin cancer classification task. For comparison we trained some transfer learning models. VGG-19 gave better results in every metric. For evaluation purposes, we give priority to the metric of "recall of 'mel' class" Even though we trained a very complex model, we weren't able to get perfect results. Due to the complexity of skin lesions, this might be expected. Even dermatologists cannot detect skin cancers without pathological tests. After all the models, we found a way to improve the results a little bit more: using CNN for feature extraction and then use both extracted features from images and given patient demographics in csv file, then training all of them using XGBoost classifier. This approach increased F1 Score on test data around 2% and increased recall of class 'mel' 20%.

To sum up, CNN's results were decent. Then the ResNet model gave us better results. But as expected, transfer learning using imagenet weights gave best results between all three CNN classifiers. Feature extraction + XGBoost can be applied for all models, but to keep it simple we only added XGBoost classifier to our best model, which is VGG-19. This way we got our best results.

Model	Data	Val. Acc.	Val. F1 Score	Test Acc.	Test F1 Score	Recall of “mel”
CNN	Grayscale	58.38%	58.52%	49.65%	53.79%	33.92%
CNN	Base	77.05%	74.72%	73.46%	72.10%	43.27%
ResNet	Base	78.07%	77.11%	71.50%	71.08%	44.44%
VGG-19	Base	80.33%	77.58%	73.53%	72.94%	39.18%
CNN	Augmented	66.49%	66.51%	55.45%	59.40%	47.37%
ResNet	Augmented	65.6%	64.76%	59.07%	62.58%	46.78%
VGG-19	Augmented	67.3%	66.9%	59.08%	62.94%	50.88%
+XGB	Augmented	70.00%	70.24%	61.10%	64.56%	69.01%

Table 5: Models Evaluation

Conclusion

During this project, we investigated different ways to improve skin cancer classification. After lots of research and model training, it can be said that the skin cancer classification task still needs more data to give more reliable results. Since one of the important criteria to detect skin cancer is change rate, this task needs longitudinal data instead of cross-sectional data. Also, to get almost perfect results, we need pathological test results. But at least, machine learning models for this task can give doctors another perspective and calculate malignancy probability which humans cannot do.

Despite the concerns and downsides, using deep learning on skin cancer detection can be a useful tool in the hands of dermatologists, which can potentially save human lives since early detection is very important in cancer. In conclusion, leveraging deep learning algorithms for skin cancer detection represents a significant leap forward in early diagnosis. These technologies can offer unprecedented accuracy and efficiency, potentially saving lives through timely intervention. As we harness the power of artificial intelligence, we move closer to a future where skin cancer is not just detected but prevented.

Work Done

Bekir Burak Çelik: Implemented PCA application, optimized CNN models and hyperparameters, and contributed to the reports. He focused on training and optimizing the ResNet model for the final.

Bilgehan Yılmaz Sandıkcı: Optimized CNN models and hyperparameters, contributed to the reports, and for the final, concentrated on training and fine-tuning the ResNet model.

Fadilah Zakaria: Optimized CNN models, contributed to the progress report, and worked on transfer learning and fine-tuning the model for the final report.

Mehmet Bayık: Implemented CNN and XGBoost models, optimized CNN models, worked on data augmentation, transfer learning, and contributed to reports.

Piotr Wojslawski: Implemented image augmentation, optimized CNN models, and for the final, addressed classes with lower performance, deciding on their retention or enhancement through synthetic data generation.

References

[1] F. Kabir, "Skin Cancer Dataset," Kaggle, 2022. Online. Available: https://www.kaggle.com/datasets/farjanakabirsamanta/skin-cancer-dataset/ (accessed Nov. 25, 2023).
[2] P. Tschandl, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," Harvard Dataverse, V4, 2018. [Online]. Available: https://doi.org/10.7910/DVN/DBW86T. [Accessed: dd-mm-yyyy].

[3]: "What are convolutional neural networks?," IBM, https://www.ibm.com/topics/convolutional-neural-networks (accessed Nov. 25, 2023).

[4]: Anh H. Reynolds, "Convolutional Neural Networks (cnns)," Anh H. Reynolds, https://anhreynolds.com/blogs/cnn.html (accessed Nov. 25, 2023).

[5]: S. Ramesh, "A guide to an efficient way to build neural network architectures- part II: Hyper-parameter...," Medium, https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7 (accessed Nov. 25, 2023).

[6]: Thanh-Hai Nyugen Et. al,"A VGG-19 Model with Transfer Learning and Image Segmentation for Classification of Tomato Leaf Disease", MDPI, 2022. https://www.mdpi.com/2624-7402/4/4/56 (accessed Dec. 22, 2023).

[7]: Aakash Kaushik, "Understanding the VGG19 Architecture" Open Genus IQ, https://iq.opengenus.org/vgg19-architecture/ (accessed Dec. 22, 2023).

[8]: P. Mavaie, L. Holder, and M. K. Skinner, "Hybrid deep learning approach to improve classification of low-volume high-dimensional data," *BMC Bioinformatics*, vol. 24, no. 1, Jul. 2023 [Online]. Available: https://doi:10.1186/s12859-023-05557-w

Appendix A: PCA Application for Images

```
# %%
import cv2
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# %%
def pca_test(image_dir, pca_components = 50, save = False):
    temp_dir = image_dir
    img = cv2.cvtColor(cv2.imread(temp_dir), cv2.COLOR_BGR2RGB)
    r, g, b = cv2.split(img)
    r, g, b = r / 255, g / 255, b / 255

    pca_r = PCA(n_components = pca_components)
    reduced_r = pca_r.fit_transform(r)

    pca_g = PCA(n_components = pca_components)
    reduced_g = pca_g.fit_transform(g)

    pca_b = PCA(n_components = pca_components)
    reduced_b = pca_b.fit_transform(b)

    print(f"Red Channel  : {sum(pca_r.explained_variance_ratio_)}")
    print(f"Green Channel: {sum(pca_g.explained_variance_ratio_)}")
    print(f"Blue Channel : {sum(pca_b.explained_variance_ratio_)}")

    print(reduced_r.shape)
    print(reduced_g.shape)
    print(reduced_b.shape)

    fig = plt.figure(figsize = (10, 2))
    fig.add_subplot(131)
    plt.title("Red Channel")
    plt.ylabel('Variation explained')
    plt.xlabel('Eigen Value')
    plt.bar(list(range(1,pca_components+1)),pca_r.explained_variance_ratio_)
    fig.add_subplot(132)
    plt.title("Green Channel")
```

```
    plt.xlabel('Eigen Value')
    plt.bar(list(range(1,pca_components+1)),pca_g.explained_variance_ratio_)
    fig.add_subplot(133)
    plt.title("Blue Channel")
    plt.xlabel('Eigen Value')
    plt.bar(list(range(1,pca_components+1)),pca_b.explained_variance_ratio_)
    plt.show()

    reconstructed_r = pca_r.inverse_transform(reduced_r)
    reconstructed_g = pca_g.inverse_transform(reduced_g)
    reconstructed_b = pca_b.inverse_transform(reduced_b)

    img_reconstructed = (cv2.merge((reconstructed_r, reconstructed_g,
reconstructed_b)))
    img_reconstructed = (img_reconstructed * 255).astype(np.uint8)

    fig2 = plt.figure(figsize = (15, 2))
    fig2.add_subplot(121)
    plt.title("Original Image")
    plt.imshow(img)
    fig2.add_subplot(111)
    plt.title("Reconstructed Image")
    plt.imshow(img_reconstructed)
    if save== True:
        plt.imsave('pca-test.jpg', img_reconstructed)

# %%
# Test for one image
pca_test('dataverse_files/HAM10000_images/ISIC_0024306.jpg', pca_components = 50, save
= False)

# %%
def img_pca(img_name, pca_components, input_dir, output_dir):

    temp_dir = input_dir + img_name

    img = cv2.cvtColor(cv2.imread(temp_dir), cv2.COLOR_BGR2RGB)
    r, g, b = cv2.split(img)
    r, g, b = r / 255, g / 255, b / 255

    pca_r = PCA(n_components = pca_components)
    reduced_r = pca_r.fit_transform(r)
```

```
    pca_g = PCA(n_components = pca_components)
    reduced_g = pca_g.fit_transform(g)
    pca_b = PCA(n_components = pca_components)
    reduced_b = pca_b.fit_transform(b)

    reconstructed_r = pca_r.inverse_transform(reduced_r)
    reconstructed_g = pca_g.inverse_transform(reduced_g)
    reconstructed_b = pca_b.inverse_transform(reduced_b)

    img_reconstructed = (cv2.merge((reconstructed_r, reconstructed_g,
reconstructed_b)))
    img_reconstructed = img_reconstructed * 255
    img_reconstructed = img_reconstructed.astype(np.uint8)

    output_image = output_dir + img_name
    plt.imsave(output_image, img_reconstructed)

# %%
metadata = pd.read_csv('dataverse_files/HAM10000_metadata.csv')
img_list = metadata['image_id'].tolist()

input_dir = 'dataverse_files/HAM10000_images/' # this is the folder with all the
images
output_dir = 'dataverse_files/HAM10000_images_pca/' # this is the folder where the pca
images will be saved
if not os.path.exists(output_dir):
    os.mkdir(output_dir)


# this will take a while to run ( around 30 mins )
# uncomment to run loop

# for image in img_list:
#     img_dir = image + '.jpg'
#     img_pca(img_dir, 50, input_dir, output_dir)
```

Appendix B: Data Augmentation for Images

```
# %%
import os
import datetime
import random
import warnings
import shutil
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize,LinearSegmentedColormap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import sklearn.exceptions
from sklearn.utils import class_weight
warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)
import tensorflow as tf
from tensorflow.python.client import device_lib
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import
Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalization,Activatio
n,MaxPooling2D, ZeroPadding2D
from keras.models import Model,Sequential
from keras.optimizers import Adam,SGD,RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
# import tensorflow_addons as tfa
#finish_sound = "afplay /Users/mehmet/Documents/vs-code/winsquare.mp3"
# !jupyter nbconvert --to html skin-cancer-cnn.ipynb

# %%
df=pd.read_csv('dataverse_files/HAM10000_metadata.csv')
df = df[df['dx'] != 'vasc']
df = df[df['dx'] != 'df']

# %%
```

```
df_unique = df.copy()
df_unique.drop_duplicates( subset=['lesion_id'], keep=False, inplace=True)
#df_unique=df_unique.drop([ 'dx_type', 'age', 'sex', 'localization', 'dataset'],
axis=1)
df_unique = df_unique.reset_index(drop=True)
#df=df.drop([ 'dx_type', 'age', 'sex', 'localization', 'dataset'], axis=1)
df_extra = df[~df['image_id'].isin(df_unique['image_id'])] # lesions with multiple
images

df_one_of_each = df.copy()
df_extra_unique = df_extra['lesion_id'].unique()
# how to get one image of each lesion_id
for i in df_extra_unique:
   df_one_of_each = df_one_of_each.drop(df_one_of_each[df_one_of_each['lesion_id'] ==
i].index[1:])

labels = df['dx'].sort_values().unique()

train_df_unique, val_df_unique = train_test_split(df_unique, train_size=0.8,
shuffle=True, random_state=123, stratify=df_unique['dx'])

train_df = df[~df['image_id'].isin(val_df_unique['image_id'])] # all train data
train_df_extra = train_df[~train_df['image_id'].isin(train_df_unique['image_id'])] #
lesions with multiple images

train_df_one_of_each = train_df.copy()
train_df_extra_unique = train_df_extra['lesion_id'].unique()
# how to get one image of each lesion_id
for i in train_df_extra_unique:
   train_df_one_of_each =
train_df_one_of_each.drop(train_df_one_of_each[train_df_one_of_each['lesion_id'] ==
i].index[1:])

print( 'All data: ',df.shape[0], '| Unique Images:',df_unique.shape[0], '| Data with
extras:',df_extra.shape[0])
print('One image from each lesion in data:',df_one_of_each.shape[0])
print('All train data:',train_df.shape[0], '| Unique Train
Data:',train_df_unique.shape[0], '| Train data with extras:',train_df_extra.shape[0])
print('One image from each lesion in train data:',train_df_one_of_each.shape[0])
print('Validation data',val_df_unique.shape[0])
```

```
# %%
#plotting class distribution for lesion_id and image_id before rebalancing the class
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 3))
bins = np.linspace(0 - .25, len(labels)-1 + .25, 2*len(labels))

# Plotting class distribution for training set

ax1.hist(train_df['dx'].sort_values(),bins=bins)
ax1.hist(train_df_one_of_each['dx'].sort_values(),bins=bins)
ax1.set_title("Class Distribution for Training Set")
ax1.set_xlabel('Class Labels')
ax1.set_ylabel('Number of Images')
ax1.legend(['Extra Images for Same Lesions', 'Unique Lesions'])

# Plotting class distribution  for val set
ax2.hist(val_df_unique['dx'].sort_values(),bins=bins)
ax2.hist(val_df_unique['dx'].sort_values(),bins=bins)
ax2.set_title("Class Distribution for Validation Set")
ax2.set_xlabel('Class Labels')
ax2.set_ylabel('Number of Images')
ax2.legend(['Extra Images for Same Lesions', 'Unique Lesions'])

plt.tight_layout()
#plt.savefig('class_distribution.png')
plt.show()

# %%
def random_augment(image):
   tf.random.set_seed(123)
   # Randomly applied horizontal flip - reasonable for skin images
   image = tf.image.random_flip_left_right(image)

   # Random brightness - simulate different lighting conditions
   image = tf.image.random_brightness(image, max_delta=0.1)

   # Random contrast - simulate variations in camera quality and settings
   image = tf.image.random_contrast(image, lower=0.8, upper=1.2)

   # Random rotation - lesions can be oriented in any direction
   image = tf.image.rot90(image, k=tf.random.uniform(shape=[], minval=0, maxval=4,
dtype=tf.int32))
```

```
    # Gaussian blur - simulate slight focus variations
    # image = tfa.image.gaussian_filter2d(image, filter_shape=(3, 3), sigma=1.0)

    return image

# %%
def augmentention_on_dataset(df, df_unique, dataset_type, images_directory,
max_sample, min_sample, x=''):
    if not os.path.exists(images_directory+'_augmented'+x):
        os.mkdir(images_directory+'_augmented'+x)

    if df.shape == df_unique.shape:
        print('For Validation Set')
        for label in df['dx'].unique():
            df_label = df[df['dx'] == label]
            df_unique_label = df_unique[df_unique['dx'] == label]

            #If in the label is more than n images we delete all duplicates and
sometimes part of random unique imagas
            if(len(df_label) > max_sample):
                df = pd.concat([df[df['dx'] != label], df_unique[df_unique['dx'] ==
label]])
                df_label = df[df['dx'] == label]
                drop_indices = np.random.choice(df[df['dx'] == label].index,
len(df_label) - max_sample, replace=False)
                df = pd.concat([df[df['dx'] != label], df_label.drop(drop_indices)])

            #If in the label is less than n images we randomly choose unique images for
augmentation
            if(len(df_label) < min_sample):
                selected_indices = np.random.choice(df_unique_label.index, min_sample -
len(df_label), replace=True)
                df_to_add = df_unique_label.loc[selected_indices]
                df_to_add['image_id'] = df_to_add['image_id'] + '-' + dataset_type +
(df_to_add.groupby('image_id').cumcount() + 1).astype(str)
                df = pd.concat([df, df_to_add])

        for j in df[df['image_id'].str.len() == 12]['image_id'].values:
            # copy image without change to augmented folder
            shutil.copy(images_directory + '/'+j+'.jpg', images_directory +
'_augmented'+x+ '/'+j+'.jpg')
```

```
        for i in df[df['image_id'].str.len() > 12]['image_id'].values:
            image = tf.io.read_file(os.path.join(images_directory, i.split('-')[0] +
'.jpg'))
            image = tf.image.decode_jpeg(image, channels=3)

            # Apply augmentations
            augmented_image = random_augment(image)

            # Convert back to image format and save the augmented image
            augmented_image = tf.cast(augmented_image, tf.uint8)
            augmented_image = tf.image.encode_jpeg(augmented_image)
            tf.io.write_file(os.path.join(images_directory+'_augmented'+x, i + '.jpg'),
augmented_image)

        df['image_id'] = df['image_id'].astype(str) + '.jpg'
        df_shuffled = df.copy().sample(frac=1, random_state=123).reset_index(drop=True)

    else:
        print('For Training Set')

        df_extra = df[~df['image_id'].isin(df_unique['image_id'])]
        df_extra_unique = df_extra.copy()
        df_extra_unique_list = df_extra['lesion_id'].unique()
        # how to get one image of each lesion_id
        for i in df_extra_unique_list:
            df_extra_unique =
df_extra_unique.drop(df_extra_unique[df_extra_unique['lesion_id'] == i].index[1:])
        df_extra_extra =
df_extra[~df_extra['image_id'].isin(df_extra_unique['image_id'])]

        for label in df['dx'].unique():
            df_label = df[df['dx'] == label]
            df_unique_label = df_unique[df_unique['dx'] == label]
            df_extra_label = df_extra[df_extra['dx'] == label]
            df_extra_unique_label = df_extra_unique[df_extra_unique['dx'] == label]
            df_extra_extra_label = df_extra_extra[df_extra_extra['dx'] == label]

            # If there are more than n images in the label, we delete all duplicates
and sometimes part of random unique images
            if(len(df_label) > max_sample):
                delete = len(df_label) - max_sample
                if len(df_extra_extra_label) > delete:
```

```
                    drop_indices = np.random.choice(df_extra_extra[df_extra_extra['dx']
== label].index, delete, replace=False)
                    df = pd.concat([df[df['dx'] != label],
df_extra_extra_label.drop(drop_indices), df_unique[df_unique['dx'] == label],
df_extra_unique[df_extra_unique['dx'] == label]])
                else:
                    if len(df_extra_label) > delete:
                        drop2 = delete - len(df_extra_extra_label)
                        drop_indices2 =
np.random.choice(df_extra_unique[df_extra_unique['dx'] == label].index, drop2,
replace=False)
                        df = pd.concat([df[df['dx'] != label],
df_extra_unique_label.drop(drop_indices2), df_unique[df_unique['dx'] == label]])
                    else:
                        drop2 = delete - len(df_extra_label)
                        drop_indices2 = np.random.choice(df_unique[df_unique['dx'] ==
label].index, drop2, replace=False)
                        df = pd.concat([df[df['dx'] != label],
df_unique_label.drop(drop_indices2)])

            # If there are less than n images in the label, we randomly choose unique
images for augmentation
            if(len(df_label) < min_sample):
                selected_indices = np.random.choice(df_unique_label.index, min_sample -
len(df_label), replace=True)
                df_to_add = df_unique_label.loc[selected_indices]
                df_to_add['image_id'] = df_to_add['image_id'] + '-' + dataset_type +
(df_to_add.groupby('image_id').cumcount() + 1).astype(str)
                df = pd.concat([df, df_to_add])

        for j in df[df['image_id'].str.len() == 12]['image_id'].values:
            # copy image without change to augmented folder
            shutil.copy(images_directory + '/'+j+'.jpg', images_directory +
'_augmented'+x+ '/'+j+'.jpg')

        for i in df[df['image_id'].str.len() > 12]['image_id'].values:
            image = tf.io.read_file(os.path.join(images_directory, i.split('-')[0] +
'.jpg'))
            image = tf.image.decode_jpeg(image, channels=3)

            # Apply augmentations
            augmented_image = random_augment(image)
```

```
            # Convert back to image format and save the augmented image
            augmented_image = tf.cast(augmented_image, tf.uint8)
            augmented_image = tf.image.encode_jpeg(augmented_image)
            tf.io.write_file(os.path.join(images_directory+'_augmented'+x, i + '.jpg'),
augmented_image)

        df['image_id'] = df['image_id'].astype(str) + '.jpg'
        df_shuffled = df.copy().sample(frac=1, random_state=123).reset_index(drop=True)
        #df_shuffled.to_csv(f'dataverse_files/HAM10000_metadata_augmented_' +
dataset_type +'.csv', index=False)


    return df_shuffled

# %%
print_list = 'All', 'Unique', 'One of Each', 'Val all unique'
print(print_list)
for label in labels:
    list1 = len(train_df[train_df['dx'] == label]),
len(train_df_unique[train_df_unique['dx'] == label]),
len(train_df_one_of_each[train_df_one_of_each['dx'] == label]),
len(val_df_unique[val_df_unique['dx'] == label])
    space = ' '

    print(label,(5-len(label))*space ,list1)
train_df.shape[0], train_df_unique.shape[0], train_df_one_of_each.shape[0],
val_df_unique.shape[0]

# %%
input_path = "dataverse_files/HAM10000_images_pca"
x = '_3'

train_all_df = augmentention_on_dataset(train_df, train_df_unique, 'train_df',
input_path, 1100, 1000, x)
val_all_df = augmentention_on_dataset(val_df_unique, val_df_unique, 'val_df',
input_path, 80, 70, x)
train_all_df.to_csv(f'dataverse_files/HAM10000_metadata_augmented'+x+'_' + 'train'
+'.csv', index=False)
val_all_df.to_csv(f'dataverse_files/HAM10000_metadata_augmented'+x+'_'  + 'val'
+'.csv', index=False)
```

```
train_all_df =
pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_train.csv')
val_all_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_val.csv')


# %%
# Extracting All Image - Augmented Images - Unique Images - One of Each Lesion Images

train_df_copy = train_df.copy()
train_df_copy['image_id'] = train_df_copy['image_id'].astype(str) + '.jpg'

train_augmented_df =
train_all_df[~train_all_df['image_id'].isin(train_df_copy['image_id'])]
train_unique =
train_all_df[~train_all_df['image_id'].isin(train_augmented_df['image_id'])]

df_one_of_each_augmented = train_unique.copy()
df_extra_unique = df_extra['lesion_id'].unique()
# how to get one image of each lesion_id
for i in df_extra_unique:
   df_one_of_each_augmented =
df_one_of_each_augmented.drop(df_one_of_each_augmented[df_one_of_each_augmented['lesio
n_id'] == i].index[1:])

list11 = train_all_df.shape[0], train_augmented_df.shape[0], train_unique.shape[0],
df_one_of_each_augmented.shape[0]


val_df_unique_copy = val_df_unique.copy()
val_df_unique_copy['image_id'] = val_df_unique_copy['image_id'].astype(str) + '.jpg'
val_augmented_df =
val_all_df[~val_all_df['image_id'].isin(val_df_unique_copy['image_id'])]
val_unique = val_all_df[~val_all_df['image_id'].isin(val_augmented_df['image_id'])]
list12 = val_all_df.shape[0], val_augmented_df.shape[0], val_unique.shape[0]
list13 = 'all', 'augmented', 'unique', 'one of each'
print('Training Set')
print(list13)
print(list11)

for label in labels:
   list1 = len(train_all_df[train_all_df['dx'] == label]),
len(train_augmented_df[train_augmented_df['dx'] == label]),
```

```
len(train_unique[train_unique['dx'] == label]),
len(df_one_of_each_augmented[df_one_of_each_augmented['dx'] == label])
   space = ' '
   print(label,(5-len(label))*space ,list1)
print('\n')
print('Validation Set')
list14 = 'all', 'augmented', 'unique'
print(list14)
print(list12)
for label in labels:
   list2 = len(val_all_df[val_all_df['dx'] ==
label]),len(val_augmented_df[val_augmented_df['dx'] == label]),
len(val_unique[val_unique['dx'] == label])
   space = ' '
   print(label,(5-len(label))*space , list2)

# %%
# Plotting class distribution for lesion_id and image_id after rebalancing the class
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 3))
bins = np.linspace(0 - .25, len(labels)-1 + .25, 2*len(labels))

# Plotting class distribution for training set
train_hist1 = ax1.hist(train_all_df['dx'].sort_values(), bins=bins,
color='tab:purple')
train_hist2 = ax1.hist(train_unique['dx'].sort_values(), bins=bins, color='tab:blue')
train_hist3 = ax1.hist(df_one_of_each_augmented['dx'].sort_values(), bins=bins,
color='tab:orange')
ax1.set_title("Class Distribution for Training Set")
ax1.set_xlabel('Class Labels')
ax1.set_ylabel('Number of Images')
# Plotting class distribution for validation set
val_hist1 = ax2.hist(val_all_df['dx'].sort_values(), bins=bins, color='tab:purple')
val_hist2 = ax2.hist(val_unique['dx'].sort_values(), bins=bins, color='tab:orange')
ax2.set_title("Class Distribution for Validation Set")
ax2.set_xlabel('Class Labels')
ax2.set_ylabel('Number of Images')
# Set legend colors
ax1.legend([train_hist1[2][0], train_hist2[2][0], train_hist3[2][0]], ['Augmented
Images', 'Extra Images of Same Lesions', 'Unique Images'])
ax2.legend([val_hist1[2][0], val_hist2[2][0]], ['Augmented Images', 'Unique Images'])

plt.tight_layout()
```

```
#plt.savefig('class_distribution_augmented.png')
plt.show()
```

Appendix C: CNN Code

```
# %%
import os
import datetime
import random # random.seed(42)
import warnings
import numpy as np # np.random.seed(42)
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize,LinearSegmentedColormap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import sklearn.exceptions
from sklearn.utils import class_weight
warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)

from tensorflow.config.experimental import enable_op_determinism #
enable_op_determinism()
from tensorflow.random import set_seed # set_seed(42)
from tensorflow.python.client import device_lib

from keras import backend as K
from keras.utils import load_img, img_to_array, set_random_seed # set_random_seed(42)
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Flatten, Conv2D, BatchNormalization,
Activation, MaxPooling2D
from keras.layers import Input, GlobalAveragePooling2D, ZeroPadding2D
from keras.models import Model, Sequential
from keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras import applications
finish_sound = "afplay /Users/mehmet/Documents/vs-code/winsquare.mp3"
```

```
# !conda install -y -n ml ipykernel=6.23.2 numpy==1.24.0 matplotlib=3.7.1 pandas=2.0.2
seaborn=0.12.1 scikit-learn=1.3.2 tensorflow=2.11.1
# !jupyter nbconvert --to html skin-cancer-cnn.ipynb
[x.name for x in device_lib.list_local_devices()]

# %%
# Train and test data paths
train_path = "dataverse_files/HAM10000_images_pca"
test_path = "dataverse_files/ISIC2018_Task3_Test_Images"

# Read the data
df = pd.read_csv('dataverse_files/HAM10000_metadata.csv')
df_test = pd.read_csv('dataverse_files/ISIC2018_Task3_Test_GroundTruth.csv')

# Delete df and vasc classes
df = df[df['dx'] != 'vasc']
df = df[df['dx'] != 'df']
df_test = df_test[df_test['dx'] != 'vasc']
df_test = df_test[df_test['dx'] != 'df']

labels = df['dx'].sort_values().unique()

# Add .jpg to image_id column
df['image_id'] = df['image_id'].astype(str) + '.jpg'
df_test['image_id'] = df_test['image_id'].astype(str) + '.jpg'

# Drop unused columns
# df=df.drop(['lesion_id',  'dx_type', 'age', 'sex', 'localization', 'dataset'],
axis=1)
# df_test=df_test.drop(['lesion_id',  'dx_type', 'age', 'sex', 'localization',
'dataset'], axis=1)
df=df.drop(['lesion_id', 'dataset'], axis=1)
df_test=df_test.drop(['lesion_id', 'dataset'], axis=1)

# 'ISIC_0035068.jpg' is missing in the test set file, lets remove it from test set
dataframe
df_test = df_test[df_test['image_id'] != 'ISIC_0035068.jpg']

print(labels,'\n')

df.sort_values(by=['image_id'], inplace=True)
df.reset_index(inplace=True, drop=True)
```

```
df_test.sort_values(by=['image_id'], inplace=True)
df_test.reset_index(inplace=True, drop=True)

train_df, val_df=train_test_split(df, train_size=0.9, shuffle=True, random_state=123,
stratify=df['dx'])
train_df.reset_index(inplace=True, drop=True)
val_df.reset_index(inplace=True, drop=True)
test_df = df_test.copy().sample(frac=1, random_state=123).reset_index(drop=True) #
shuffle test set

# %%
# #To use augmented data
# x = '_3'
# train_path = 'dataverse_files/HAM10000_images_pca_augmented'+x
# train_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_train.csv')
# val_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_val.csv')

# %%
for label in labels:
   list1 = len(train_df[train_df['dx'] == label]), len(val_df[val_df['dx'] == label]),
len(test_df[test_df['dx'] == label])
   space = ' '
   print(label,(5-len(label))*space ,list1)

train_df.shape, val_df.shape, test_df.shape

# %%
rescale=1./255
color_mode = 'rgb'
target_size = (32, 32)
batch_size  = 64
# 600 x 450
train_data_np = np.array([img_to_array(load_img(train_path+'/'+img,
target_size=target_size)) for img in train_df['image_id'].values.tolist()])

datagen = ImageDataGenerator(rescale=rescale,
                             featurewise_center=True,
                             featurewise_std_normalization=True)
datagen.fit(train_data_np)
train_set = datagen.flow_from_dataframe(train_df,
                                        directory=train_path,
                                        x_col="image_id",
```

```
                                        y_col="dx",
                                        color_mode=color_mode,
                                        target_size=target_size,
                                        batch_size=batch_size,
                                        class_mode='categorical',
                                        shuffle=False
                                        )

val_set = datagen.flow_from_dataframe(val_df,
                                      directory=train_path,
                                      x_col="image_id",
                                      y_col="dx",
                                      color_mode=color_mode,
                                      target_size=target_size,
                                      batch_size=batch_size,
                                      class_mode='categorical',
                                      shuffle=False
                                      )
test_set = datagen.flow_from_dataframe(test_df,
                                       directory=test_path,
                                       x_col="image_id",
                                       y_col="dx",
                                       color_mode=color_mode,
                                       target_size=target_size,
                                       batch_size=batch_size,
                                       class_mode='categorical',
                                       shuffle=False
                                       )

# %%
no_of_classes = len(labels)

model = Sequential()

#1st CNN layer
model.add(Conv2D(filters=32,
                 kernel_size=(3,3),
                 input_shape=(target_size[0],target_size[1],3)
                 ))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (3,3)))
```

```
model.add(Dropout(0.25))

#2nd CNN layer
model.add(Conv2D(filters=64,
                 kernel_size=(5,5),
                ))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (3,3)))
model.add(Dropout(0.25))


# Passing it to a Fully Connected layer
model.add(Flatten())


# Fully Connected layer
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully Connected layer
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.25))

#Last Layer
model.add(Dense(no_of_classes, activation='softmax'))


#model.add(GlobalAveragePooling2D()) # for last cnn layer before flatten

#model.add(BatchNormalization()) # can be used for all layers except output layer
(better for cnn layers)

#model.add(Dropout(0.25)) # after activation

model.summary()

# %%
def loss_plot(model, history, now, save=True):
   # convert the history.history dict to a pandas DataFrame:
```

```
    if type(history) is not pd.DataFrame:
        history = pd.DataFrame(history)
    if save == True:
        hist_csv_file = f'model-comparison/{now}/history.csv'
        with open(hist_csv_file, mode='w') as f:
            history.to_csv(f)
    epochs = range(1, history.shape[0]+1)
    plt.figure(figsize=(5, 2))
    plt.plot(epochs, history['accuracy'], label='Accuracy')
    plt.plot(epochs, history['val_accuracy'], label='Validation Accuracy')
    max_val_acc_epoch = np.argmax(history['val_accuracy']) + 1
    max_val_acc = history['val_accuracy'][max_val_acc_epoch-1]
    label='Best Epoch = '+str(max_val_acc_epoch)+'\nVal. Acc. =
'+str((max_val_acc*100).round(2))+ '%'
    plt.plot(max_val_acc_epoch, max_val_acc, 'ro', label=label)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.xlim([0, history.shape[0]+0.1])
    plt.ylim([0.5, 1])
    plt.title('Training and Validation Accuracy')
    plt.legend(loc='upper left')
    if save == True:
        plt.savefig(f'model-comparison/{now}/val-acc.png')
        np.savetxt('model-
comparison/{}/{}.txt'.format(now,str((max_val_acc*100).round(2))), [max_val_acc],
fmt='%f')
        stats = str(now) + ' ' + str((max_val_acc*100).round(2)) + '\n'
        with open('model-comparison/best-models.txt', 'a') as f:
            f.write(stats)
    plt.show()

# %%
def EvaluateModel(model, test_set, str1, now, save = True):

    print('\n PREDICTING LABELS OF TEST IMAGES')
    result = model.predict(test_set)
    y_pred = np.argmax(result, axis=1)

    if save==True:
        #save y_pred to csv file
        os.mkdir('model-comparison/'+now+'/'+str1)
```

```
        np.savetxt('model-comparison/{}/{}/pred.csv'.format(now,str1), y_pred,
delimiter=',', fmt='%d')

    y_true = test_set.classes # List containing true labels for each image.

    # Understanding classification power of model on each class
    report = classification_report(y_true, y_pred,
target_names=test_set.class_indices.keys())
    report_d = pd.DataFrame(classification_report(y_true, y_pred, output_dict=True,
target_names=test_set.class_indices.keys())).transpose()
    report_d['support']['accuracy'] = report_d['support']['macro avg']

    annot = report_d.copy()
    annot.iloc[:, 0:3] = (annot.iloc[:, 0:3]*100).applymap('{:.2f}'.format) + ' %'
    annot.iloc[7, 1] = ''
    annot.iloc[7, 0] = ''
    annot['support'] = annot['support'].astype(int)

    # how to save report as image
    norm = Normalize(-1,1)
    cmap = LinearSegmentedColormap.from_list("", [[norm(-1.0), "white"],[norm( 1.0),
"white"]])
    plot = sns.heatmap(report_d, annot=annot, cmap=cmap, cbar=False, fmt='')
    fig = plot.get_figure()
    if save==True:
        fig.savefig('model-comparison/{}/{}/report.png'.format(now,str1))

    f1_score = ((report_d['f1-score']['weighted avg']*100000//10)/100)
    accuracy = ((report_d['f1-score']['accuracy']*100000//10)/100)
    print('\nAccuracy of model prediction is: {:.2f} %'.format(accuracy))
    print('\nF1-score of model prediction is: {:.2f} %'.format(f1_score))

    cm = confusion_matrix(y_true, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=test_set.class_indices.keys()
                                  )
    disp.plot(cmap='Reds')
    disp.ax_.set_title('Confusion Matrix')
    plt.show()
    if save==True:
        disp.figure_.savefig('model-comparison/{}/{}/cm.png'.format(now,str1))
```

```
# %%
def train_new_model(model):
   # Extra
   #class_weights =
class_weight.compute_class_weight(class_weight='balanced',classes=np.unique(train_set.
classes), y=train_set.classes)
   #class_weights_dict=dict(zip(np.unique(train_set.classes),class_weights))
   #keras.utils.set_random_seed(42)
   # inside model.fit: class_weight=class_weights_dict,
   # class_weights =
class_weight.compute_class_weight(class_weight='balanced',classes=np.unique(train_set.
classes), y=train_set.classes)
   # class_weights_dict=dict(zip(np.unique(train_set.classes),class_weights))
   # class_weights_dict

   # give number of each class in train set
   # unique, counts = np.unique(train_set.classes, return_counts=True)
   # dict(zip(unique, counts))

   # class_weights_dict = {0 : 1.0,
   #                       1 : 1.0,
   #                       2 : 1.0,
   #                       3 : 6.0,
   #                       4 : 1.0}
   # # 0: akiec, 1: bcc, 2: bkl, 3: mel, 4: nv
   # class_weights_dict
   # Train new model and evaluate
   now = datetime.datetime.now().strftime("%d-%m-%H-%M")
   os.mkdir('model-comparison/'+now)
   def myprint(s):
       with open(f'model-comparison/{now}/modelsummary.txt','a') as f:
           print(s, file=f)
   model.summary(print_fn=myprint)
   with open('model-comparison/last.txt', 'w') as f:
       f.write(str(now))
   return now

# %%
# Train new model and evaluate
now = train_new_model(model)
```

```
# Define the optimizer
optimizer = Adam(learning_rate=0.01*(batch_size/256), beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=False)
#optimizer = SGD(learning_rate=0.01*(batch_size/256), momentum=0.9, nesterov=True)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', patience=3, verbose=1,
factor=0.5, min_lr=0.0000001)
checkpoint = ModelCheckpoint(f"model-comparison/{now}/model.h5",
monitor='val_accuracy', save_best_only=True, mode='max')
early_stop = EarlyStopping(monitor='val_accuracy', patience=20, mode='max',
restore_best_weights=True)

history = model.fit(train_set,
                    epochs=50,
                    validation_data = val_set,
                    callbacks=[reduce_lr, checkpoint, early_stop],
                    )

loss_plot(model, history.history, now)

model = load_model(f"model-comparison/{now}/model.h5")
EvaluateModel(model, val_set, 'val', now)
EvaluateModel(model, test_set, 'test', now)

os.system(finish_sound)
```

Appendix D: Resnet Code

```
# %%
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.colors import Normalize,LinearSegmentedColormap
import pandas as pd

# %%
```

```
import seaborn as sns
from collections import Counter
import datetime
import os
import torch
import torch.nn as nn
from torchvision import datasets,transforms,models
from torch.utils.data import Dataset,DataLoader
from torch.optim import lr_scheduler
from torchsummary import summary


from PIL import Image

import sys
import torch.optim as optim

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
f1_score, recall_score, classification_report, ConfusionMatrixDisplay
os.environ['KMP_DUPLICATE_LIB_OK']='True'

finish_sound = "afplay /Users/mehmet/Documents/vs-code/winsquare.mp3"

# %%
"""
# Dataset manipulation
# For Augmented Data
csv_train = 'dataverse_files/HAM10000_metadata_augmented_3_train.csv'
csv_val = 'dataverse_files/HAM10000_metadata_augmented_3_val.csv'
# # For Base Data
# csv_train = 'dataverse_files/HAM10000_metadata_train_split.csv'
# csv_val = 'dataverse_files/HAM10000_metadata_val_split.csv'
csv_test = 'dataverse_files/ISIC2018_Task3_Test_GroundTruth_augmented.csv'

# Define your label encoder and decoder
encoder_decoder = {'akiec': 0, 'bcc': 1, 'bkl': 2, 'mel': 3, 'nv': 4}
label_encoder = LabelEncoder()
label_encoder.classes_ = list(encoder_decoder.keys())

# Function to encode labels in a DataFrame
def encode_labels(df, column_name):
```

```
    df[column_name] = df[column_name].map(encoder_decoder)
    return df

# Encode labels in the training set
train_df = pd.read_csv(csv_train)
train_df = encode_labels(train_df, 'dx')
train_df.to_csv('dataverse_files/encoded_train.csv', index=False)

# Encode labels in the validation set
val_df = pd.read_csv(csv_val)
val_df = encode_labels(val_df, 'dx')
val_df.to_csv('dataverse_files/encoded_val.csv', index=False)

# Encode labels in the test set
test_df = pd.read_csv(csv_test)
test_df = encode_labels(test_df, 'dx')
test_df.to_csv('dataverse_files/encoded_test.csv', index=False) """

# %%
train_val_folder = 'dataverse_files/HAM10000_images_pca_augmented_3_resnet'
# # For Base Data
# train_val_folder = 'dataverse_files/HAM10000_images_pca_base_resnet'
test_folder = 'dataverse_files/ISIC2018_Task3_Test_Images_resnet'
csv_train = 'dataverse_files/encoded_train.csv'
csv_val = 'dataverse_files/encoded_val.csv'
csv_test = 'dataverse_files/encoded_test.csv'

class CustomDataset(Dataset):
    def __init__(self, csv_file, image_folder, transform=None):
        self.data = pd.read_csv(csv_file)
        self.image_folder = image_folder
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        img_name = f'{self.image_folder}/{self.data.iloc[idx, 1]}'  # Use index 1 for
image paths
        image = Image.open(img_name).convert('RGB')

        label = int(self.data.iloc[idx, 2])  # Use index 1 for image labels
```

```
        if self.transform:
            image = self.transform(image)

        return image, label

# Define transformations
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

# Create instances of custom datasets
train_dataset = CustomDataset(csv_file=csv_train, image_folder=train_val_folder,
transform=transform)
val_dataset = CustomDataset(csv_file=csv_val, image_folder=train_val_folder,
transform=transform)
test_dataset = CustomDataset(csv_file=csv_test, image_folder=test_folder,
transform=transform)

# You can create DataLoader instances for each dataset
batch_size = 32
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Check dataset sizes
print(f'Train dataset size: {len(train_dataset)}')
print(f'Validation dataset size: {len(val_dataset)}')
print(f'Test dataset size: {len(test_dataset)}')

# Check number of classes
print(f'Number of classes: {len(train_dataset.data.dx.unique())}')
print(f'Number of classes: {len(val_dataset.data.dx.unique())}')
print(f'Number of classes: {len(test_dataset.data.dx.unique())}')


# Check number of images per class in train, val and test sets
print(train_dataset.data.dx.value_counts())
print(val_dataset.data.dx.value_counts())
print(test_dataset.data.dx.value_counts())
```

```
# %%
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride = 1, downsample = None):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Sequential(
                        nn.Conv2d(in_channels, out_channels, kernel_size = 3, stride =
stride, padding = 1),
                        nn.BatchNorm2d(out_channels),
                        nn.ReLU())
        self.conv2 = nn.Sequential(
                        nn.Conv2d(out_channels, out_channels, kernel_size = 3, stride =
1, padding = 1),
                        nn.BatchNorm2d(out_channels))
        self.downsample = downsample
        self.relu = nn.ReLU()
        self.out_channels = out_channels

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.conv2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out

# %%
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes = 3):
        super(ResNet, self).__init__()
        self.inplanes = 64
        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 64, kernel_size = 7, stride = 2, padding = 3),
                        nn.BatchNorm2d(64),
                        nn.ReLU())
        self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1)
        self.layer0 = self._make_layer(block, 64, layers[0], stride = 1)
        self.layer1 = self._make_layer(block, 128, layers[1], stride = 2)
        self.layer2 = self._make_layer(block, 256, layers[2], stride = 2)
        self.layer3 = self._make_layer(block, 512, layers[3], stride = 2)
```

```
        self.avgpool = nn.AvgPool2d(7, stride=1)
        self.fc = nn.Linear(512, num_classes)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        if stride != 1 or self.inplanes != planes:

            downsample = nn.Sequential(
                nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=stride),
                nn.BatchNorm2d(planes),
            )
        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes
        for i in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)


    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return x

# %%
device = torch.device("cuda:0" if torch.cuda.is_available()
                              else "cpu")

# %%
class BasicBlock(nn.Module):
    expansion = 1
```

```
    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super().__init__()
        self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=3, stride=stride,
                               padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1,
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out

# %%
def _make_layer(block, inplanes,planes, blocks, stride=1):
    downsample = None
    if stride != 1 or inplanes != planes:
        downsample = nn.Sequential(
            nn.Conv2d(inplanes, planes, 1, stride, bias=False),
            nn.BatchNorm2d(planes),
        )
    layers = []
    layers.append(block(inplanes, planes, stride, downsample))
    inplanes = planes
    for _ in range(1, blocks):
```

```
        layers.append(block(inplanes, planes))
    return nn.Sequential(*layers)

# %%
layers=[3, 4, 6, 3]

# %%
layer1 =_make_layer(BasicBlock, inplanes=64,planes=64, blocks=layers[0])
layer1

# %%
layer2 = _make_layer(BasicBlock, 64, 128, layers[1], stride=2)
layer2

# %%
class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=5):
        super().__init__()

        self.inplanes = 64

        self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2, padding=3,
                               bias=False)
        self.bn1 = nn.BatchNorm2d(self.inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 , num_classes)


    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None

        if stride != 1 or self.inplanes != planes:
            downsample = nn.Sequential(
```

```
                nn.Conv2d(self.inplanes, planes, 1, stride, bias=False),
                nn.BatchNorm2d(planes),
            )

        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))

        self.inplanes = planes

        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes))

        return nn.Sequential(*layers)


    def forward(self, x):
        x = self.conv1(x)           # 224x224
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)         # 112x112

        x = self.layer1(x)          # 56x56
        x = self.layer2(x)          # 28x28
        x = self.layer3(x)          # 14x14
        x = self.layer4(x)          # 7x7

        x = self.avgpool(x)         # 1x1
        x = torch.flatten(x, 1)     # remove 1 X 1 grid and make vector of tensor shape
        x = self.fc(x)

        return x

# %%
model = ResNet(block = BasicBlock, layers = layers).to(device)
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.001,momentum=0.9)
# Model summary resnet
summary(model, (3, 32, 32))

# %%
# TRAIN MODEL
```

```
now = datetime.datetime.now().strftime("%d-%m-%H-%M")
os.mkdir('model-comparison/'+now)

n_epochs = 15
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

for epoch in range(1, n_epochs + 1):
   running_loss = 0.0
   correct_train = 0
   total_train = 0
   model.train()

   for i, (inputs, labels) in enumerate(train_dataloader):
       if torch.cuda.is_available():
           inputs, labels = inputs.cuda(), labels.cuda()

       inputs = inputs.to(device)
       labels = labels.to(device)

       optimizer.zero_grad()
       outputs = model(inputs)
       loss = criterion(outputs, labels)
       loss.backward()
       optimizer.step()

       running_loss += loss.item()

       _, predicted = outputs.max(1)
       total_train += labels.size(0)
       correct_train += predicted.eq(labels).sum().item()

       if i % 20 == 19:
           #print("Epoch {}, Batch {}, Training Loss: {:.4f}".format(epoch, i + 1,
running_loss / 20))
           running_loss = 0.0

   # Calculate training accuracy
   train_accuracy = 100 * correct_train / total_train
   train_losses.append(running_loss / len(train_dataloader))
```

```
    train_accuracies.append(train_accuracy)

    # Validation
    model.eval()
    val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for inputs, labels in val_dataloader:
            if torch.cuda.is_available():
                inputs, labels = inputs.cuda(), labels.cuda()

            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()

            _, predicted = outputs.max(1)
            total_val += labels.size(0)
            correct_val += predicted.eq(labels).sum().item()

    # Calculate validation accuracy
    val_accuracy = 100 * correct_val / total_val
    val_losses.append(val_loss / len(val_dataloader))
    val_accuracies.append(val_accuracy)

    print("Epoch {}, Training Loss: {:.4f}, Training Accuracy: {:.2f}%, Validation
Loss: {:.4f}, Validation Accuracy: {:.2f}%".format(
        epoch, train_losses[-1], train_accuracies[-1], val_losses[-1], val_accuracies[-
1]))

print('\nFinished Training')

# Save model
torch.save(model.state_dict(), f'model-comparison/{now}/resnet_model.pt')

# Save history
history = pd.DataFrame({
    'train_loss': train_losses,
```

```
    'train_accuracy': train_accuracies,
    'val_loss': val_losses,
    'val_accuracy': val_accuracies
})
history.to_csv(f'model-comparison/{now}/history.csv', index=False)

# %%
def loss_plot(history, now, save=True):

    epochs = range(1, history.shape[0]+1)
    plt.figure(figsize=(5, 2))
    plt.plot(epochs, history['train_accuracy']/100, label='Accuracy')
    plt.plot(epochs, history['val_accuracy']/100, label='Validation Accuracy')
    max_val_acc_epoch = np.argmax(history['val_accuracy']) + 1
    max_val_acc = history['val_accuracy'][max_val_acc_epoch-1]/100
    label='Best Epoch = '+str(max_val_acc_epoch)+'\nVal. Acc. =
'+str((max_val_acc*100).round(2))+ '%'
    plt.plot(max_val_acc_epoch, max_val_acc, 'ro', label=label)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.xlim([0, history.shape[0]+0.1])
    plt.ylim([0.5, 1])
    plt.title('Training and Validation Accuracy')
    plt.legend(loc='upper left')
    if save == True:
        plt.savefig(f'model-comparison/{now}/val-acc.png')
        np.savetxt('model-
comparison/{}/{}.txt'.format(now,str((max_val_acc*100).round(2))), [max_val_acc],
fmt='%f')
        stats = str(now) + ' ' + str((max_val_acc*100).round(2)) + '\n'
        with open('model-comparison/best-models.txt', 'a') as f:
            f.write(stats)
    plt.show()

# %%
def EvaluateModel(y_pred, y_true, target_names, str1, now, save = True):


    print(f'\n PREDICTING LABELS OF {str1} IMAGES')
    y_pred = y_pred

    if save==True:
```

```
        #save y_pred to csv file
        os.mkdir('model-comparison/'+now+'/'+str1)
        np.savetxt('model-comparison/{}/{}/pred.csv'.format(now,str1), y_pred,
delimiter=',', fmt='%d')

    y_true = y_true

    # Understanding classification power of model on each class
    report = classification_report(y_true, y_pred, target_names=target_names)
    report_d = pd.DataFrame(classification_report(y_true, y_pred, output_dict=True,
target_names=target_names)).transpose()
    report_d['support']['accuracy'] = report_d['support']['macro avg']

    annot = report_d.copy()
    annot.iloc[:, 0:3] = (annot.iloc[:, 0:3]*100).applymap('{:.2f}'.format) + ' %'
    annot.iloc[7, 1] = ''
    annot.iloc[7, 0] = ''
    annot['support'] = annot['support'].astype(int)

    # how to save report as image
    norm = Normalize(-1,1)
    cmap = LinearSegmentedColormap.from_list("", [[norm(-1.0), "white"],[norm( 1.0),
"white"]])
    plot = sns.heatmap(report_d, annot=annot, cmap=cmap, cbar=False, fmt='')
    fig = plot.get_figure()
    if save==True:
        fig.savefig('model-comparison/{}/{}/report.png'.format(now,str1))

    f1_score = ((report_d['f1-score']['weighted avg']*100000//10)/100)
    accuracy = ((report_d['f1-score']['accuracy']*100000//10)/100)
    print('\nAccuracy of model prediction is: {:.2f} %'.format(accuracy))
    print('\nF1-score of model prediction is: {:.2f} %'.format(f1_score))

    cm = confusion_matrix(y_true, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=target_names
                                  )
    disp.plot(cmap='Reds')
    disp.ax_.set_title('Confusion Matrix')
    plt.show()
    if save==True:
```

```
        disp.figure_.savefig('model-comparison/{}/{}/cm.png'.format(now,str1))

# %%
loss_plot(history, now)

# %%
# VALIDATION RESULTS

with torch.no_grad():
    ground_truth = []
    prediction = []
    for images, labels in val_dataloader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)

        _, predicted = torch.max(outputs, 1)
        ground_truth.append(labels.cpu().tolist())
        prediction.append(predicted.cpu().tolist())

# Concatenate lists into single list
ground_truth = [item for sublist in ground_truth for item in sublist]
prediction = [item for sublist in prediction for item in sublist]

target_names = ['akiec', 'bcc', 'bkl', 'mel', 'nv']
EvaluateModel(prediction, ground_truth, target_names, 'VALIDATION', now=now, save =
True)

# %%
# TEST RESULTS

with torch.no_grad():
    ground_truth = []
    prediction = []
    for images, labels in test_dataloader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)

        _, predicted = torch.max(outputs, 1)
        ground_truth.append(labels.cpu().tolist())
        prediction.append(predicted.cpu().tolist())
```

```
# Concatenate lists into single list
ground_truth = [item for sublist in ground_truth for item in sublist]
prediction = [item for sublist in prediction for item in sublist]

target_names = ['akiec', 'bcc', 'bkl', 'mel', 'nv']
EvaluateModel(prediction, ground_truth, target_names, 'TEST', now=now, save = True)
```

Appendix E: Transfer Learning

```
# %%
import os
import datetime
import random # random.seed(42)
import warnings
import numpy as np # np.random.seed(42)
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize,LinearSegmentedColormap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import sklearn.exceptions
from sklearn.utils import class_weight
warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)

from tensorflow.config.experimental import enable_op_determinism #
enable_op_determinism()
from tensorflow.random import set_seed # set_seed(42)
from tensorflow.python.client import device_lib

from keras import backend as K
from keras.utils import load_img, img_to_array, set_random_seed # set_random_seed(42)
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Flatten, Conv2D, BatchNormalization,
Activation, MaxPooling2D
```

```
from keras.layers import Input, GlobalAveragePooling2D, ZeroPadding2D
from keras.models import Model, Sequential
from keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras import applications
finish_sound = "afplay /Users/mehmet/Documents/vs-code/winsquare.mp3"
# !conda install -y -n ml ipykernel=6.23.2 numpy==1.24.0 matplotlib=3.7.1 pandas=2.0.2
seaborn=0.12.1 scikit-learn=1.3.2 tensorflow=2.11.1
# !jupyter nbconvert --to html skin-cancer-cnn.ipynb
[x.name for x in device_lib.list_local_devices()]

# %%
# Train and test data paths
train_path = "dataverse_files/HAM10000_images_pca"
test_path = "dataverse_files/ISIC2018_Task3_Test_Images"

# Read the data
df = pd.read_csv('dataverse_files/HAM10000_metadata.csv')
df_test = pd.read_csv('dataverse_files/ISIC2018_Task3_Test_GroundTruth.csv')

# Delete df and vasc classes
df = df[df['dx'] != 'vasc']
df = df[df['dx'] != 'df']
df_test = df_test[df_test['dx'] != 'vasc']
df_test = df_test[df_test['dx'] != 'df']

labels = df['dx'].sort_values().unique()

# Add .jpg to image_id column
df['image_id'] = df['image_id'].astype(str) + '.jpg'
df_test['image_id'] = df_test['image_id'].astype(str) + '.jpg'

# Drop unused columns
# df=df.drop(['lesion_id',  'dx_type', 'age', 'sex', 'localization', 'dataset'],
axis=1)
# df_test=df_test.drop(['lesion_id',  'dx_type', 'age', 'sex', 'localization',
'dataset'], axis=1)
df=df.drop(['lesion_id', 'dataset'], axis=1)
df_test=df_test.drop(['lesion_id', 'dataset'], axis=1)

# 'ISIC_0035068.jpg' is missing in the test set file, lets remove it from test set
dataframe
```

```
df_test = df_test[df_test['image_id'] != 'ISIC_0035068.jpg']

print(labels,'\n')

df.sort_values(by=['image_id'], inplace=True)
df.reset_index(inplace=True, drop=True)
df_test.sort_values(by=['image_id'], inplace=True)
df_test.reset_index(inplace=True, drop=True)

train_df, val_df=train_test_split(df, train_size=0.9, shuffle=True, random_state=123,
stratify=df['dx'])
train_df.reset_index(inplace=True, drop=True)
val_df.reset_index(inplace=True, drop=True)
test_df = df_test.copy().sample(frac=1, random_state=123).reset_index(drop=True) #
shuffle test set

# %%
# #To use augmented data
# x = '_3'
# train_path = 'dataverse_files/HAM10000_images_pca_augmented'+x
# train_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_train.csv')
# val_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_val.csv')

# %%
for label in labels:
   list1 = len(train_df[train_df['dx'] == label]), len(val_df[val_df['dx'] == label]),
len(test_df[test_df['dx'] == label])
   space = ' '
   print(label,(5-len(label))*space ,list1)

train_df.shape, val_df.shape, test_df.shape

# %%
rescale=1./255
color_mode = 'rgb'
target_size = (32, 32)
batch_size  = 64
# 600 x 450
train_data_np = np.array([img_to_array(load_img(train_path+'/'+img,
target_size=target_size)) for img in train_df['image_id'].values.tolist()])

datagen = ImageDataGenerator(rescale=rescale,
```

```
                            featurewise_center=True,
                            featurewise_std_normalization=True)
datagen.fit(train_data_np)
train_set = datagen.flow_from_dataframe(train_df,
                                        directory=train_path,
                                        x_col="image_id",
                                        y_col="dx",
                                        color_mode=color_mode,
                                        target_size=target_size,
                                        batch_size=batch_size,
                                        class_mode='categorical',
                                        shuffle=False
                                        )

val_set = datagen.flow_from_dataframe(val_df,
                                      directory=train_path,
                                      x_col="image_id",
                                      y_col="dx",
                                      color_mode=color_mode,
                                      target_size=target_size,
                                      batch_size=batch_size,
                                      class_mode='categorical',
                                      shuffle=False
                                      )
test_set = datagen.flow_from_dataframe(test_df,
                                       directory=test_path,
                                       x_col="image_id",
                                       y_col="dx",
                                       color_mode=color_mode,
                                       target_size=target_size,
                                       batch_size=batch_size,
                                       class_mode='categorical',
                                       shuffle=False
                                       )

# %%
def loss_plot(model, history, now, save=True):
   # convert the history.history dict to a pandas DataFrame:
   if type(history) is not pd.DataFrame:
       history = pd.DataFrame(history)
   if save == True:
       hist_csv_file = f'model-comparison/{now}/history.csv'
```

```
        with open(hist_csv_file, mode='w') as f:
            history.to_csv(f)
    epochs = range(1, history.shape[0]+1)
    plt.figure(figsize=(5, 2))
    plt.plot(epochs, history['accuracy'], label='Accuracy')
    plt.plot(epochs, history['val_accuracy'], label='Validation Accuracy')
    max_val_acc_epoch = np.argmax(history['val_accuracy']) + 1
    max_val_acc = history['val_accuracy'][max_val_acc_epoch-1]
    label='Best Epoch = '+str(max_val_acc_epoch)+'\nVal. Acc. =
'+str((max_val_acc*100).round(2))+ '%'
    plt.plot(max_val_acc_epoch, max_val_acc, 'ro', label=label)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.xlim([0, history.shape[0]+0.1])
    plt.ylim([0.5, 1])
    plt.title('Training and Validation Accuracy')
    plt.legend(loc='upper left')
    if save == True:
        plt.savefig(f'model-comparison/{now}/val-acc.png')
        np.savetxt('model-
comparison/{}/{}.txt'.format(now,str((max_val_acc*100).round(2))), [max_val_acc],
fmt='%f')
        stats = str(now) + ' ' + str((max_val_acc*100).round(2)) + '\n'
        with open('model-comparison/best-models.txt', 'a') as f:
            f.write(stats)
    plt.show()

# %%
def EvaluateModel(model, test_set, str1, now, save = True):

    print('\n PREDICTING LABELS OF TEST IMAGES')
    result = model.predict(test_set)
    y_pred = np.argmax(result, axis=1)

    if save==True:
        #save y_pred to csv file
        os.mkdir('model-comparison/'+now+'/'+str1)
        np.savetxt('model-comparison/{}/{}/pred.csv'.format(now,str1), y_pred,
delimiter=',', fmt='%d')

    y_true = test_set.classes # List containing true labels for each image.
```

```
    # Understanding classification power of model on each class
    report = classification_report(y_true, y_pred,
target_names=test_set.class_indices.keys())
    report_d = pd.DataFrame(classification_report(y_true, y_pred, output_dict=True,
target_names=test_set.class_indices.keys())).transpose()
    report_d['support']['accuracy'] = report_d['support']['macro avg']

    annot = report_d.copy()
    annot.iloc[:, 0:3] = (annot.iloc[:, 0:3]*100).applymap('{:.2f}'.format) + ' %'
    annot.iloc[7, 1] = ''
    annot.iloc[7, 0] = ''
    annot['support'] = annot['support'].astype(int)

    # how to save report as image
    norm = Normalize(-1,1)
    cmap = LinearSegmentedColormap.from_list("", [[norm(-1.0), "white"],[norm( 1.0),
"white"]])
    plot = sns.heatmap(report_d, annot=annot, cmap=cmap, cbar=False, fmt='')
    fig = plot.get_figure()
    if save==True:
        fig.savefig('model-comparison/{}/{}/report.png'.format(now,str1))

    f1_score = ((report_d['f1-score']['weighted avg']*100000//10)/100)
    accuracy = ((report_d['f1-score']['accuracy']*100000//10)/100)
    print('\nAccuracy of model prediction is: {:.2f} %'.format(accuracy))
    print('\nF1-score of model prediction is: {:.2f} %'.format(f1_score))

    cm = confusion_matrix(y_true, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=test_set.class_indices.keys()
                                  )
    disp.plot(cmap='Reds')
    disp.ax_.set_title('Confusion Matrix')
    plt.show()
    if save==True:
        disp.figure_.savefig('model-comparison/{}/{}/cm.png'.format(now,str1))

# %%
def train_new_model(model):
    # Extra
```

```
    #class_weights =
class_weight.compute_class_weight(class_weight='balanced',classes=np.unique(train_set.
classes), y=train_set.classes)
    #class_weights_dict=dict(zip(np.unique(train_set.classes),class_weights))
    #keras.utils.set_random_seed(42)
    # inside model.fit: class_weight=class_weights_dict,

    # Train new model and evaluate
    now = datetime.datetime.now().strftime("%d-%m-%H-%M")
    os.mkdir('model-comparison/'+now)
    def myprint(s):
        with open(f'model-comparison/{now}/modelsummary.txt','a') as f:
            print(s, file=f)
    model.summary(print_fn=myprint)
    with open('model-comparison/last.txt', 'w') as f:
        f.write(str(now))
    return now

# %%
# Transfer Learning

import ssl
ssl._create_default_https_context = ssl._create_unverified_context

base_model = applications.VGG19(weights='imagenet', include_top=False,
input_shape=(target_size[0],target_size[1],3))

for layer in base_model.layers:
    layer.trainable = True

# Add layers at the end
model = base_model.output
model = Flatten()(model)

model = Dense(512, kernel_initializer='he_uniform')(model)
model = Dropout(0.2)(model)
model = BatchNormalization()(model)
model = Activation('relu')(model)

model = Dense(128, kernel_initializer='he_uniform')(model)
model = Dropout(0.2)(model)
model = BatchNormalization()(model)
```

```
model = Activation('relu')(model)

model = Dense(32, kernel_initializer='he_uniform')(model)
model = Dropout(0.2)(model)
model = BatchNormalization()(model)
model = Activation('relu')(model)

output = Dense(len(labels), activation='softmax')(model)

model = Model(inputs=base_model.input, outputs=output)

optimizer = Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
n_epoch = 10

# %%
now = train_new_model(model)

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='auto',
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, verbose=1,
mode='auto')
checkpoint = ModelCheckpoint(f"model-comparison/{now}/model.h5",
monitor='val_accuracy', save_best_only=True, mode='max')

history = model.fit(train_set,
                    epochs=30,
                    callbacks=[reduce_lr,early_stop,checkpoint],
                    validation_data=val_set,
                    shuffle=True)


# Evaluate Transfer Learning model
loss_plot(model, history.history, now)
EvaluateModel(model, val_set, 'val', now)
EvaluateModel(model, test_set, 'test', now)
os.system(finish_sound)

# %%
```

```
# Load best model of VGG19
now = '18-12-03-57'
model_loaded = load_model('model-comparison/18-12-03-57/model.h5')
history = pd.read_csv('model-comparison/18-12-03-57/history.csv')

# Evaluate Transfer Learning model
loss_plot(model_loaded, history, now)
EvaluateModel(model_loaded, val_set, 'val', now, save=False)
EvaluateModel(model_loaded, test_set, 'test', now, save=False)
os.system(finish_sound)
```

Appendix F: XGBoost Classifier

```
# %%
import os
import datetime
import random # random.seed(42)
import warnings
import numpy as np # np.random.seed(42)
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize,LinearSegmentedColormap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import sklearn.exceptions
from sklearn.utils import class_weight
warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)

from tensorflow.config.experimental import enable_op_determinism #
enable_op_determinism()
from tensorflow.random import set_seed # set_seed(42)
from tensorflow.python.client import device_lib

from keras import backend as K
from keras.utils import load_img, img_to_array, set_random_seed # set_random_seed(42)
from keras.models import load_model
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Flatten, Conv2D, BatchNormalization,
Activation, MaxPooling2D
from keras.layers import Input, GlobalAveragePooling2D, ZeroPadding2D
from keras.models import Model, Sequential
from keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras import applications
finish_sound = "afplay /Users/mehmet/Documents/vs-code/winsquare.mp3"
# !conda install -y -n ml ipykernel=6.23.2 numpy==1.24.0 matplotlib=3.7.1 pandas=2.0.2
seaborn=0.12.1 scikit-learn=1.3.2 tensorflow=2.11.1
# !jupyter nbconvert --to html skin-cancer-cnn.ipynb
[x.name for x in device_lib.list_local_devices()]

# %%
# Train and test data paths
train_path = "dataverse_files/HAM10000_images_pca"
test_path = "dataverse_files/ISIC2018_Task3_Test_Images"

# Read the data
df = pd.read_csv('dataverse_files/HAM10000_metadata.csv')
df_test = pd.read_csv('dataverse_files/ISIC2018_Task3_Test_GroundTruth.csv')

# Delete df and vasc classes
df = df[df['dx'] != 'vasc']
df = df[df['dx'] != 'df']
df_test = df_test[df_test['dx'] != 'vasc']
df_test = df_test[df_test['dx'] != 'df']

labels = df['dx'].sort_values().unique()

# Add .jpg to image_id column
df['image_id'] = df['image_id'].astype(str) + '.jpg'
df_test['image_id'] = df_test['image_id'].astype(str) + '.jpg'

# Drop unused columns
# df=df.drop(['lesion_id',  'dx_type', 'age', 'sex', 'localization', 'dataset'],
axis=1)
# df_test=df_test.drop(['lesion_id',  'dx_type', 'age', 'sex', 'localization',
'dataset'], axis=1)
df=df.drop(['lesion_id', 'dataset'], axis=1)
df_test=df_test.drop(['lesion_id', 'dataset'], axis=1)
```

```
# 'ISIC_0035068.jpg' is missing in the test set file, lets remove it from test set
dataframe
df_test = df_test[df_test['image_id'] != 'ISIC_0035068.jpg']

print(labels,'\n')

df.sort_values(by=['image_id'], inplace=True)
df.reset_index(inplace=True, drop=True)
df_test.sort_values(by=['image_id'], inplace=True)
df_test.reset_index(inplace=True, drop=True)

train_df, val_df=train_test_split(df, train_size=0.9, shuffle=True, random_state=123,
stratify=df['dx'])
train_df.reset_index(inplace=True, drop=True)
val_df.reset_index(inplace=True, drop=True)
test_df = df_test.copy().sample(frac=1, random_state=123).reset_index(drop=True) #
shuffle test set

# %%
#To use augmented data
x = '_3'
train_path = 'dataverse_files/HAM10000_images_pca_augmented'+x
train_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_train.csv')
val_df = pd.read_csv('dataverse_files/HAM10000_metadata_augmented'+x+'_val.csv')

# %%
for label in labels:
   list1 = len(train_df[train_df['dx'] == label]), len(val_df[val_df['dx'] == label]),
len(test_df[test_df['dx'] == label])
   space = ' '
   print(label,(5-len(label))*space ,list1)

train_df.shape, val_df.shape, test_df.shape

# %%
rescale=1./255
color_mode = 'rgb'
target_size = (32, 32)
batch_size  = 64
# 600 x 450
```

```
train_data_np = np.array([img_to_array(load_img(train_path+'/'+img,
target_size=target_size)) for img in train_df['image_id'].values.tolist()])

datagen = ImageDataGenerator(rescale=rescale,
                             featurewise_center=True,
                             featurewise_std_normalization=True)
datagen.fit(train_data_np)
train_set = datagen.flow_from_dataframe(train_df,
                                        directory=train_path,
                                        x_col="image_id",
                                        y_col="dx",
                                        color_mode=color_mode,
                                        target_size=target_size,
                                        batch_size=batch_size,
                                        class_mode='categorical',
                                        shuffle=False
                                        )

val_set = datagen.flow_from_dataframe(val_df,
                                      directory=train_path,
                                      x_col="image_id",
                                      y_col="dx",
                                      color_mode=color_mode,
                                      target_size=target_size,
                                      batch_size=batch_size,
                                      class_mode='categorical',
                                      shuffle=False
                                      )
test_set = datagen.flow_from_dataframe(test_df,
                                       directory=test_path,
                                       x_col="image_id",
                                       y_col="dx",
                                       color_mode=color_mode,
                                       target_size=target_size,
                                       batch_size=batch_size,
                                       class_mode='categorical',
                                       shuffle=False
                                       )

# %%
def loss_plot(model, history, now, save=True):
   # convert the history.history dict to a pandas DataFrame:
```

```
    if type(history) is not pd.DataFrame:
        history = pd.DataFrame(history)
    if save == True:
        hist_csv_file = f'model-comparison/{now}/history.csv'
        with open(hist_csv_file, mode='w') as f:
            history.to_csv(f)
    epochs = range(1, history.shape[0]+1)
    plt.figure(figsize=(5, 2))
    plt.plot(epochs, history['accuracy'], label='Accuracy')
    plt.plot(epochs, history['val_accuracy'], label='Validation Accuracy')
    max_val_acc_epoch = np.argmax(history['val_accuracy']) + 1
    max_val_acc = history['val_accuracy'][max_val_acc_epoch-1]
    label='Best Epoch = '+str(max_val_acc_epoch)+'\nVal. Acc. =
'+str((max_val_acc*100).round(2))+ '%'
    plt.plot(max_val_acc_epoch, max_val_acc, 'ro', label=label)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.xlim([0, history.shape[0]+0.1])
    plt.ylim([0.5, 1])
    plt.title('Training and Validation Accuracy')
    plt.legend(loc='upper left')
    if save == True:
        plt.savefig(f'model-comparison/{now}/val-acc.png')
        np.savetxt('model-
comparison/{}/{}.txt'.format(now,str((max_val_acc*100).round(2))), [max_val_acc],
fmt='%f')
        stats = str(now) + ' ' + str((max_val_acc*100).round(2)) + '\n'
        with open('model-comparison/best-models.txt', 'a') as f:
            f.write(stats)
    plt.show()

# %%
def EvaluateModel(model, test_set, str1, now, save = True, model_type='CNN'):

    # if file now does not exist, create it
    if not os.path.exists('model-comparison/'+now):
        os.mkdir('model-comparison/'+now)
    if model_type == 'CNN':
        print('\n PREDICTING LABELS OF TEST IMAGES')
        result = model.predict(test_set)
        y_pred = np.argmax(result, axis=1)
    else:
```

```
        y_pred = model
    if save==True:
        #save y_pred to csv file
        if not os.path.exists('model-comparison/'+now+'/'+str1):
            os.mkdir('model-comparison/'+now+'/'+str1)
        np.savetxt('model-comparison/{}/{}/pred.csv'.format(now,str1), y_pred,
delimiter=',', fmt='%d')

    y_true = test_set.classes # List containing true labels for each image.

    # Understanding classification power of model on each class
    report = classification_report(y_true, y_pred,
target_names=test_set.class_indices.keys())
    report_d = pd.DataFrame(classification_report(y_true, y_pred, output_dict=True,
target_names=test_set.class_indices.keys())).transpose()
    report_d['support']['accuracy'] = report_d['support']['macro avg']

    annot = report_d.copy()
    annot.iloc[:, 0:3] = (annot.iloc[:, 0:3]*100).applymap('{:.2f}'.format) + ' %'
    annot.iloc[7, 1] = ''
    annot.iloc[7, 0] = ''
    annot['support'] = annot['support'].astype(int)

    # how to save report as image
    norm = Normalize(-1,1)
    cmap = LinearSegmentedColormap.from_list("", [[norm(-1.0), "white"],[norm( 1.0),
"white"]])
    plot = sns.heatmap(report_d, annot=annot, cmap=cmap, cbar=False, fmt='')
    fig = plot.get_figure()
    if save==True:
        fig.savefig('model-comparison/{}/{}/report.png'.format(now,str1))

    f1_score = ((report_d['f1-score']['weighted avg']*100000//10)/100)
    accuracy = ((report_d['f1-score']['accuracy']*100000//10)/100)
    print('\nAccuracy of model prediction is: {:.2f} %'.format(accuracy))
    print('\nF1-score of model prediction is: {:.2f} %'.format(f1_score))

    cm = confusion_matrix(y_true, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=test_set.class_indices.keys()
                                  )
```

```
    disp.plot(cmap='Reds')
    disp.ax_.set_title('Confusion Matrix')
    plt.show()
    if save==True:
        disp.figure_.savefig('model-comparison/{}/{}/cm.png'.format(now,str1))

# %%
def train_new_model(model):
    # Extra
    #class_weights =
class_weight.compute_class_weight(class_weight='balanced',classes=np.unique(train_set.
classes), y=train_set.classes)
    #class_weights_dict=dict(zip(np.unique(train_set.classes),class_weights))
    #keras.utils.set_random_seed(42)
    # inside model.fit: class_weight=class_weights_dict,

    # Train new model and evaluate
    now = datetime.datetime.now().strftime("%d-%m-%H-%M")
    os.mkdir('model-comparison/'+now)
    def myprint(s):
        with open(f'model-comparison/{now}/modelsummary.txt','a') as f:
            print(s, file=f)
    model.summary(print_fn=myprint)
    with open('model-comparison/last.txt', 'w') as f:
        f.write(str(now))
    return now

# %%
# Extract features from CNN layers

name = '18-12-03-57' # VGG-19
#name = '18-12-03-36' # CNN
model = load_model(f"model-comparison/{name}/model.h5")

from tensorflow.keras.models import Model

# Create a new model that outputs the activations of a specific layer
#layer_name = 'dense_1'  # For CNN
layer_name = 'dense_2'  # For VGG-19

feature_model = Model(inputs=model.input, outputs=model.get_layer(layer_name).output)
```

```
# Use the feature model to extract features from your data
features_train = feature_model.predict(train_set)
features_val = feature_model.predict(val_set)
features_test = feature_model.predict(test_set)
features_train.shape, features_val.shape, features_test.shape

# %%
train = pd.concat([train_df, pd.DataFrame(features_train)], axis=1)
train.drop(['image_id'], axis=1, inplace=True)
val = pd.concat([val_df, pd.DataFrame(features_val)], axis=1)
val.drop(['image_id'], axis=1, inplace=True)
test = pd.concat([test_df, pd.DataFrame(features_test)], axis=1)
test.drop(['image_id'], axis=1, inplace=True)
# Drop Lesion ID and dataset columns
train.drop(['lesion_id'], axis=1, inplace=True)
train.drop(['dataset'], axis=1, inplace=True)
val.drop(['lesion_id'], axis=1, inplace=True)
val.drop(['dataset'], axis=1, inplace=True)
#test.drop(['lesion_id'], axis=1, inplace=True)

#Convert numeric dx_type
train['dx_type'] = train['dx_type'].map({'histo':0, 'consensus':1, 'confocal':2,
'follow_up':3, 'consensus':4})
val['dx_type'] = val['dx_type'].map({'histo':0, 'consensus':1, 'confocal':2,
'follow_up':3, 'consensus':4})
test['dx_type'] = test['dx_type'].map({'histo':0, 'consensus':1, 'confocal':2,
'follow_up':3, 'consensus':4})
# Fill NaN values with mode of dx_type
train['dx_type'].fillna(train['dx_type'].mode()[0], inplace=True)
val['dx_type'].fillna(val['dx_type'].mode()[0], inplace=True)
test['dx_type'].fillna(test['dx_type'].mode()[0], inplace=True)

#Convert numeric dx
train['dx'] = train['dx'].map({'akiec':0, 'bcc':1, 'bkl':2, 'mel':3, 'nv':4, 'vasc':5,
'df':6})
val['dx'] = val['dx'].map({'akiec':0, 'bcc':1, 'bkl':2, 'mel':3, 'nv':4, 'vasc':5,
'df':6})
test['dx'] = test['dx'].map({'akiec':0, 'bcc':1, 'bkl':2, 'mel':3, 'nv':4, 'vasc':5,
'df':6})

#Convert Sex feature to numeric (binary)
train['sex'] = train['sex'].map({'male':0, 'female':1})
```

```
val['sex'] = val['sex'].map({'male':0, 'female':1})
test['sex'] = test['sex'].map({'male':0, 'female':1})
#Fill NaN values with mode of sex
train['sex'].fillna(train['sex'].mode()[0], inplace=True)
val['sex'].fillna(val['sex'].mode()[0], inplace=True)
test['sex'].fillna(test['sex'].mode()[0], inplace=True)

# Convert localization to numeric
train['localization'] = train['localization'].map({'abdomen':0, 'scalp':1, 'upper
extremity':2, 'lower extremity':3, 'trunk':4, 'back':5, 'chest':6, 'face':7, 'neck':8,
'ear':9, 'unknown':10, 'hand':11, 'foot':12, 'acral':13, 'genital':14})
val['localization'] = val['localization'].map({'abdomen':0, 'scalp':1, 'upper
extremity':2, 'lower extremity':3, 'trunk':4, 'back':5, 'chest':6, 'face':7, 'neck':8,
'ear':9, 'unknown':10, 'hand':11, 'foot':12, 'acral':13, 'genital':14})
test['localization'] = test['localization'].map({'abdomen':0, 'scalp':1, 'upper
extremity':2, 'lower extremity':3, 'trunk':4, 'back':5, 'chest':6, 'face':7, 'neck':8,
'ear':9, 'unknown':10, 'hand':11, 'foot':12, 'acral':13, 'genital':14})
#Fill NaN values with mode of localization
train['localization'].fillna(train['localization'].mode()[0], inplace=True)
val['localization'].fillna(val['localization'].mode()[0], inplace=True)
test['localization'].fillna(test['localization'].mode()[0], inplace=True)

#Fill NaN values with mean of age
train['age'].fillna(train['age'].mean(), inplace=True)
val['age'].fillna(val['age'].mean(), inplace=True)
test['age'].fillna(test['age'].mean(), inplace=True)

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score

X_train = train.drop(['dx'], axis=1)
y_train = train['dx']
X_val = val.drop(['dx'], axis=1)
y_val = val['dx']
X_test = test.drop(['dx'], axis=1)
y_test = test['dx']

# To numpy array

X_train = X_train.to_numpy()
y_train = y_train.to_numpy()
X_val = X_val.to_numpy()
```

```
y_val = y_val.to_numpy()
X_test = X_test.to_numpy()
y_test = y_test.to_numpy()

# Standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# # Y one hot encoding
# from sklearn.preprocessing import OneHotEncoder
# enc = OneHotEncoder()
# enc.fit(y_train.reshape(-1, 1))
# y_train = enc.transform(y_train.reshape(-1, 1)).toarray()
# y_val = enc.transform(y_val.reshape(-1, 1)).toarray()
# y_test = enc.transform(y_test.reshape(-1, 1)).toarray()

X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shape

# %%
clf = LogisticRegression(random_state=0, max_iter=1000, solver='lbfgs',
multi_class='auto')

clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)
EvaluateModel(y_pred, val_set, 'val', name+'-LR', save=True, model_type='LR')

y_pred = clf.predict(X_test)
EvaluateModel(y_pred, test_set, 'test', name+'-LR', save=True, model_type='LR')

# %%
import xgboost as xgb
modelxg = xgb.XGBClassifier(base_score=0.5, booster='gbtree',
                                        colsample_bylevel=1,
                                        colsample_bynode=1,
                                        colsample_bytree=1, gamma=0,
                                        learning_rate=0.1, max_delta_step=0,
                                        max_depth=3, min_child_weight=1,
```

```
                                        missing=1, n_estimators=100,
                                        n_jobs=-1, nthread=None,
                                        objective='binary:logistic',
                                        random_state=1855, reg_alpha=0,
                                        reg_lambda=1, scale_pos_weight=1,
                                        seed=None, silent=None, subsample=1,
                                        verbosity=0)

modelxg.fit(X_train,y_train)

y_pred = modelxg.predict(X_val)
EvaluateModel(y_pred, val_set, 'val', name+'-XGB', save=True, model_type='XGB')

y_pred = modelxg.predict(X_test)
EvaluateModel(y_pred, test_set, 'test', name+'-XGB', save=True, model_type='XGB')
```